

ทฤษฎีการคำนวณ

Theory of Computation

- เรียนรู้ลำดับขั้นของภาษาคอมพิวเตอร์อย่างเป็นระบบ
- สร้างจักรกลรองรับภาษาประเภทต่าง ๆ
- อธิบายด้วยรูปภาพที่เข้าใจง่าย
- กดสอบความเข้าใจด้วยแบบฝึกหัดท้ายบทพร้อมเฉลย



สำนักพิมพ์ ส.ส.น.
สานักงานส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น)

ดร. เกียรติคุล เผยรนัยรุนนากิจ

สารบัญ

บทที่ 1 ความรู้พื้นฐานทางคณิตศาสตร์	1
1.1 เชต พังก์ชัน กราฟ	1
1.2 ตัวอักษร สตริง และภาษา	9
1.3 เทคนิคการพิสูจน์	11
1.4 ไวยากรณ์และอโตมาตา	17
★ บทที่ 2 ไฟโนต์อโตมาตา	23
2.1 ไฟโนต์อโตมาตา	23
2.2 ความสัมพันธ์ระหว่างภาษาและอโตมาตา	31
2.3 ไฟโนต์อโตมาตาที่คาดเดาไม่ได้	34
2.4 ความเท่าเทียมกันของ DFA และ NFA	41
บทที่ 3 นิพจน์เรกุลาร์และไวยากรณ์เรกุลาร์	53
3.1 นิพจน์เรกุลาร์	53
3.2 เทคนิคการลดรูปสถานะของไฟโนต์อโตมาตา	61
3.3 ไวยากรณ์เรกุลาร์	63
3.4 ความสัมพันธ์ระหว่างภาษาเรกุลาร์และไวยากรณ์เรกุลาร์	66
บทที่ 4 คุณสมบัติของภาษาเรกุลาร์	77
4.1 คุณสมบัติปิดของภาษาเรกุลาร์	77
4.2 คำामาพืนฐานเกี่ยวกับภาษาเรกุลาร์	81
4.3 ปีมปิงเลมมาสำหรับภาษาเรกุลาร์	85
4.4 วิธีการพิสูจน์ความไม่เป็นเรกุลาร์ของภาษาโดยใช้ปีมปิงเลมมา	86
4.5 ไอโมมอร์ฟิซึมฟังก์ชัน	93
4.6 ผลหารทางขวา	96
4.7 การประยุกต์ใช้งานภาษาเรกุลาร์และไฟโนต์อโตมาตาในตัวแปลภาษา	98
บทที่ 5 ไวยากรณ์คอนเทกซ์ฟรีและภาษาคอนเทกซ์ฟรี	101
5.1 ไวยากรณ์คอนเทกซ์ฟรี	102
5.2 วิจิวิภาคของไวยากรณ์	108
5.3 ความสัมสูน	109

5.4 การจัดรูปแบบไวยากรณ์.....	113
5.5 รูปแบบไวยากรณ์แบบซอมสกี.....	115
5.6 รูปแบบไวยากรณ์แบบกรีบนาค.....	116
5.7 ตัวแปลภาษา.....	116
บทที่ 6 พุชดาวน์ออโตมาตา.....	125
6.1 พุชดาวน์ออโตมาตาที่คาดเดาได้.....	126
6.2 พุชดาวน์ออโตมาตาที่คาดเดาไม่ได้.....	131
6.3 พุชดาวน์ออโตมาตาที่คาดเดาไม่ได้ยอมรับภาษาค่อนเทิกซ์ฟรี.....	140
6.4 ความสัมพันธ์ระหว่าง NPDA และ DPDA.....	147
บทที่ 7 การพิสูจน์ความไม่เป็นคดีของภาษาค่อนเทิกซ์ฟรีและคุณสมบัติของภาษาค่อนเทิกซ์ฟรี.....	151
7.1 คุณสมบัติปิดของภาษาค่อนเทิกซ์ฟรี.....	151
7.2 การอินเตอร์เซกชันระหว่างภาษาค่อนเทิกซ์ฟรีและภาษาเรกูลาร์.....	154
7.3 คำตามพื้นฐานเกี่ยวกับภาษาค่อนเทิกซ์ฟรี.....	159
7.4 ปั๊มปิงเลมมาสำหรับภาษาค่อนเทิกซ์ฟรี.....	160
บทที่ 8 ทั่วเริงแมชีน.....	173
8.1 ส่วนประกอบของทั่วเริงแมชีน.....	174
8.2 ภาษาที่ทั่วเริงแมชีนรองรับ.....	180
8.3 ทั่วเริงแมชีนที่คาดเดาไม่ได้.....	188
8.4 การใช้งานทั่วเริงแมชีนเพื่อทำหน้าที่ในฟังก์ชันการคำนวณ.....	191
8.5 สมมุติฐานของทั่วเริง.....	199
8.6 ทั่วเริงแมชีนที่ถูกตัดแปลง.....	199
บทที่ 9 เปื้อนหายันสูงของทฤษฎีการคำนวณ.....	203
9.1 ภาษาเวียนบังเกิดและภาษาเวียนบังเกิดที่นับได้.....	203
9.2 ไวยากรณ์ไม่จำกัดและไวยากรณ์ค่อนเทิกซ์เซนลิติฟ.....	205
9.3 ปัญหาการตัดสินใจที่แก้ได้และแก้ไม่ได้.....	207
9.4 ปัญหาแทร็กเทเบิล อินแทร็กเทเบิล และปัญหาเอ็นพี-สมบูรณ์.....	210

ภาคพนวก ก เฉลยแบบฝึกหัดท้ายบท.....	215
ภาคพนวก ข บุคคลสำคัญของกฤษฎีการคำนวณ.....	233
บรรณานุกรม.....	236
ดัชนี.....	237

ความรู้พื้นฐาน ทางคณิตศาสตร์

วิชาทฤษฎีการคำนวณ เป็นวิชาที่ต้องใช้ความรู้ทางคณิตศาสตร์คอมพิวเตอร์มาช่วยอธิบายการพิสูจน์และการแก้ปัญหาค่อนข้างมาก ดังนั้นจึงจำเป็นอย่างยิ่งที่เราจะต้องศึกษาและทราบความรู้และทฤษฎีพื้นฐานทางด้านคณิตศาสตร์คอมพิวเตอร์ นอกจากนี้ค่าสตาร์ทางด้านทฤษฎีการคำนวณประกอบด้วยสัญลักษณ์ต่าง ๆ มากมาย โดยสัญลักษณ์เหล่านี้ใช้ในการอธิบายแนวคิดและการพิสูจน์ ดังนั้นจึงจำเป็นต้องทำความเข้าใจสัญลักษณ์เหล่านี้ รวมทั้งจดจำสัญลักษณ์ เพื่อความสะดวกในการอ่านเนื้อหาในบทต่อไปของหนังสือเล่มนี้ ผู้อ่านอาจเกิดความยากลำบากเล็กน้อยในตอนเริ่มต้น เนื่องจากต้องจดจำสัญลักษณ์ค่อนข้างมาก อย่างไรก็ตาม สัญลักษณ์เหล่านี้มีจำนวนจำกัด และจะถูกอ้างถึงบ่อยครั้ง ดังนั้นผู้อ่านจะสามารถจดจำสัญลักษณ์เหล่านี้ได้โดยอัตโนมัติ

1.1 เชต พังก์เซ็ต กราฟ

1. เชต (Set) คือ กลุ่มของวัตถุโดยไม่คำนึงถึงลำดับการจัดเรียง ถ้า a เป็นสมาชิกของเชต S จะเขียนแทนด้วย $a \in S$ ในทางกลับกัน จะใช้สัญลักษณ์ \notin ในการแทนความไม่เป็นสมาชิก และแทนเชตด้วยวงเล็บปีกๆ (Curly Braces, {}) ซึ่งล้อมรอบสมาชิกต่าง ๆ ภายในเชต เช่น

$$S = \{a, b, c\} \text{ เป็นเชตของสมาชิก } a, b \text{ และ } c \text{ โดยไม่คำนึงถึงลำดับ}$$

เชตดังกล่าวถือว่าเป็นเชตที่จำกัด (Finite Set) เนื่องจากเราทราบจำนวนสมาชิกที่แน่นอนภายในเชต แต่ในบางกรณี เราไม่สามารถทราบจำนวนสมาชิกภายในเชตได้ เช่น เชตของตัวเลขที่หารด้วยสามลงตัว ซึ่งถือว่าเป็นเชตที่ไม่จำกัด (Infinite Set) ซึ่งสามารถเขียนแทนเชตนี้ได้ด้วย

$$S = \{n : n \bmod 3 = 0\} \text{ ซึ่งหมายถึง เชตของจำนวนที่หารด้วย } 3 \text{ ลงตัว}$$

$$\text{หรือ } S = \{0, 3, 6, 9, 12, \dots\}$$

กระบวนการที่กระทำกับเชต (Set Operation) คือ กระบวนการต่าง ๆ ที่สามารถกระทำกับเชตได้ โดยประกอบด้วยยูเนียน (\cup), อินเตอร์เซกชัน (\cap), ผลต่าง ($-$) และคอมพลีเมนต์ โดยมีนิยามดังต่อไปนี้

กำหนดให้ S_1 และ S_2 เป็นเชตใด ๆ

$$\text{ยูเนียน (Union)} : \quad S_1 \cup S_2 = \{a : a \in S_1 \text{ หรือ } a \in S_2\}$$

$$\text{อินเตอร์เซกชัน (Intersection)} : \quad S_1 \cap S_2 = \{a : a \in S_1 \text{ และ } a \in S_2\}$$

ผลต่าง (Difference) :

$$S_1 - S_2 = \{a : a \in S_1 \text{ และ } a \notin S_2\}$$

คอมพลีเม้นต์ (Complement) :

$$\overline{S_1} = \{a : a \notin S_1 \text{ และ } a \in \text{ยูนิเวิร์ส (Universe)}\}$$

- ตัวอย่างที่ 1.1 กำหนดให้เซต A และ B มีค่าดังต่อไปนี้

$$A = \{1, 2, 3\} \quad B = \{2, 3, 4, 5\}$$

จงหาผลลัพธ์ของการกระทำของเซตระหว่าง A และ B โดยแสดงแผนภาพเวนน์ (Venn Diagram)

- ยูเนียน

$$A \cup B = \{1, 2, 3, 4, 5\}$$

- อินเตอร์เซกชัน

$$A \cap B = \{2, 3\}$$

- ผลต่าง

$$A - B = \{1\}$$

$$B - A = \{4, 5\}$$

- คอมพลีเม้นต์

สมมุติให้ยูนิเวิร์ส (Universe) = {1, ..., 7}

$$A = \{1, 2, 3\}, \quad \overline{A} = \{4, 5, 6, 7\}$$

เซตว่าง (Empty Set) คือ เซตที่ไม่มีจำนวนสมาชิกอยู่เลย แทนด้วยสัญลักษณ์ \emptyset หรือ $\{\}$ เซตว่างมีคุณสมบัติดังต่อไปนี้

กำหนดให้ S คือ เซตใด ๆ

$$S \cup \emptyset = S$$

$$S \cap \emptyset = \emptyset$$

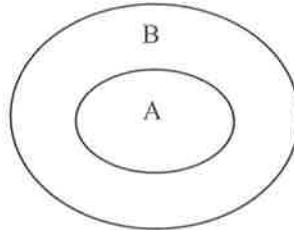
$$S - \emptyset = S$$

$$\emptyset - S = \emptyset$$

$$\bar{\emptyset} = \text{ยูนิเวิร์ส}$$

เซตย่อย (Subset) ถ้าเซต A เป็นเซตย่อยของเซต B จะถูกเขียนแทนด้วย $A \subseteq B$ ซึ่งหมายถึง สมาชิกทุกตัวภายในเซต A เป็นสมาชิกของเซต B โดยสัญลักษณ์ดังกล่าวสามารถอธิบายลึกๆ ได้ว่าสามารถอธิบายลึกๆ ได้ว่า A เท่ากับเซต B ก็ได้ หากทั้งสองเซตมีสมาชิกเหมือนกันทั้งหมด

เซตย่อยที่เหมาะสม (Proper Subset, \subset) มีนิยามที่คล้ายกับเซตย่อย แต่ต่างกันตรงที่เซตทั้งสองไม่มีโอกาสที่จะเท่ากัน เช่น $A \subset B$ สามารถแสดงได้ดังรูป



เพาเวอร์เซต (Power Set) คือ เซตของทุก ๆ เซตย่อยทั้งหมดของเซต S โดยแทนสัญลักษณ์ของเพาเวอร์เซตของ S ด้วย 2^S เช่น

$$S = \{a, b\}$$

$$2^S = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$$

เซตที่แยกจากกัน (Disjoint Set) คือ การไม่มีสมาชิกที่เหมือนกันระหว่างเซตสองเซต นั่นคือผลการอินเตอร์เซกชันระหว่างเซตทั้งสองจะต้องมีค่าเท่ากับเซตว่าง เช่น

$$A = \{1, 2, 3\}, B = \{5, 6\}$$

$$A \cap B = \emptyset$$

ขนาดของเซต (Set Cardinality) คือ จำนวนสมาชิกภายในเซต ใช้สัญลักษณ์แทนด้วยเครื่องหมาย ‘|’ เช่น

$$A = \{1, 2, 3\}$$

$$|A| = 3$$

กฎของเดอมอร์แกน (DeMorgan's Laws) เป็นกฎที่มีประโยชน์ในการแก้ปัญหาระบบของเซต มีอยู่ 2 ข้อ คือ

$$\overline{S_1 \cup S_2} = \overline{S_1} \cap \overline{S_2} = \overline{S_1} \cap \overline{S_2}$$

$$\overline{S_1 \cap S_2} = \overline{S_1} \cup \overline{S_2} = \overline{S_1} \cup \overline{S_2}$$

- ผลคูณคาร์ทีเซียน (Cartesian Product) $S_1 \times S_2$ คือ เซตซึ่งมีสมาชิกเป็นคู่ลำดับ โดยคู่ลำดับแรกเป็นสมาชิกในเซต S_1 และคู่ลำดับที่สองเป็นสมาชิกในเซต S_2 สามารถอธิบายได้ด้วยตัวอย่างดังต่อไปนี้ ให้ $S_1 = \{a, b\}$ และ $S_2 = \{1, 2, 3\}$

$$S_1 \times S_2 = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$$

2. พังก์ชันและความสัมพันธ์ (Function and Relations)

- พังก์ชัน (Function) ทำหน้าที่ในการเชื่อมโยงระหว่างอินพุตไปยังเอาต์พุต โดยอินพุตหนึ่งค่าจะให้ค่าเอาต์พุตเพียงค่าเดียวเท่านั้น ยกตัวอย่าง เช่น

$$f(x) = y ; x \text{ คือ อินพุต และ } y \text{ คือ เอาต์พุต}$$

- ความสัมพันธ์ (Relation) ทำหน้าที่เชื่อมโยงความสัมพันธ์ระหว่างเซตของอินพุตซึ่งเรียกว่า โดเมน (Domain) ไปยังเซตของเอาต์พุตซึ่งเรียกว่า เรนจ์ (Range) เช่น

$$X = \{1, 2, 4\}$$

$$Y = \{4, 5, 6\}$$

ถ้าให้ R เป็นความสัมพันธ์น้อยกว่า ($<$)

$$R = X; R Y_i = \{(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (4, 5), (4, 6)\}$$

ความสัมพันธ์ที่เท่าเทียมกัน (Equivalence Relation) มีดังนี้

กำหนดให้ R เป็นความสัมพันธ์บนเซต S

- ความสัมพันธ์แบบสะท้อน (Reflexive) aRa สำหรับทุก ๆ a ที่อยู่ในเซต S นั่นคือ (a, a) เช่น $\{(1, 1), (2, 2), (3, 3)\}$

- ความสัมพันธ์แบบถ่ายทอด (Transitive) ถ้า aRb และ bRc แล้ว aRc นั่นคือ ถ้ามีความสัมพันธ์ (a, b) และ (b, c) แล้ว จะต้องมีความสัมพันธ์ (a, c) เช่น $\{(1, 2), (2, 3), (1, 3)\}$

- ความสัมพันธ์แบบสมมาตร (Symmetric) ถ้า aRb แล้ว bRa นั่นคือ ถ้ามีความสัมพันธ์ (a, b) แล้ว จะต้องมีความสัมพันธ์ (b, a) เช่น $\{(1, 2), (2, 1)\}$

สำหรับความสัมพันธ์ R ที่เท่าเทียมกัน (Equivalence Relation) ประเภทของความเท่าเทียมกัน (Equivalence Class) ถูกนิยามโดย $x = \{y : x R y\}$ เช่น

$$R = \{(1, 1), (2, 2), (1, 2), (2, 1), (3, 3), (4, 4), (3, 4), (4, 3)\}$$

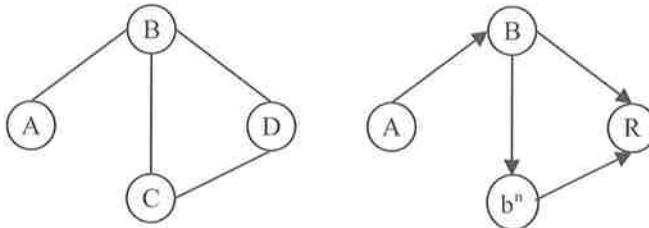
- ประเภทของความเท่าเทียมกันของ 1 = {1, 2}

- ประเภทของความเท่าเทียมกันของ 3 = {3, 4}

3. กราฟและต้นไม้ (Graph and Tree)

- กราฟ (Graph) คือ โครงสร้างข้อมูลที่ประกอบด้วยองค์ประกอบ 2 ส่วน คือ โนนด (Nodes หรือ Vertices) และกิ่ง (Edges) โดยโนนดต่าง ๆ จะถูกเชื่อมเข้าด้วยกันโดยกิ่งที่อยู่ในกราฟ เราแบ่งประเภทของกราฟ

ได้เป็น 2 ประเภทใหญ่ ๆ คือ กราฟที่ไม่มีทิศทาง (Undirected Graph) และกราฟที่มีทิศทาง (Directed Graph) ดังรูป



(ก) กราฟที่ไม่มีทิศทาง

(ข) กราฟที่มีทิศทาง

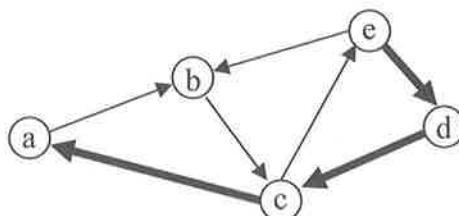
รูปที่ 1.1 กราฟ

ในการอธิบายกราฟโดยไม่ใช้แผนภาพ เราจะใช้สัญลักษณ์กราฟ $G = (V, E)$ โดยที่ V คือ เซตของโหนดที่อยู่ภายในกราฟ และ E คือ เซตของคู่ลำดับของโหนดเพื่อใช้อธิบายว่ามีโหนดไหนเชื่อมต่อถึงกันบ้าง เช่น จากรูปกราฟที่ไม่มีทิศทางสามารถอธิบายได้ดังนี้

$$G = (\{A, B, C, D\}, \{(A, B), (B, D), (B, C), (C, D)\})$$

หากเป็นกราฟที่มีทิศทางก็จะอธิบายคล้าย ๆ กัน ต่างกันตรงที่คู่ลำดับของโหนดที่เชื่อมกัน ซึ่งหากเราโหนดไหนขึ้นต้นก่อนก็จะหมายถึงจุดเริ่มต้นของลูกศร สำหรับคำศัพท์อื่น ๆ ที่อาจได้พบในเรื่องกราฟมีดังนี้

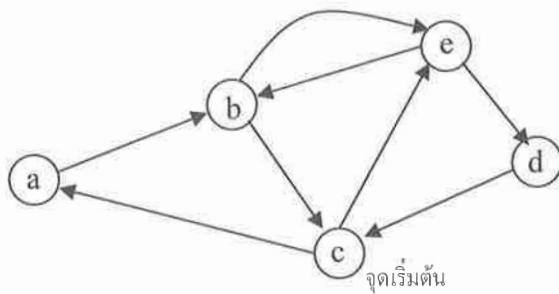
- ดีกรี (Degree) คือ จำนวนกิ่ง (Edges) ของโหนด ๆ หนึ่ง
- กราฟย่อย (Subgraph) A จะเป็นกราฟย่อยของ B ก็ต่อเมื่อเซตของโหนดของกราฟ A เป็นเซตย่อยของเซตของโหนดของกราฟ B นอกจากนี้เซตของกิ่งภายในกราฟ A ยังจะต้องเป็นเซตย่อยของเซตของกิ่งภายในกราฟ B อีกด้วย
- การเดิน (Walk) คือ ลำดับของกิ่งที่เชื่อมติดกัน เช่น



การเดินของกราฟรูปนี้คือ $(e, d), (d, c), (c, a)$

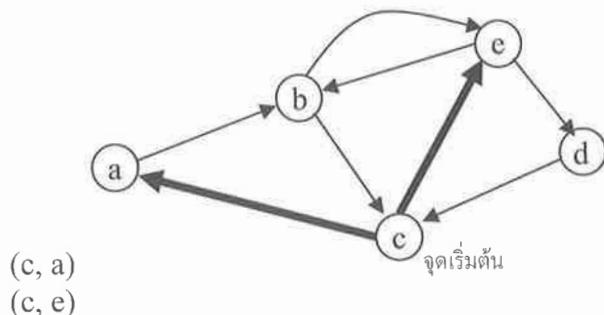
- ทาง (Path) คือ การเดินโดยไม่มีกิ่งใดซ้ำกันซ้ำ
- ทางแบบง่าย (Simple Path) คือ การเดินโดยไม่มีโหนดใดซ้ำกันซ้ำ

ตัวอย่างที่ 1.2 กำหนดให้โหนด c เป็นโหนดเริ่มต้น (Origin) จงหาทางแบ่งจ่ายทุก ๆ เส้นทางที่เป็นไปได้

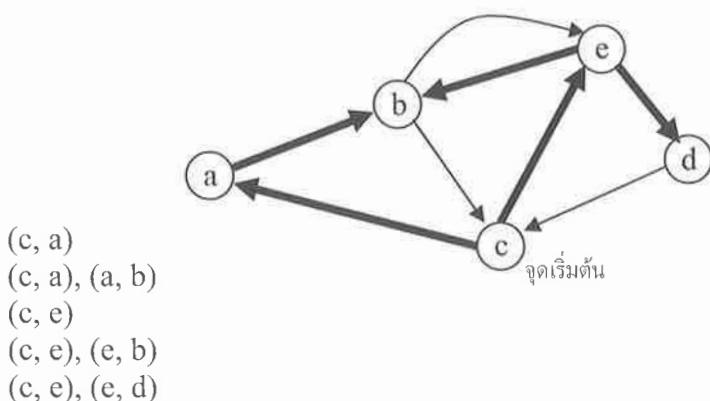


รูปที่ 1.2 กราฟที่มีโหนด c เป็นจุดเริ่มต้น

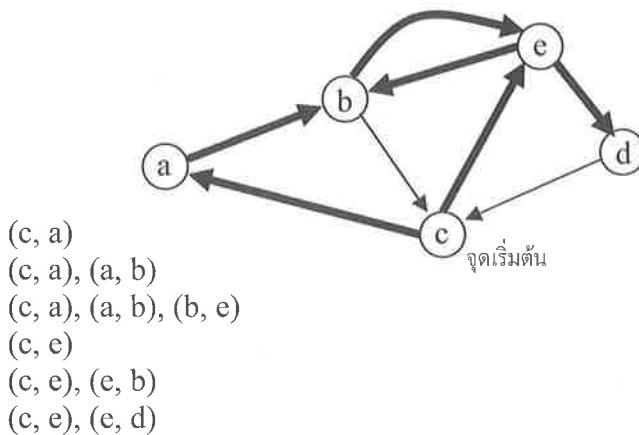
ขั้นตอนที่ 1 เริ่มต้นจากโหนด c เดินไปในทิศทางที่สามารถเดินได้ 1 ขั้นตอน



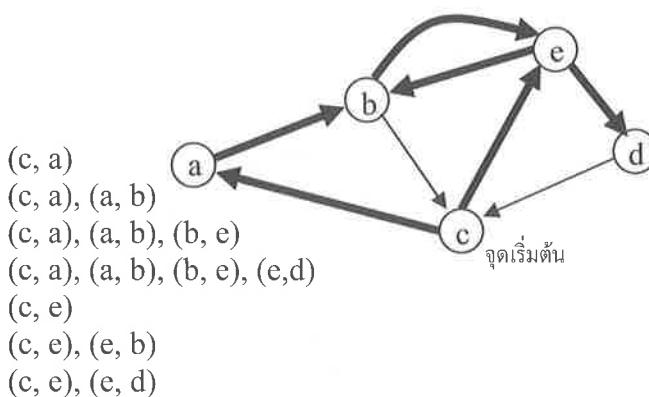
ขั้นตอนที่ 2 จากโหนด a และ e เดินไปในทิศทางที่สามารถเดินได้ 1 ขั้นตอน



ขั้นตอนที่ 3 จากโหนด b และ d ทำเช่นเดียวกันกับขั้นตอนที่ 2

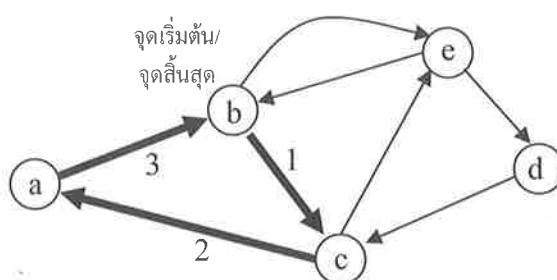


ขั้นตอนที่ 4



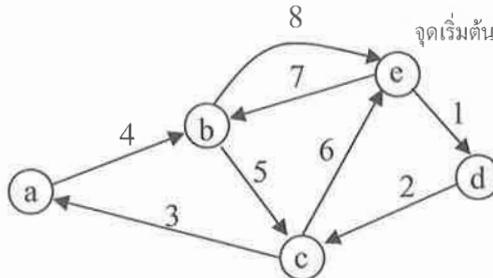
ดังนั้นทางแบบง่าย (Simple Path) ทั้งหมดที่เป็นไปได้ของกราฟดังกล่าวมีอยู่ทั้งสิ้น 7 เส้นทาง ดังแสดงในขั้นตอนที่ 4

- ใช้เคิล (Cycle) กราฟจะเกิดใช้เคิลซึ่ง ทางวิถีทาง (Path) ใด ๆ ที่มีจุดเริ่มต้นและจุดสิ้นสุดเป็นจุดเดียวกัน เช่น



รูปที่ 1.3 ใช้เคิลที่เกิดภายในกราฟ

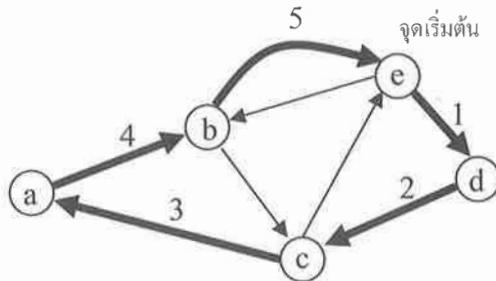
- ออยเลอร์ทัวร์ (Euler Tour) คือ ใช้เคิลที่ผ่านทุกจุดโดยแต่ละจุดจะผ่านเพียงครั้งเดียวเท่านั้น เช่น



รูปที่ 1.4 กราฟที่มีออยเลอร์ทัวร์

ตัวเลขในแต่ละจุดหมายถึงลำดับของการเดินผ่านกิ่งภายในออยเลอร์ทัวร์

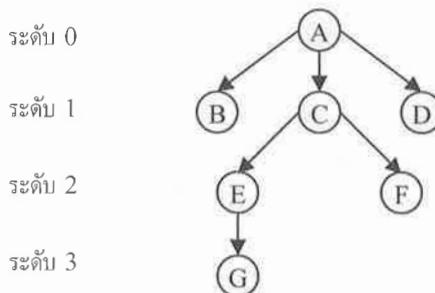
- ไฮมิลโทเนียนไซเคิล (Hamiltonian Cycle) คือ ใช้เคิลที่เกิดจากการเดินผ่านโหนดทุกโหนดของกราฟ โดยแต่ละโหนดจะถูกผ่านเพียงครั้งเดียวเท่านั้น



รูปที่ 1.5 กราฟที่มีไฮมิลโทเนียนไซเคิล

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

- ต้นไม้ (Tree) ถือเป็นกราฟชนิดหนึ่งที่มีทิศทางและไม่มีไซเคิล นอกจากนี้ ต้นไม้มักยังต้องมีโหนดที่ทำหน้าที่เป็นจุดเริ่มต้นบนสุด เรียกว่า ราก (Root) ดังรูป



รูปที่ 1.6 ต้นไม้ที่มีความลึกเท่ากับ 3

จากรูป โนหน A ทำหน้าที่เป็นroot ส่วนโนหน B, G, F และ D ทำหน้าที่เป็นโนหนใน ความลึกของต้นไม้ ต้นนี้คือ 3 หากมีกิ่งเชื่อมระหว่าง 2 โนหน โนหนที่อยู่สูงกว่าจะถูกเรียกว่า โนหนพ่อ-แม่ (Parent) ส่วนโนหนที่อยู่ต่ำกว่าจะถูกเรียกว่า โนหนลูก (Child)

1.2 ตัวอักษร สตริง และภาษา

ตัวอักษร (Alphabet) คือ หน่วยย่อยที่สุดของภาษา เชตของตัวอักษรนิยมแทนด้วยสัญลักษณ์ Σ เช่น

$$\Sigma = \{a, e, i, o, u\}$$

จากตัวอย่าง เชตของตัวอักษร (Σ) ประกอบด้วยตัวอักษร 5 ตัว คือ a, e, i, o, u โดยปกติแล้วเชตของตัวอักษร (Σ) จะต้องนับได้และเป็นเชตที่จำกัด

สตริง (String) เกิดจากการนำตัวอักษรหลาย ๆ ตัวมาเรียงต่อกัน เช่น aeiou คือ สตริงที่เกิดจากเชตของตัวอักษร Σ เราสามารถหาความยาวของสตริง w ได้โดยการนับจำนวนของตัวอักษรภายในสตริง w (สัญลักษณ์ที่ใช้แทนความยาวของสตริง w คือ $|w|$)

$$w = aeiou, |w| = 5$$

สตริงว่าง¹ (Empty String) คือ สตริงที่ไม่มีตัวอักษรใด ๆ ประกอบอยู่เลย ใช้สัญลักษณ์แทนด้วย ε โดยความหมายของสตริงว่างจะมีค่าเป็นศูนย์

$$|\varepsilon| = 0$$

$$\varepsilon w = w\varepsilon = w$$

สตริงย่อ (Substring) คือ ส่วนใด ๆ ของตัวอักษรที่อยู่ติดกันภายในสตริง เช่น

$$w = aeiou$$

เชตของสตริงย่อของ w คือ $\{\varepsilon, a, e, i, o, u, ae, ei, io, ou, \dots, aeio, eiou, aeiou\}$

สังเกตว่าสตริงว่าง (ε) ถือเป็นสตริงย่อของทุก ๆ สตริง

สตริงย่อส่วนหน้า (Prefix) และสตริงย่อส่วนหลัง (Suffix) หากสตริง w ประกอบด้วยสตริงย่อ u และ v

$$w = uv$$

เราเรียกว่า u ของสตริงย่อ u ว่า สตริงย่อส่วนหน้า และสตริงย่อ v ว่า สตริงย่อส่วนหลัง เช่น กำหนดให้สตริง $w = abba$

สตริงย่อส่วนหน้า (Prefix)

สตริงย่อส่วนหลัง (Suffix)

ε	abba
a	bbba
ab	bba
abb	ba
abbb	a
abba	ε

¹ คำราบงาเล่นอาจใช้สัญลักษณ์ของสตริงว่างแตกต่างกันไป เช่น λ หรือ ^ แต่ที่นิยมและเป็นมาตรฐานมักจะแทนด้วยสัญลักษณ์ ε (epsilon)

การเชื่อมต่อสตริง (Concatenation) คือ การนำสตริง 2 ตัว มาเชื่อมต่อเข้าด้วยกัน เช่น

$$x = aei, \quad y = ou, \quad xy = aeiou$$

หากนำสตริง 2 ตัวมาเชื่อมต่อกัน ความยาวของสตริงที่ได้จะมีค่าเท่ากับผลบวกของความยาวของสตริงทั้งสองตัวนี้ นั่นคือ

$$|xy| = |x| + |y|$$

นอกจากการเชื่อมต่อสตริงจะกระทำบนสตริง 2 ตัวได้แล้ว ยังสามารถกระทำกับภาษาสองภาษาได้อีกด้วย ตัวอย่างเช่น

$$\text{กำหนดให้ } L_1 = \{a, ab, ba\} \text{ และ } L_2 = \{b, aa\}$$

$$L_1 L_2 = \{ab, aaa, abb, abaa, bab, baaa\}$$

การทำซ้ำสตริง (Repetition) คือ การนำสตริงตัวเดิมมาต่อ กันเข้าหากัน ๆ ครั้ง เช่น ถ้า w เป็นสตริง w^n แผนการทำซ้ำของสตริง w ทั้งหมด n ครั้ง โดยที่ $w^0 = \epsilon$ เช่น

$$\text{กำหนดให้ } w = abc$$

$$w^0 = \epsilon$$

$$w^1 = abc$$

$$w^2 = abcabc$$

$$w^3 = abcabca$$

การทำซ้ำสามารถกระทำกับสตริงที่อยู่ภายใต้รูปแบบของภาษา L ได้ดังนี้

$$L = \{a, b\}$$

$$L^2 = LL = \{a, b\} \times \{a, b\} = \{aa, ab, ba, bb\}$$

$$L^3 = \{a, b\} \times \{a, b\} \times \{a, b\} = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$$

$$L^0 = \{\epsilon\}$$

การทำซ้ำแบบ * (* Operation) หาก Σ เป็นเซตของตัวอักษร เราใช้ Σ^* แทนเซตของสตริงที่เกิดจาก การนำสมาชิกภายใน Σ มาต่อ กัน ตั้งแต่ 0 ครั้งขึ้นไป เช่น

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

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

$$= \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, \dots\}$$

การทำซ้ำแบบ + (+ Operation) หาก Σ เป็นเซตของตัวอักษร เราใช้ Σ^+ แทนเซตของสตริงที่เกิดจาก การนำสมาชิกภายใน Σ มาต่อ กัน ตั้งแต่ 1 ครั้งขึ้นไป เช่น

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

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$$

$$= \{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, \dots\}$$

โปรดสังเกตว่าการทำซ้ำแบบ + จะไม่รวมสตริงว่างเหมือนกับการทำซ้ำแบบ * ดังนั้น

$$\Sigma^+ = \Sigma^* - \epsilon$$

การย้อนกลับ (Reverse) คือ การสร้างสตริงใหม่โดยการย้อนกลับตัวอักษรภายในสตริงเดิม หนังสือเล่มนี้ใช้สัญลักษณ์ w^R แทนการย้อนกลับของสตริง w เช่น

$$w = abcdef$$

$$w^R = fedcba$$

ภาษา (Language) คือ เซตของสตริงที่เกิดจากการประกอบกันของเซตตัวอักษร (Σ) โดยถูกต้องตามหลักของไวยากรณ์ภาษา เช่น ภาษาอังกฤษเกิดจากการนำตัวอักษร a ถึง z มาเรียงต่อ กันตามไวยากรณ์ภาษาอังกฤษ

$$\Sigma = \{a, b, c, \dots, x, y, z\}$$

$$\Sigma^* = \{\varepsilon, a, b, \dots, z, aa, ab, \dots, like, \dots, bee, \dots, fly, \dots, zbgxfd, \dots\}$$

$$\text{ภาษาของ } \Sigma = \{like, apple, bee, can, fly, man, loves, woman, \dots\}$$

โดยทั่วไป ภาษาของ Σ จะเป็นเซตย่อยของ Σ^* และเรามักเรียกสตริงที่อยู่ภายในภาษาว่าประโยค (Sentence)

1.3 เทคนิคการพิสูจน์

เทคนิคการพิสูจน์ที่เป็นที่นิยมใช้กันในการคณิตศาสตร์มี 3 วิธี คือ การพิสูจน์โดยการสร้าง (Proof By Construction), การพิสูจน์โดยอินดักชัน (Proof By Induction) และการพิสูจน์โดยการขัดแย้ง (Proof By Contradiction)

1. การพิสูจน์โดยการสร้าง (Proof By Construction) เป็นการแสดงวิธีทำ เพื่อเป็นการยืนยันถึงที่ต้องการพิสูจน์ เช่น หากเราต้องการพิสูจน์ว่า

“สำหรับจำนวนคู่ n ที่มีค่ามากกว่า 2 เราสามารถสร้างกราฟที่ทุก ๆ โหนดภายในกราฟมีจำนวนกิ่งเท่ากับ 3 ได้เสมอ”² เราสามารถพิสูจน์โดยการสร้างด้วยการเขียนอัลกอริทึมให้ดังนี้

พิสูจน์ : พังก์ชัน 3-Edge-Node-Graph (n)

/* n คือ จำนวนของโหนดที่อยู่ภายใต้กราฟ */

เริ่มต้น

ถ้า ($n > 2$) และ ($n \bmod 2 = 0$) ทำ

- สำหรับทุก ๆ โหนด i โดยที่ $i = 0$ ถึง $n - 2$

เชื่อมโหนด i กับโหนด $i + 1$ เข้าด้วยกัน

- เชื่อมโหนด $n - 1$ เข้ากับโหนด 0

- สำหรับทุก ๆ โหนด j โดยที่ $j = 0$ ถึง $n/2 - 1$

เชื่อมโหนด j กับโหนด $n/2 + j$ เข้าด้วยกัน

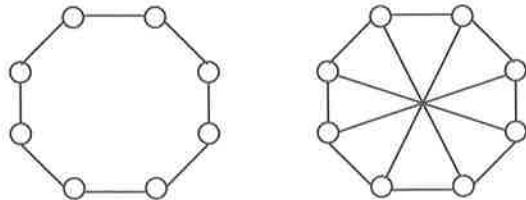
สิ้นสุด

² 來源จาก “Introduction to the Theory of Computation”, Michael Sipser, PWS Publishing Company

จากอัลกอริทึมดังกล่าว พจนะสรุปวิธีในการสร้างกราฟได้ดังนี้

(1) เชื่อมโหนดต่าง ๆ เข้าด้วยกันเป็นรูปวงกลม

(2) เชื่อมโหนดในครึ่งวงกลมส่วนบนเข้ากับโหนดในครึ่งวงกลมส่วนล่าง โดยจะต้องเป็นโหนดที่อยู่ตรงข้ามกันจึงจะเชื่อมกันได้



รูปที่ 1.7 ตัวอย่างของการพิสูจน์โดยการสร้าง

จากระบวนการและขั้นตอนดังกล่าว คือ วิธีการสร้างกราฟตามที่เราต้องการ ดังนั้นข้อความดังกล่าวได้รับการพิสูจน์แล้วโดยวิธีการพิสูจน์โดยการสร้าง

2. การพิสูจน์โดยอินดักชัน (Proof by Induction) ในกระบวนการพิสูจน์ทางคณิตศาสตร์ คอมพิวเตอร์ทั้งหมด การพิสูจน์แบบอินดักชัน (Induction Proof) เป็นกระบวนการพิสูจน์ที่นิยมใช้กันมากที่สุด หลักการมีอยู่ว่าหากต้องการพิสูจน์ว่าคุณสมบัติหนึ่งเป็นจริงเสมอสำหรับกฉุ่มของจำนวนนับ $n = \{1, 2, 3, \dots\}$ จะต้องพิสูจน์ 2 ส่วน ส่วนแรกต้องพิสูจน์ว่าคุณสมบัติดังกล่าวเป็นจริงสำหรับกรณีพื้นฐาน (Basis Case : $n = n_0$) ส่วนที่ 2 เป็นกรณีเฉพาะ (Inductive Case) ซึ่งต้องพิสูจน์ว่าหากตั้งสมมุติฐานว่าคุณสมบัติดังกล่าวเป็นจริงสำหรับจำนวนนับ 1 ถึง n คุณสมบัตินี้จะต้องเป็นจริงสำหรับจำนวน $n + 1$ ด้วย ดังนั้นการพิสูจน์โดยอินดักชันจึงพอสรุปเป็นขั้นตอนได้ดังนี้

ขั้นตอนที่ 1 : กรณีพื้นฐาน (Basis Case)

- พิสูจน์ว่ากรณีพื้นฐานที่สุด (ซึ่งอาจจะมีได้มากกว่าหนึ่งกรณี) ของสิ่งที่ต้องการพิสูจน์เป็นจริง เช่น พิสูจน์ว่า P_1 เป็นจริง โดยที่ P_1 คือกรณีพื้นฐานของสิ่งที่ต้องการพิสูจน์

ขั้นตอนที่ 2 : กรณีเฉพาะ (Inductive Case)

- ตั้งสมมุติฐานกับสิ่งที่ต้องการพิสูจน์ว่าเป็นจริงสำหรับขอบเขตหนึ่ง เช่น สมมุติว่า P_1, P_2, \dots, P_k เป็นจริง
- สำหรับกรณีที่ P_{k+1} ถ้า P_1, P_2, \dots, P_k เป็นจริงแล้ว P_{k+1} ยังคงเป็นจริง ก็สรุปได้ว่าทุก ๆ ค่าของ P_i มีค่าเป็นจริงเสมอ

ตัวอย่างที่ 1.3 กำหนดให้ $\sum n$ คือ ผลรวมของตัวเลขจำนวนนับตั้งแต่ 1 ถึง n

$$\sum n = \frac{n(n+1)}{2} ; n = \{1, 2, 3, \dots\}$$

พิสูจน์ : จากโจทย์ดังกล่าวเป็นการพิสูจน์ว่าคุณสมบัติของผลบวกของเลขจำนวนนับตั้งแต่ 1 ถึง n มีค่าเท่ากับ $\frac{n(n+1)}{2}$ ขึ้นตอนการพิสูจน์แบ่งออกเป็น 2 ขั้นตอน คือ

(1) กรณีพื้นฐาน : เป็นการพิสูจน์ว่ากรณีพื้นฐานของคุณสมบัติดังกล่าวเป็นจริง นั่นคือ

$$\sum 1 = \frac{1(1+1)}{2} = 1 \quad \text{ซึ่งเป็นจริง}$$

(2) กรณีเฉพาะ : เป็นการนำกรณีพิสูจน์ที่เราตั้งสมมุติฐานว่าเป็นจริงมาพิสูจน์กรณีถัดไป หากผลลัพธ์ออกมากเป็นจริง เรายังสรุปได้ว่าสมมุติฐานที่เราตั้งไว้นั้นถูกต้อง

$$\text{สมมุติฐาน : } \text{สมมุติว่า } \sum i = \frac{i(i+1)}{2} \text{ เมื่อ } 1 \leq i \leq n \text{ และ } i \text{ เป็นจำนวนนับ เป็นจริง}$$

ดังนั้นหากเราสามารถพิสูจน์ได้ว่า $\sum (n+1) = \frac{(n+1)(n+1+1)}{2}$ แสดงว่าคุณสมบัตินี้เป็นจริง
เราเริ่มพิสูจน์กรณี $i = n + 1$ ซึ่งจะได้ว่า

$$\sum (n+1) = 1 + 2 + 3 + \dots + n + (n+1)$$

จากสมมุติฐานที่เราตั้งไว้ $(1 + 2 + 3 + \dots + n)$ มีค่าเท่ากับ $\frac{n(n+1)}{2}$ ดังนั้น

$$\sum (n+1) = \frac{n(n+1)}{2} + (n+1)$$

$$\sum (n+1) = \frac{n(n+1) + 2(n+1)}{2} = \frac{(n+1)(n+1+1)}{2}$$

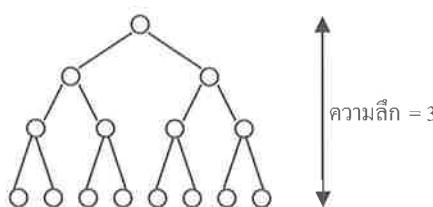
จะเห็นได้ว่าผลลัพธ์ที่ได้มีรูปแบบเหมือนกับสมมุติฐานที่เราตั้งไว้ ดังนั้นจึงสรุปได้ว่าสมมุติฐานดังกล่าวเป็นจริง และเป็นการพิสูจน์แบบอนันตักษณ์ว่า

$$\sum n = \frac{n(n+1)}{2} ; n = \{1, 2, 3, \dots\}$$

ตัวอย่างที่ 1.4 จงพิสูจน์ว่าจำนวนโหนดทั้งหมดในต้นไม้ไบナรีที่สมดุล (Balanced Binary Tree) ที่มีความลึกเท่ากับ h มีค่าเท่ากับ

$$\text{NumNode} = 2^{h+1} - 1$$

โดยต้นไม้ไบนารีที่สมดุล คือ ต้นไม้ที่โหนดภายในมีจำนวนของโหนดลูกเท่ากับ 2 โหนด (ยกเว้นโหนดใบ) และทุก ๆ เส้นทางจากโหนดรากไปถึงทุก ๆ โหนดในมีความลึกเท่ากัน เช่น



รูปที่ 1.8 ต้นไม้ไบนารีที่สมดุล

พิสูจน์ : (1) กรณีพื้นฐาน : ให้ความลึก (h) มีค่าเท่ากับ 0

$\text{NumNode} = 2^{0+1} - 1 = 1$ ซึ่งเป็นจริง (จำนวนโหนดของต้นไม้ที่มีความสูงเท่ากับ 0 มีเพียงโหนดเดียวเท่านั้น คือ โหนดราก)

(2) กรณีเชิงพาณิชย์ : สมมุติให้ $\text{NumNode}_h = 2^{h+1} - 1$ เป็นจริง สำหรับ $0 \leq h \leq n$
พิสูจน์กรณี $h = n + 1$ จะได้ว่า

$$\text{NumNode}_{n+1} = 1 + 2 + 2^2 + \dots + 2^n + 2^{n+1}$$

เนื่องจากสมมุติฐานของเรามี $1 + 2 + 2^2 + \dots + 2^n$ มีค่าเท่ากับ $2^{n+1} - 1$

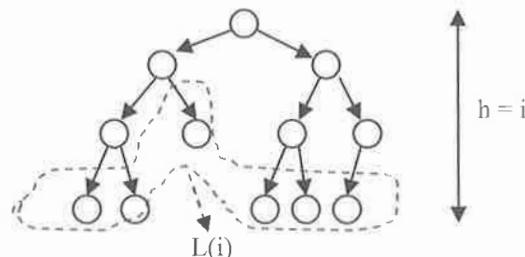
$$\begin{aligned}\therefore \text{NumNode}_{n+1} &= 2^{n+1} - 1 + 2^{n+1} \\ &= 2^{(n+1)+1} - 1\end{aligned}$$

จะเห็นได้ว่าผลที่ได้ตรงกับสมมุติฐานที่เราตั้งไว้ ดังนี้จึงเป็นการพิสูจน์ลิงที่เราต้องการ

ตัวอย่างที่ 1.5 จงพิสูจน์ว่าจำนวนโหนดที่เป็นใบ (Leaf Node) ในต้นไม้ใบnarีที่มีความลึกเท่ากับ H มีค่าสูงสุดไม่เกิน

$$\text{LeafNodes} \leq 2^h$$

พิสูจน์ : กำหนดให้ $L(i)$ เป็นจำนวนโหนดในที่มากที่สุดของต้นไม้ย่อย (Subtree) ที่มีความลึกเท่ากับ i



รูปที่ 1.9 $L(i)$ ของต้นไม้ใบnarี

(1) กรณีพื้นฐาน : ให้ความลึก (h) มีค่าเท่ากับ 0

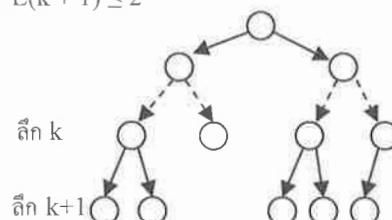
$$L(0) = 1 \quad (\text{โหนดรากเพียงโหนดเดียว})$$

(2) กรณีเชิงพาณิชย์ :

$$\text{สมมุติฐานของเรามี } L(i) \leq 2^i \text{ สำหรับ } i = 0, 1, \dots, k$$

จะต้องพิสูจน์ให้ได้ว่าสำหรับ $i = k + 1$

$$L(i) = L(k + 1) \leq 2^{k+1}$$



เนื่องจากความจริงที่ว่า $L(k+1)$ มีค่าไม่เกิน $2*L(k)$

$$L(k+1) \leq 2*L(k)$$

หมายความว่าสำหรับต้นไม้ในนารีได จำนวนโหนดที่ระดับความลึก $k+1$ จะต้องไม่เกิน 2 เท่าของจำนวนโหนดที่ระดับความลึก k

จากสมมุติฐานของเรา $L(i) \leq 2i$ สำหรับ $i = 0, 1, \dots, k$ ดังนี้

$$L(k+1) \leq 2*2^k$$

$$L(k+1) \leq 2^{k+1}$$

ดังนั้น $L(k+1)$ จึงมีค่าสูงสุดไม่เกิน 2^{k+1} ซึ่งตรงกับสิ่งที่ต้องการพิสูจน์

ตัวอย่างที่ 1.6 จงพิสูจน์ว่า $\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}$

พิสูจน์ : (1) กรณีพิเศษ : ให้ $n = 0$

$$\sum_{i=0}^n 0^2 = \frac{0(0+1)(2*0+1)}{6} = 0 \text{ เป็นจริง}$$

(2) กรณีเฉพาะ : สมมุติฐานของเราคือ

$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

ดังนั้นสำหรับ $i = n+1$,

$$\sum_{i=0}^{n+1} i^2 = 0 + 1^2 + 2^2 + 3^2 + \dots + n^2 + (n+1)^2$$

จากสมมุติฐาน เรายาราวว่า $0 + 1^2 + 2^2 + 3^2 + \dots + n^2$ มีค่าเท่ากับ $\frac{n(n+1)(2n+1)}{6}$

$$\sum_{i=0}^{n+1} i^2 = \frac{n(n+1)(2n+1)}{6} + (n+1)^2$$

$$= \frac{n(n+1)(2n+1) + 6(n+1)^2}{6}$$

$$= \frac{2n^3 + 9n^2 + 13n + 6}{6}$$

$$= \frac{(n^2 + 3n + 2)(2n + 3)}{6}$$

$$= \frac{(n+1)(n+2)(2n+3)}{6}$$

$$\therefore \sum_{i=0}^{n+1} i^2 = \frac{(n+1)(n+1+1)(2(n+1)+1)}{6}$$

ดังนั้นสมมุติฐานที่ตั้งไว้จึงเป็นจริง ทำให้สรุปได้ว่า

$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

ตัวอย่างที่ 1.7 จะพิสูจน์ว่า $\sum_{i=0}^n i^3 = (\sum_{i=0}^n i)^2$

พิสูจน์ : (1) กรณีพื้นฐาน ; $i = 0$

$$\sum_{i=0}^0 i^3 = 0 \text{ เป็นจริง}$$

(2) กรณีเชิงพาณิชย์ : สมมุติว่า $\sum_{i=0}^n i^3 = (\sum_{i=0}^n i)^2$ เป็นจริง

พิสูจน์กรณีที่ $i = n + 1$

$$\begin{aligned} \sum_{i=0}^{n+1} i^3 &= (0^3 + 1^3 + 2^3 + \dots + n^3) + (n+1)^3 \\ &= (\sum_{i=0}^n i)^2 + (n+1)^3 \end{aligned}$$

จากตัวอย่างก่อนหน้านี้ เรายาระว่า $\sum_{i=0}^n i = \frac{n(n+1)}{2}$

$$\begin{aligned} \sum_{i=0}^{n+1} i^3 &= \left(\frac{n(n+1)}{2} \right)^2 + (n+1)^3 \\ &= \frac{n^2(n+1)^2 + 4(n+1)^3}{4} \\ &= \frac{(n+1)^2(n^2 + 4(n+1))}{4} \\ &= \frac{(n+1)^2(n+2)^2}{4} \\ &= \left(\frac{(n+1)(n+1+1)}{2} \right)^2 \\ &= (\sum_{i=0}^{n+1} i)^2 \end{aligned}$$

$$\therefore \sum_{i=0}^n i^3 = (\sum_{i=0}^n i)^2$$

3. การพิสูจน์โดยการขัดแย้ง (Proof by Contradiction) เป็นการอาศัยการขัดแย้งกันระหว่าง สมมุติฐานที่ตั้งไว้กับผลที่ได้มาจากการพิสูจน์ การพิสูจน์แบบนี้แตกต่างจากการพิสูจน์โดยอินดักชันตรงที่การพิสูจน์โดยอินดักชันจะตั้งสมมุติฐานขึ้นมาแล้วอาศัยผลจากการพิสูจน์มาสนับสนุนสมมุติฐานที่ตั้งไว้

ตัวอย่างที่ 1.8 จำนวนจริง x ใด ๆ จะถูกเรียกว่าเป็นจำนวนที่สามารถอ่านลงได้โดยไม่ใช้เครื่องหมายกรณฑ์ (Rational Number) ถ้าจำนวนจริง x สามารถเขียนให้อยู่ในรูป m/n โดยที่ m และ n เป็นจำนวนเต็มและไม่มีตัวประกอบร่วมกัน จงแสดงว่า $\sqrt{2}$ ไม่ใช่จำนวนที่สามารถอ่านลงได้โดยไม่ใช้เครื่องหมายกรณฑ์

พิสูจน์ : สมมุติว่า $\sqrt{2}$ เป็นจำนวนที่สามารถอ่านลงได้โดยไม่ใช้เครื่องหมายกรณฑ์ และ

$$\sqrt{2} = \frac{m}{n} \text{ โดยที่ } m \text{ และ } n \text{ เป็นจำนวนเต็มและไม่มีตัวประกอบร่วมกัน}$$

ยกกำลังสองทั้งสองข้างของสมการ

$$2 = \frac{m^2}{n^2}$$

$$2n^2 = m^2 \quad \dots\dots(1)$$

จากสมการที่ (1) ทำให้เราทราบว่า m จะต้องเป็นเลขคู่ เนื่องจาก $2n^2$ เป็นเลขคู่ และเลขคู่เท่านั้นที่มีผลลัพธ์จากการยกกำลัง 2 เป็นเลขคู่ ดังนั้น m จะต้องเขียนให้อยู่ในรูป $2k$ เมื่อ k เป็นจำนวนเต็มได้

$$m = 2k$$

แทนค่า $m = 2k$ ลงในสมการที่ (1)

$$2n^2 = (2k)^2$$

$$n^2 = 2k^2 \quad \dots\dots(2)$$

จากสมการที่ (2) ทำให้ทราบว่า n จะต้องเป็นเลขคู่เช่นเดียวกับ m ด้วยเหตุผลเดียวกัน ซึ่งแน่นอนว่าทั้ง m และ n จะต้องมีตัวประกอบร่วมกันอย่างน้อยหนึ่งตัว (นั่นคือ 2) ทำให้เราทราบว่าสมมุติฐานที่เราตั้งไว้นั้นผิด เนื่องจากขัดแย้งกับผลลัพธ์ที่ได้จากการพิสูจน์ ดังนั้น $\sqrt{2}$ จึงไม่ใช่จำนวนที่สามารถอ่านลงได้โดยไม่ใช้เครื่องหมายกรณฑ์

1.4 ไวยากรณ์และอโตมาตา

1. **ไวยากรณ์ (Grammar)** คือ กฎเกณฑ์ที่ถูกกำหนดขึ้นเพื่อใช้ในการสร้างภาษาขึ้นมา เช่นเดียวกับไวยากรณ์ในภาษามนุษย์ที่ถูกสร้างขึ้นมาเพื่อกำหนดรูปแบบในการสื่อสารในชีวิตประจำวัน เช่น

<ประโยค> —> <ประธาน> <กริยา> <กรรม>

สำหรับไวยากรณ์ของภาษาคอมพิวเตอร์ สามารถอธิบายได้ดังนี้

กำหนดให้ G เป็นไวยากรณ์ซึ่งเขียนแทนด้วยคู่ลำดับทั้งสี่

$$G = (V, T, S, P)$$

โดยที่ V คือ เซตของตัวแปร (Variable) เช่น <ประธาน>, <กริยา>, <กรรม> เป็นต้น

T คือ เซตของสัญลักษณ์เทอร์มินอล (Terminal Symbols) ซึ่งเป็นสัญลักษณ์ที่กระจายเป็นสัญลักษณ์ตัวอิ่นไม่ได้อีกแล้ว เช่น John, Peter, eat, apple เป็นต้น

S คือ สัญลักษณ์เริ่มต้นของไวยากรณ์ (Start Symbol) เช่น <ประโยชน์>

P คือ เซตของกฎเกณฑ์ในการสร้างภาษา (Productions) เช่น

<ประโยชน์> —→ <ประธาน> <กริยา> <กรรม>

<ประธาน> —→ John | Peter

<กริยา> —→ Eats

<กรรม> —→ Apple

ความสัมพันธ์ระหว่าง V, T, S คือ

- $S \in V$
- $V \cap T = \emptyset$
- $V \neq \emptyset$
- $T \neq \emptyset$

หากสตริง w เป็นสตริงที่ถูกสร้างจากไวยากรณ์ G จะต้องมีเซตของโปรดักชัน p ที่เริ่มต้นจากสัญลักษณ์เริ่มต้น S และกระจายออกมานาไปได้เป็นสตริง w เช่น จากเซตของโปรดักชัน p ข้างต้น เราสามารถนำมารังสตริง “John Eats Apple” ได้ดังนี้

<ประโยชน์> —→ <ประธาน> <กริยา> <กรรม>

—————> John <กริยา> <กรรม>

—————> John Eats <กรรม>

—————> John Eats Apple

หากเราแทนเซตของประโยชน์ (Sentence) ที่ถูกสร้างจากไวยากรณ์ G ด้วย $L(G)$ เราจะได้ว่า $w \in L(G)$ โดยที่ w จะต้องประกอบด้วยสัญลักษณ์เทอร์มินอล (Terminal Symbol) หรือสตริงว่างเท่านั้น นอกเหนือนี้ เราจึงเรียกกระบวนการในการแปลงสัญลักษณ์เริ่มต้น S ไปเป็นสตริง w ว่า การได้มา (Derivation) เช่น

$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n \Rightarrow w$ โดยที่

- w_1, w_2, \dots, w_n ถูกเรียกว่ารูปแบบเชื่อมเทืนเชียง (Sentential Forms)

- w ถูกเรียกว่า สตริงสิ้นสุด (Terminal String)

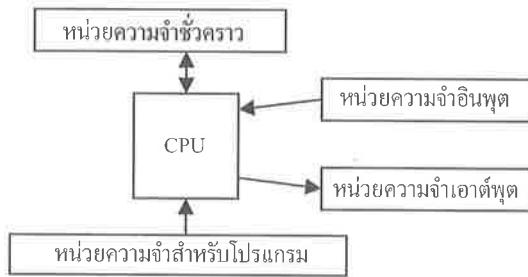
- เราสามารถเขียน $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n \Rightarrow w$ แบบย่อ ๆ ได้ว่า $S \xrightarrow{*} w$

รายละเอียดของไวยากรณ์ยังมีอีกมาก เราจะได้ศึกษาลึกลึกลื้อหาไวยากรณ์ของภาษาต่าง ๆ ในบทต่อ ๆ ไป

2. ออโตมาตา (Automata)³ องค์ประกอบพื้นฐานในการคำนวณทางคอมพิวเตอร์มีอยู่ 2 ส่วน คือ ส่วนประมวลผลและหน่วยความจำ ส่วนประมวลผลทำหน้าที่คำนวณกระบวนการพื้นฐานทางคณิตศาสตร์ เช่น บวก, ลบ, แอนด์ (AND), ออร์ (OR), เอ็กซ์คลูซีฟ-ออร์ (Exclusive-OR) เป็นต้น ส่วนหน่วยความจำถูกแบ่งออกเป็น 4 ส่วน ส่วนที่ 1 ทำหน้าที่เก็บซอฟต์แวร์ (Program Memory) ที่ผู้ใช้งานป้อนเข้าไป ส่วนที่ 2 เป็นหน่วยความจำชั่วคราว (Temporary Memory) ซึ่งทำหน้าที่เก็บข้อมูลชั่วคราวที่เกิดขึ้นในระหว่างการคำนวณ และ

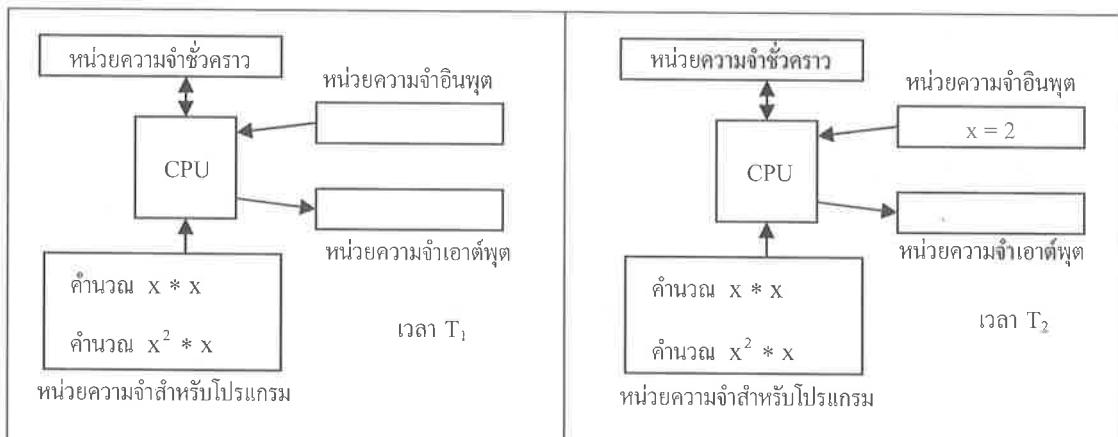
³ Automata เป็นคำภาษาจีน หากเป็นเอกสารเจ้าจะเรียกว่า Automaton

2 ส่วนสุดท้ายคือ หน่วยความจำสำหรับอินพุตและเอาต์พุตตามลำดับ หน้าที่ของหน่วยความจำ 2 ส่วนสุดท้ายนี้คือ การเก็บผลลัพธ์ที่ถูกนำมาเข้า-ออกระหว่างหน่วยประมวลผลและส่งเวดล้อมภายนอกระบบ

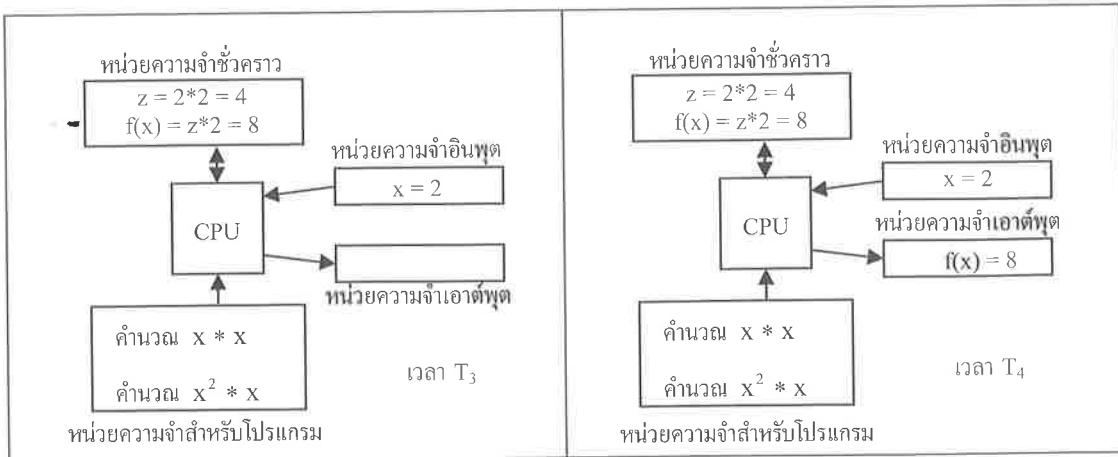


รูปที่ 1.10 องค์ประกอบพื้นฐานในการคำนวนทางคอมพิวเตอร์

ตัวอย่างการทำงานภายในคอมพิวเตอร์สามารถแสดงโดยใช้ตัวอย่างง่าย ๆ ดังต่อไปนี้ สมมุติว่าเราต้องการ ตั้งให้คอมพิวเตอร์คำนวนผลลัพธ์ของโปรแกรม $f(x) = x^3$ ค่าอินพุต 2 ที่รับเข้ามาจากการอ่านค่าจากหน้าจอ ให้หน่วยความจำอินพุต จากนั้นหน่วยประมวลผล (CPU) จะนำโปรแกรมซึ่งถูกเก็บไว้ที่หน่วยความจำสำหรับโปรแกรม เข้าสู่หน่วยความจำชั่วคราวพร้อมทั้งแทนค่า x ด้วย 2 และคำนวนผลลัพธ์ซึ่งมีค่าเท่ากับ 8 หลังจากนั้นผลลัพธ์ ดังกล่าวจะถูกนำไปพักไว้ที่หน่วยความจำเอาต์พุตเพื่อให้หน่วยแสดงผลนำค่าที่ได้ไปใช้งานต่อไป



รูปที่ 1.11 การจำลองการคำนวน $f(x) = x^3$
(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

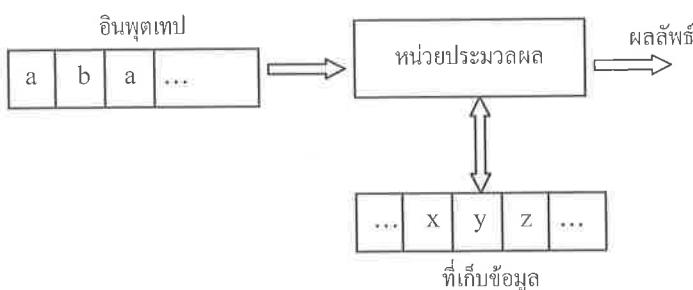


รูปที่ 1.11 (ต่อ)

ออโตมาตา เป็นคำที่ใช้เรียกแทนแบบจำลองการทำงานของคอมพิวเตอร์ ซึ่งมีส่วนประกอบอยู่ 3 ส่วน ดังนี้

- (1) อินพุตเทป (Input Tape)
- (2) หน่วยประมวลผล (Processor)
- (3) ที่เก็บข้อมูล (Storage)

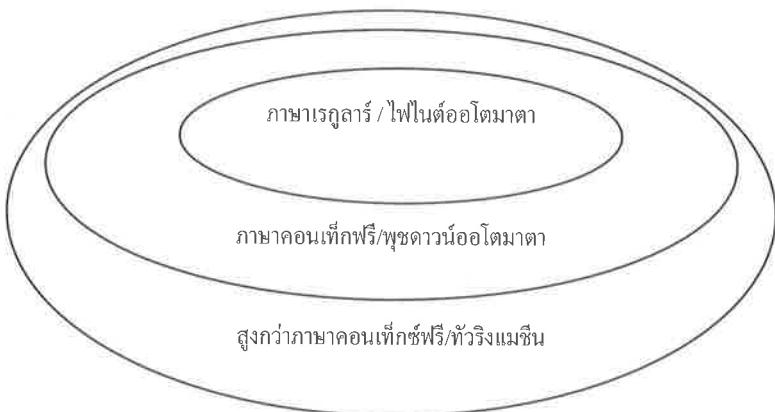
อินพุตเทปทำหน้าที่บรรจุตัวอักษรที่จะส่งให้หน่วยประมวลผล เช่น ซอฟต์แวร์โปรแกรม (Source Program) และข้อมูล (Data) หน่วยประมวลผลจะควบคุมสถานะภายในให้เปลี่ยนแปลงไปตามอินพุตที่ได้รับเข้ามา โดยมีที่เก็บข้อมูล (Storage) ทำหน้าที่เก็บผลที่ได้ในระหว่างการคำนวณ ส่วนผลลัพธ์ที่ได้จากหน่วยประมวลผลสามารถเป็นข้อมูลอะไรก็ได้ ดังรูป



รูปที่ 1.12 ออโตมาตา (Automata)

ออโตมาตามักถูกนำมาใช้เป็นเครื่องมือตรวจสอบความถูกต้องของโปรแกรม โดยอินพุตคือ ซอฟต์แวร์โปรแกรม ส่วนເອົາຕົ້ມຄວບຄຸງ ใช่ (yes) หรือ ไม่ใช่ (no) เพื่อเป็นการบอกว่าโปรแกรมที่รับเข้ามานั้นเขียนถูกต้องตามไวยากรณ์ที่ตั้งไว้หรือไม่ นอกจากนี้เรายังสามารถนำออโตมาตาไปประยุกต์ใช้ในการคำนวณทางคณิตศาสตร์ เช่น การหาผลคูณ

ของจำนวนส่องจำนวน และการหาจำนวนที่มีค่ามากที่สุด เป็นต้น ออโตมาตามีอยู่ 3 ประเภท แบ่งตามระดับของ ความสามารถในการรับภาษา คือ ไฟโนต์ออโตมาตา (Finite Automata) พุชดาวน์ออโตมาตา (Pushdown Automata) และทัวริงแมชีน (Turing Machine) ดังรูป



รูปที่ 1.13 ประเภทของภาษาและออโตมาตา

ภาษาพื้นฐานที่สุดในทางคณิตศาสตร์คือ ภาษาเรกูลาร์ซึ่งเป็นภาษาที่ไม่จำเป็นต้องใช้หน่วยความจำมากนักในการทำงาน ดังนี้ไฟโนต์ออโตมาตาจึงมีความสามารถที่สุดที่จะรับภาษานี้ ภาษามีความซับซ้อนขึ้นอีกหนึ่งระดับคือภาษาคอนเท็กซ์ฟรี เนื่องจากอนุญาตให้จำนวนของตัวอักษรภายในสตริงสามารถมีความสัมพันธ์กันได้ เช่น จำนวนตัวอักษร a และ b ภายในสตริงมีจำนวนเท่ากัน โดยจะมีจำนวนเท่ากันได้ ด้วยเหตุดังกล่าวออโตมาตาที่จะสามารถรับภาษาคอนเท็กซ์ฟรีได้จึงต้องมีหน่วยความจำที่ไม่จำกัด นั่นคือพุชดาวน์ออโตมาตา สำหรับภาษาที่มีความซับซ้อนมากกว่าภาษาคอนเท็กซ์ฟรี จะใช้ทัวริงแมชีนในการรับ ดังนั้นจึงสรุปได้ว่าทัวริงแมชีนมีความสามารถในการคำนวณสูงที่สุด และไฟโนต์ออโตมาตามีความสามารถในการคำนวณน้อยที่สุด รายละเอียดของภาษาและออโตมาตาแต่ละชนิด ผู้อ่านจะได้ศึกษาในบทต่อ ๆ ไปของหนังสือเล่มนี้ ซึ่งถือว่าเป็นเนื้อหาหลักของวิชาทฤษฎี การคำนวณในการศึกษาระดับปริญญาตรี

แบบฝึกหัด

1. จงยกตัวอย่างสตริงที่เป็นสมาชิกภายในเซตดังต่อไปนี้เซตละ 5 สตริง
 - (1) $\{x \mid x = 2n \text{ โดยที่ } n = 0, 1, 2, 3, \dots\}$
 - (2) $\{x \in \{0, 1\}^*\text{ โดยที่ค่าฐานลับของ } x \text{ หารด้วย } 2 \text{ ลงตัว}\}$
 - (3) $\{x \in \{a, b\}^*\text{ โดยที่จำนวนของตัวอักษร } a \text{ น้อยกว่าจำนวนตัวอักษร } b\}$
 - (4) $\{vxy \mid v \in \{0, 1\}^*\}$
 2. กำหนดให้ $A = \{a, b, c\}$
 $B = \{d, e, f\}$
 จงหา (1) เพาเวอร์เซตของ A
 - (2) $A - B$
 - (3) $A \times B$
 3. จงพิสูจน์ว่าสำหรับเซต A, B และ C ใด ๆ ถ้า $A \cap B = \emptyset$ และ $C \subseteq B$ แล้ว $A \cap C = \emptyset$
 4. จงอธิบายว่าเซต $\{(1, 3), (3, 1), (2, 2)\}$ มีคุณสมบัติใดบ้าง
 5. จงอธิบายเซตของสตริงดังนี้ $\{1, 11, 101, 111, 1011, 1101, 10001, 10011, \dots\}$
 6. ออโตมาตาคืออะไร มีประโยชน์อย่างไร และมีส่วนประกอบอะไรบ้าง
 7. จงพิสูจน์ว่าถ้า a และ b เป็นจำนวนคู่ แล้ว ab เป็นจำนวนคู่
 8. จงพิสูจน์ว่าสำหรับทุก ๆ $m \geq 1$,
- $$\sum_{i=1}^m i \cdot 2^i = (m-1) \cdot 2^{m+1} + 2$$
9. จงพิสูจน์ว่าสำหรับทุก ๆ $n \geq 1$ จำนวนเซตย่อย (Subset) ของ $\{1, 2, \dots, n\}$ มีจำนวนเท่ากับ 2^n
 10. จงพิสูจน์ว่าสำหรับ $n \geq 4, n! > 2^n$

ไฟโนต์อโตมาตา

ดังที่ได้กล่าวมาแล้วก่อนหน้านี้ว่าอโตมาตาเป็นแบบจำลองที่ถูกสร้างขึ้นมาเพื่อแสดงให้เห็นถึงการทำงานของคอมพิวเตอร์ยุคดิจิทัล ออโตมาตามีอยู่ด้วยกันหลายชนิด โดยชนิดที่เรียบง่ายที่สุดคือ ไฟโนต์อโตมาตา ในบทนี้ เราจะศึกษาถึงการทำงานของไฟโนต์อโตมาตา พร้อมทั้งความสัมพันธ์ระหว่างไฟโนต์อโตมาตาและนิพจน์เรกูลาร์ เพื่อเป็นพื้นฐานสำคัญในการเรียนรู้ในบทต่อไป

2.1 ไฟโนต์อโตมาตา

ก่อนจะศึกษาไฟโนต์อโตมาตา (Finite Automata) เราจะต้องทำความเข้าใจคำว่า “ไฟโนต์ (Finite)” ซึ่งในที่นี้หมายถึง ความจำกัดในแง่ของหน่วยความจำของคอมพิวเตอร์ เราสามารถใช้งานคอมพิวเตอร์ที่มีหน่วยความจำที่จำกัดทำงานอะไรได้บ้าง แน่นอนว่าคำตอนเป็นไปได้มากมาย หน่วยความจำของคอมพิวเตอร์มีความสัมพันธ์โดยตรงกับจำนวนของสถานะ (State) ภายในอโตมาตา โดยหน่วยความจำที่หน้าที่เก็บสถานะต่าง ๆ ของอโตมาตา หากหน่วยความจำมีจำกัด ความสามารถของอโตมา塔ก็จะถูกจำกัดด้วยเช่นกัน สำหรับผู้อ่านบาง คนที่เคยศึกษาเกี่ยวกับการออกแบบวงจรดิจิทัลมาแล้ว ก็จะรู้จักคำว่า “Finite State Machine” ซึ่งที่จริงแล้วก็เป็นคำที่มีความหมายเช่นเดียวกันกับไฟโนต์อโตมาตานั้นเอง

นิยาม 2.1.1 :

ไฟโนต์อโตมาตากูกแห่งด้วยองค์ประกอบห้า 5 คือ $(Q, \Sigma, \delta, q_0, F)$ โดยที่

Q คือ เซตของสถานะ (State) ของอโตมาตา ซึ่งเป็นเซตที่จำกัด

Σ คือ เซตของตัวอักษร (Alphabet) ซึ่งเป็นเซตที่จำกัด

δ คือ เซตของทรานซิชันฟังก์ชัน (Transition Function) ซึ่งมีความสัมพันธ์ คือ

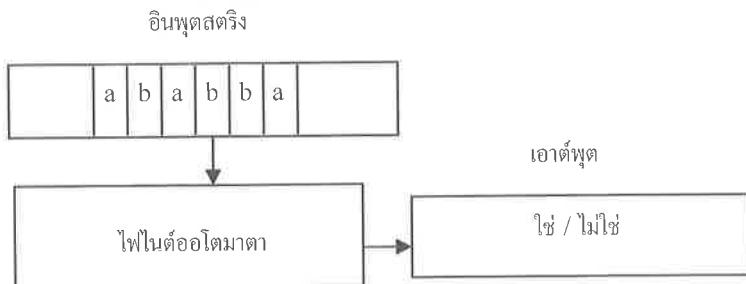
$$Q \times \Sigma \rightarrow Q$$

เช่น หากสถานะของไฟโนต์อโตมาตากือ q_1 และรับอินพุตเข้ามาเป็น a สถานะของไฟโนต์อโตมาตาจะถูกเปลี่ยนไปเป็น q_2 โดยที่ $a \in \Sigma, q_1 \in Q, q_2 \in Q$

q_0 คือ สถานะเริ่มต้นของไฟโนต์อโตมาตา ซึ่ง $q_0 \in Q$

F คือ เซตของสถานะสุดท้าย (Final State) โดยที่ $F \subseteq Q$ และ F เป็นสถานะที่ถือว่าไฟโนต์อโตมาตายอมรับอินพุตที่ป้อนเข้ามา

เพื่อให้เกิดความเข้าใจเกี่ยวกับลักษณะการทำงานของไฟโนต์อโตมาตา สามารถอธิบายอโตมาตาดังกล่าวด้วยรูปดังต่อไปนี้



รูปที่ 2.1 การทำงานของไฟโนต์อโตมาตา

ไฟโนต์อโตมาตาจะอ่านอินพุตเข้ามาทีละตัวอักษร (จากซ้ายไปขวา) เพื่อตรวจสอบว่าอโตมาตาอยู่รับอินพุตสตริงที่รับเข้ามาหรือไม่ ถ้ายอมรับอินพุตสตริงดังกล่าว ไฟโนต์อโตมาตาจะเปลี่ยนสถานะเข้าสู่สถานะสุดท้าย (Final State) เพื่อแทนการตอบ “ใช่” แต่ถ้าไม่ยอมรับอินพุตสตริงดังกล่าว สถานะของอโตมาตาจะเป็นสถานะอื่นที่ไม่ใช่สถานะสุดท้าย (Nonfinal State) เพื่อแทนการตอบ “ไม่ใช่”

ตัวอย่างที่ 2.1 ไฟโนต์อโตมาตา M_1 ประกอบด้วย

$$M_1 = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

โดยที่ δ คือ

$$\begin{aligned}\delta(q_0, 0) &= q_0, \\ \delta(q_0, 1) &= q_1, \\ \delta(q_1, 0) &= q_1, \\ \delta(q_1, 1) &= q_1\end{aligned}$$

ไฟโนต์อโตมาตา M_1 ประกอบด้วย 2 สถานะ (States) คือ q_0 และ q_1 ไฟโนต์อโตมาตาที่รับอินพุต 0 และ 1 เท่านั้น นอกจานี้ M_1 มีสถานะเริ่มต้นที่ q_0 และมี q_1 เป็นสถานะสุดท้าย การทำงานจะเริ่มต้นโดยรับอินพุตสตริงที่มี 1 เป็นส่วนประกอบอย่างน้อยหนึ่งตัว เช่น 1, 01, 001, 010 เป็นต้น

ผู้อ่านพยายามคิดเห็นเดียวกันว่าการอธิบายไฟโนต์อโตมาตาด้วยทรานซิชันฟังก์ชันนั้นเข้าใจค่อนข้างยาก เนื่องจากอาจจะเกิดความยากลำบากในการตรวจสอบการทำงานของทรานซิชันฟังก์ชันที่ซับซ้อนกว่านี้ ดังนั้นจึงมีการนำทรานซิชันกราฟมาช่วยอธิบายการทำงานของไฟโนต์อโตมาตา

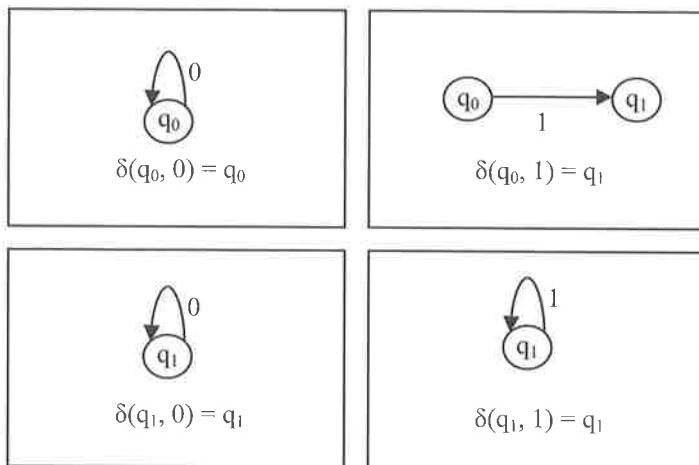
ขั้นตอนในการสร้างทรานซิชันกราฟมีดังนี้

1. สำหรับทรานซิชันฟังก์ชัน $\delta(q_a, x) = q_b$ ให้สร้างโหนด q_a และโหนด q_b จากนั้นเชื่อมโหนด q_a ไปยัง q_b โดยกำกับเส้นเชื่อมด้วยตัวอักษร x
2. นำกราฟที่ได้สำหรับแต่ละทรานซิชันฟังก์ชันมาเชื่อมต่อกัน โดยโหนดที่มีชื่อซ้ำกันให้ใช้โหนดร่วมกัน
3. สำหรับโหนดที่เป็นสถานะเริ่มต้น ให้เขียนลูกศรชี้เข้า เพื่อให้ทราบจุดเริ่มต้นการทำงานของไฟโนต์อโตมาตา

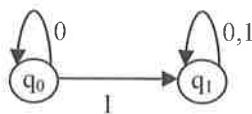
4. ทำสัญลักษณ์โนหนดที่เป็นสถานะสุดท้ายให้แตกต่างไปจากโนหนดอื่น โดยทั่วไปนิยมแทนด้วยสัญลักษณ์วงกลมซ่อนกันสองชั้น

สำหรับอโตมาตา M_1 ในตัวอย่างที่แล้ว เราสามารถสร้างทรานซิชันกราฟได้ดังนี้

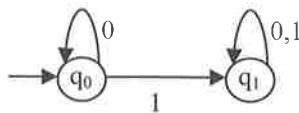
1. สร้างทรานซิชันกราฟย่อย สำหรับแต่ละทรานซิชันพังก์ชัน



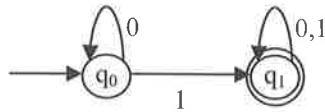
2. เชื่อมทรานซิชันกราฟย่อยเข้าด้วยกัน



3. ระบุสถานะเริ่มต้นโดยใช้ลูกศรซึ่งเข้าไปปั้งโนหนด q_0



4. ระบุสถานะสุดท้ายโดยสร้างวงกลมสองชั้นล้อมรอบโนหนด q_1



จากรูปทรานซิชันกราฟดังกล่าว ทำให้เราเข้าใจการทำงานของอโตมาตา M_1 ได้่ายขึ้นโดยไม่ต้องยุ่งยาก กับสัญลักษณ์ต่าง ๆ ในทรานซิชันพังก์ชัน ดังนั้นการใช้ทรานซิชันกราฟจึงเป็นทางเลือกที่ดีอีกทางหนึ่งในการอธิบายไฟในต่ออโตมาตา

การแทนทรานซิชันฟังก์ชันด้วยทรานซิชันกราฟ เป็นการอำนวยความสะดวกในการเข้าใจการทำงานของอโตมาตาได้อ่าย冗復เร็ว นอกจักนี้ยังมีวิธีการอำนวยความสะดวกในการเขียนทรานซิชันฟังก์ชันให้สั้นและเป็นระเบียบ นั่นคือการแทนทรานซิชันฟังก์ชันด้วยตารางทรานซิชัน (Transition Table) เช่น สมมุติว่าเรามีทรานซิชันฟังก์ชันสำหรับอโตมาตาดังต่อไปนี้

$$\delta(q_0, 0) = q_0,$$

$$\delta(q_0, 1) = q_1,$$

$$\delta(q_1, 0) = q_1,$$

$$\delta(q_1, 1) = q_1$$

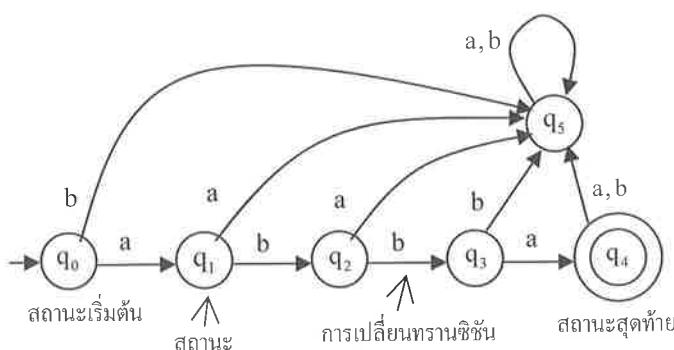
เซตของทรานซิชันฟังก์ชันดังกล่าวสามารถแทนด้วยตารางทรานซิชันได้ดังนี้

δ	0	1
q_0	q_0	q_1
q_1	q_1	q_1

แม้ว่าตารางทรานซิชันจะสามารถอธิบายทรานซิชันฟังก์ชันได้สั้นและสะดวกกว่า แต่หากพิจารณาในแง่ความง่ายในการเข้าใจแล้ว การอธิบายด้วยทรานซิชันกราฟจะสามารถอธิบายการทำงานของทรานซิชันฟังก์ชันได้ดีกว่า

สิ่งหนึ่งที่สังเกตได้จากตัวอย่างนี้คือ ทรานซิชันฟังก์ชันจะให้ค่าสถานะออกมายืนยันสถานะเดียวเท่านั้น เช่น $\delta(q_0, 0) = q_0$ ซึ่งเราเรียกไฟโน่ต์อโตมาตาที่มีลักษณะดังกล่าวว่า ไฟโน่ต์อโตมาตาที่คาดเดาได้ (Deterministic Finite Automata : DFA)

ตัวอย่างที่ 2.2 จงแสดงการทำงานของไฟโน่ต์อโตมาตาที่คาดเดาได้ดังต่อไปนี้ โดยกำหนดให้อินพุตสตริงคือ abba

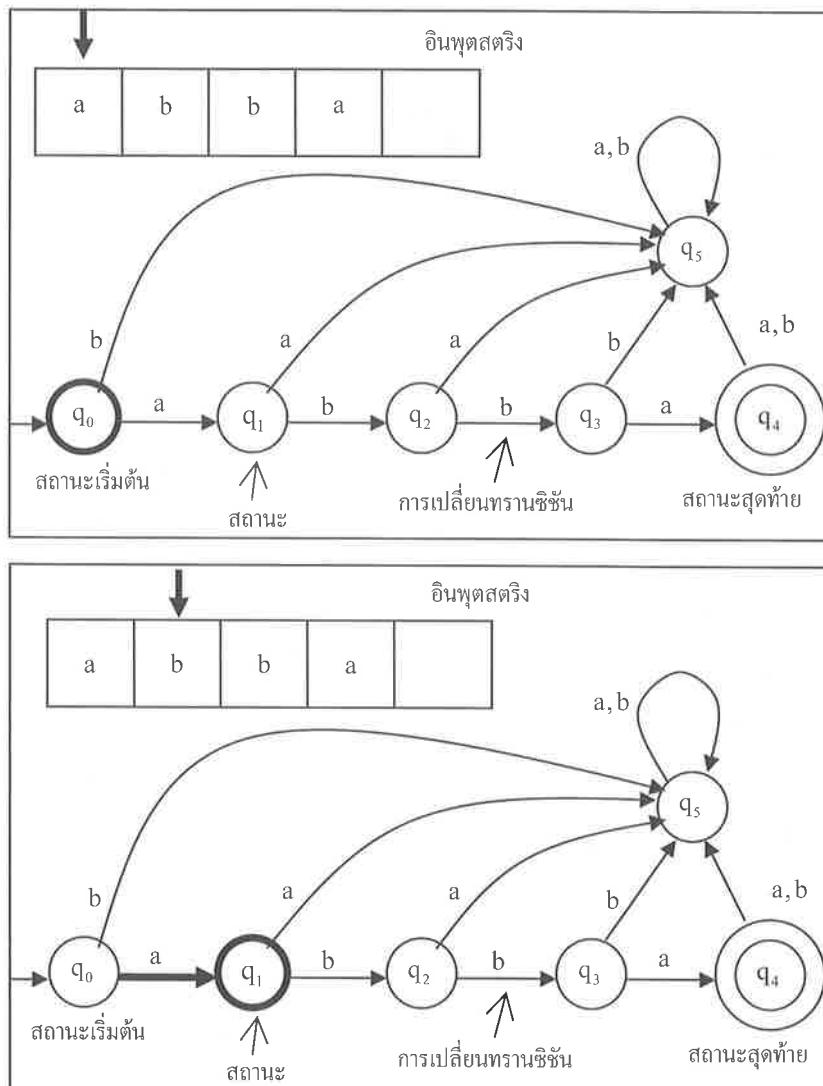


รูปที่ 2.2 ไฟโน่ต์อโตมาตา

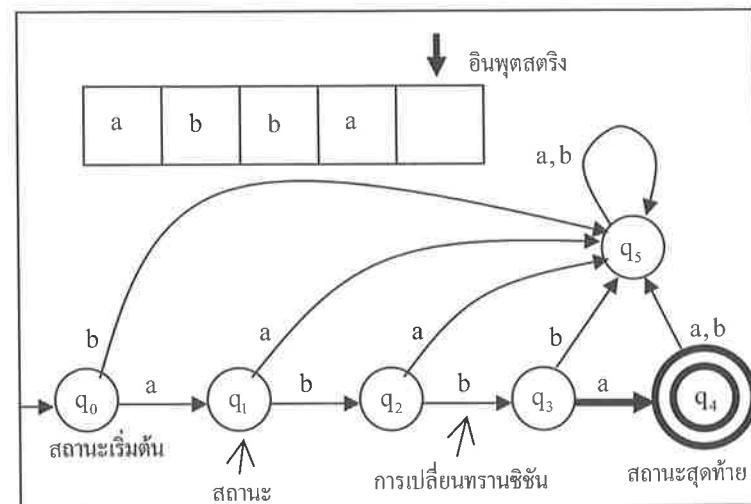
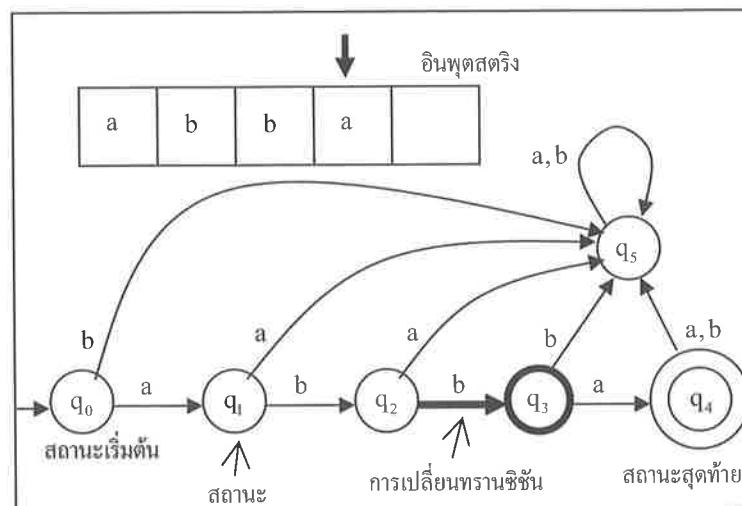
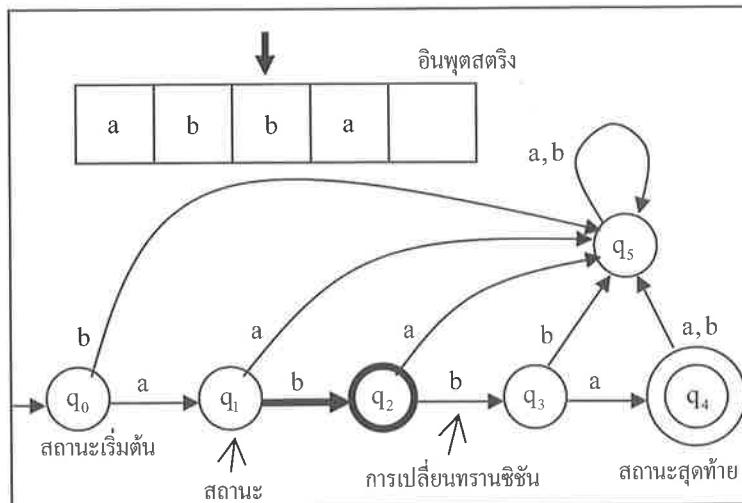
(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

ไฟโนต์อโตมาตานี้เป็นอโตมาตาที่คาดเดาได้ (DFA) โดยมีสถานะเริ่มต้น q_0 และมีสถานะสุดท้ายที่ q_4 นอกจากนี้จะเห็นได้ว่ามีสถานะหนึ่งที่เมื่อการทำงานของ DFA มาก禹ดอยู่แล้วจะไม่มีทางออกจากสถานะนั้นได้เลย นั่นคือ q_5 เราเรียกสถานะนี้ว่า สถานะกับดัก (Trap State)

การทำงานของ DFA จะเริ่มต้นที่สถานะเริ่มต้น q_0 จากนั้นอ่านอินพุตตัวแรกเข้ามา (ซึ่งคือตัวอักษร a) และเปลี่ยนสถานะไปอยู่ที่ q_1

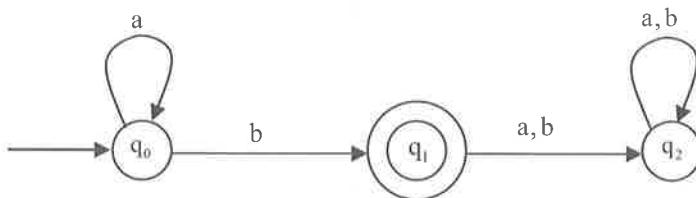


หลังจากนั้น อ่านอินพุต bba เข้ามาทีละตัว และเปลี่ยนสถานะไปยัง q_2 , q_3 และ q_4 ตามลำดับ



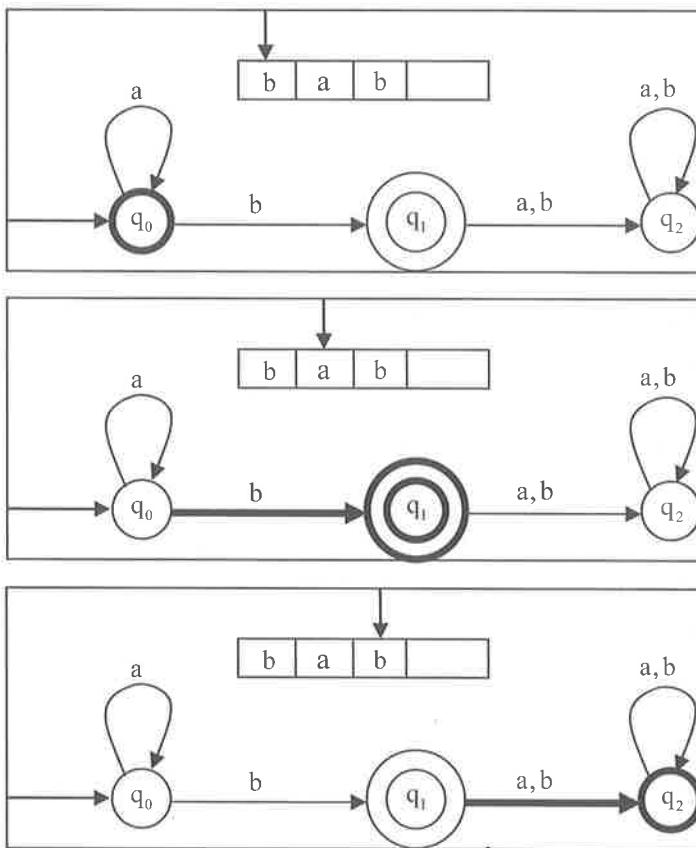
เนื่องจากอินพุตสตริง $abba$ สามารถทำให้ไฟไนต์อัตโนมາตາที่คาดเดาได้ไปหยุดอยู่ที่สถานะสุดท้าย q_4 ได้ดังนั้นสตริง $abba$ จึงถูกยอมรับโดยอัตโนมາตานี้

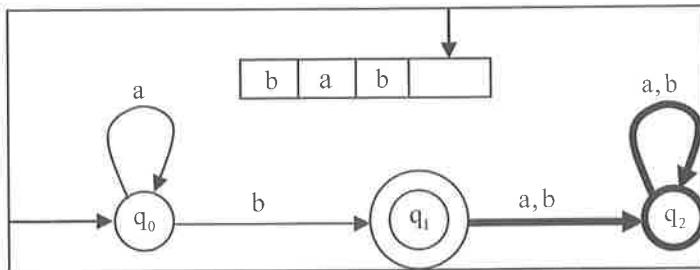
ตัวอย่างที่ 2.3 จงแสดงการทำงานของไฟไนต์อัตโนมາต้าที่คาดเดาได้ดังต่อไปนี้ โดยกำหนดให้อินพุตสตริงคือ bab



รูปที่ 2.3 ไฟไนต์อัตโนมາต้าที่คาดเดาได้

อัตโนมາตานี้มีสถานะเริ่มต้นที่ q_0 และมีสถานะลิ้นสุดที่ q_1 โดยมี q_2 ทำหน้าที่เป็นสถานะกับดัก การทำงานของอัตโนมາตามีการป้อนอินพุตสตริง bab สามารถจำลองได้ดังนี้





เนื่องจากอินพุตสตริง bab นำอโตมาตาไปหยุดอยู่ที่สถานะ q_2 ซึ่งไม่ใช่สถานะสุดท้าย ดังนั้นอินพุตสตริง bab จึงถูกปฏิเสธโดยอโตมาตานี้

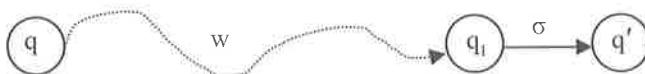
นอกเหนือจากทรานชิชันฟังก์ชันแล้ว เรายังต้องศึกษานิยามของคำอีกด้านนึง นั่นก็คือ ทรานเชิชันฟังก์ชันขยาย (Extended Transition Function) ซึ่งใช้แทนด้วยสัญลักษณ์ δ^* :

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

โดยเราสามารถนิยาม δ^* แบบเวียนบังเกิด (Recursive Definition) ได้ดังนี้

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

$$\delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$$

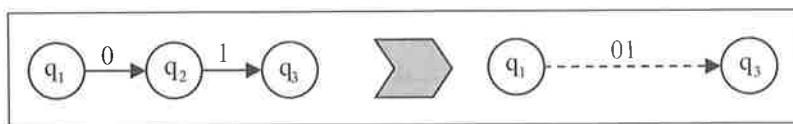


$$\text{รูปที่ 2.4 } \delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$$

สตริง $w\sigma$ ถูกแบ่งออกเป็น 2 ส่วน คือ w และ σ โดยสตริง w อยู่ในทรานชิชันฟังก์ชันซึ่งมีสถานะที่เกิดจากทรานชิชันฟังก์ชันขยาย $\delta^*(q, w)$ จากนั้นย่อสตริง w ต่อไปเรื่อย ๆ ตามความสัมพันธ์แบบเวียนบังเกิด จนมาหยุดที่กรณีพื้นฐาน คือ $\delta^*(q, \varepsilon) = q$ จะสังเกตเห็นว่าการอธิบายทรานชิชันฟังก์ชันโดยอาศัยความสัมพันธ์แบบเวียนบังเกิดไม่จำเป็นต้องอธิบายว่าสถานะที่ได้จากการทำทรานชิชันขยายกับสตริงย่ออยู่นั้นได้สถานะของมาเป็นอะไร เช่น $\delta^*(q, w)$ ยกเว้นกรณีพื้นฐาน คือ $\delta^*(q, \varepsilon) = q$ ดังนั้นจึงไม่มีความจำเป็นที่จะต้องอธิบายสถานะ q_1 จากรูปว่ามีที่มาอย่างไร เนื่องจากสถานะ q_1 จะไม่ปรากฏอยู่ในทรานชิชันฟังก์ชันแบบขยายนั้นเอง สำหรับตัวอย่างของทรานชิชันฟังก์ชันแบบขยายสามารถยกตัวอย่างได้ดังนี้

$$\delta(q_1, 0) = q_2, \quad \delta(q_2, 1) = q_3$$

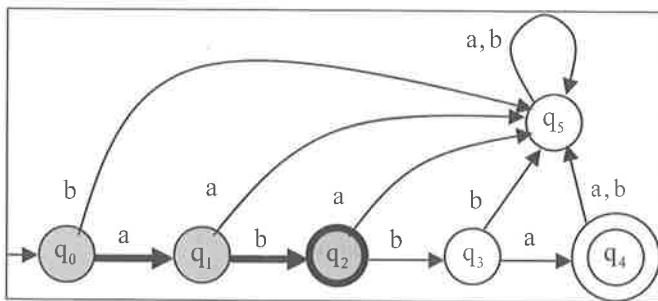
เราสามารถเขียนໄດ້ในรูปแบบย่อ $\delta^*(q_1, 01) = q_3$



$$\text{รูปที่ 2.5 } \delta^*(q_1, 01) = q_3$$

ดังนั้นหากพบรานชิชันฟังก์ชันแบบขยาย $\delta^*(q_a, w) = q_b$ โดยที่ $w \in \Sigma^*$ เราสามารถทราบได้ว่ามีทาง (Path) ที่เชื่อมระหว่างสถานะ q_a ไปยังสถานะ q_b โดยใช้อินพุตสตริง w นอกจากนี้เราระไม่จำเป็นต้องทราบสถานะที่อยู่ระหว่างทางเพื่อใช้ในการเปลี่ยนสถานะจาก q_a มาเป็น q_b

ตัวอย่างที่ 2.4 จงหาผลลัพธ์ของ $\delta^*(q_0, ab)$ จากไฟฟ้าอิเล็กทรอนิกส์ที่แสดงในรูปดังต่อไปนี้ โดยใช้尼ยามเวียนบังเกิดของранชิชันฟังก์ชันขยาย



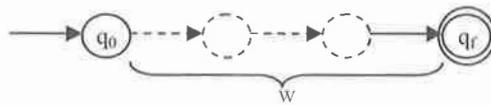
รูปที่ 2.6 ไฟฟ้าอิเล็กทรอนิกส์

จากอิเล็กทรอนิกส์ที่แสดงดังรูป เริ่มต้นที่สถานะเริ่มต้นคือ q_0 จากนั้นรับตัวอักษร a และ b เพื่อจะไปหยุดอยู่ที่สถานะ q_1 และ q_2 ตามลำดับ ดังนี้เราสามารถทราบทันทีว่าผลลัพธ์ของ $\delta^*(q_0, ab)$ คือสถานะ q_2 อย่างไรก็ตาม ตัวอย่างนี้ต้องการฝึกความเข้าใจเรื่องนิยามแบบเวียนบังเกิดของранชิชันฟังก์ชันแบบขยาย ดังนั้นคำตอบของตัวอย่างนี้สามารถแสดงวิธีทำได้ดังนี้

$$\begin{aligned}
 \delta^*(q_0, ab) &= \delta(\delta^*(q_0, a), b) \\
 &= \delta(\delta(\delta^*(q_0, \epsilon), a), b) \\
 &= \delta(\delta(q_0, a), b) \\
 &= \delta(q_1, b) \\
 &= q_2
 \end{aligned}$$

2.2 ความสัมพันธ์ระหว่างภาษาและอิเล็กทรอนิกส์

อิเล็กทรอนิกส์ M จะรับภาษา L ก็ต่อเมื่อสำหรับสตริง w ทุก ๆ ตัวที่เป็นสมาชิกของภาษา L อิเล็กทรอนิกส์ M จะต้องรับสตริง w โดยเริ่มที่สถานะเริ่มต้น (Start State) จากนั้นท่องไปในอิเล็กทรอนิกส์แล้วไปถึงสุดที่สถานะสุดท้าย (Final States) ของอิเล็กทรอนิกส์ M ดังรูป



รูปที่ 2.7 การทำงานของอโตมาตา M

นิยาม 2.2.1 : สำหรับไฟน์เนตออโตมาตา $M = (Q, \Sigma, \delta, q_0, F)$

ภาษาที่ยอมรับโดย M หรือ $L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}$

หรือ $L(M) = \{\text{สตริงซึ่งสามารถขับเคลื่อนของอโตมาตา } M \text{ จากสถานะเริ่มต้นเข้าสู่สถานะสุดท้าย}\}$

รูปที่ 2.8 นิยามของ $L(M)$

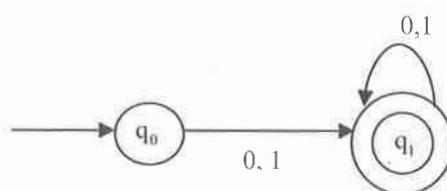
หากมีสตริง w “ตัวได้ตัวหนึ่ง” ของภาษา L ไม่สามารถท่องไปในอโตมาตา M แล้วไปถึงสุดที่สถานะสุดท้าย (Final State) ได้ สามารถสรุปได้ว่าอโตมาตา M ไม่รับภาษา L นอกจากนี้ เราใช้สัญลักษณ์ $\overline{L(M)}$ แทนภาษาที่อโตมาตา M ยอมรับ และใช้ $\overline{L(M)}$ แทนภาษาที่อโตมาตา M ไม่ยอมรับ

นิยาม 2.2.2 : ภาษาที่ปฏิเสธโดยอโตมาตา M หรือ $\overline{L(M)}$ ถูกนิยามโดย

$$\overline{L(M)} = \{w \in \Sigma^* : \delta^*(q_0, w) \notin F\}$$

รูปที่ 2.9 นิยามของ $\overline{L(M)}$

ตัวอย่างที่ 2.5 กำหนดให้ไฟน์เนตออโตมาตา M_1 คือ

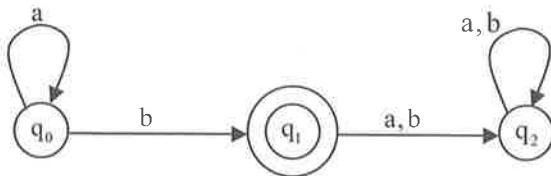
รูปที่ 2.10 ไฟน์เนตออโตมาตา M_1

จงหาภาษาที่ยอมรับโดย M_1

จาก DFA ดังกล่าว จะเห็นได้ว่าขอเพียงให้มีอินพุตเป็น 0 หรือ 1 ก็สามารถทำให้ M_1 ยอมรับได้ ดังนั้น DFA M_1 จึงรับอินพุตสตริงที่ประกอบด้วย 0 และ/หรือ 1 ที่มีความยาวของสตริงตั้งแต่ 1 ตัวอักษรขึ้นไป

$$L(M_1) = \{w \mid w \in \{0, 1\}^* - \varepsilon\}$$

ตัวอย่างที่ 2.6 กำหนดให้ไฟฟ้าในต่ออโตมาตา M นิยามดังนี้



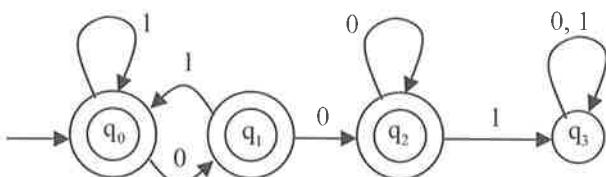
รูปที่ 2.11 ไฟฟ้าในต่ออโตมาตา M

จงหาภาษาที่ยอมรับโดย M

เนื่องจากการที่จะเคลื่อนที่จากสถานะ q_0 มาถึงสถานะสุดท้าย q_1 ได้ ออโตมาตาจะต้องรับอินพุต b เข้ามา 1 ตัว ส่วนสถานะ q_2 นั้นถือว่าเป็นสถานะกับดัก ดังนั้นภาษาที่ยอมรับโดยไฟฟ้าในต่ออโตมาตา M จึงนิยามดังนี้

$$L(M) = \{a^n b : n \geq 0\}$$

ตัวอย่างที่ 2.7 กำหนดให้ไฟฟ้าในต่ออโตมาตา M นิยามดังนี้



รูปที่ 2.12 ไฟฟ้าในต่ออโตมาตา M

จงหาภาษาที่รองรับโดยอโตมาตา M ?

สำหรับวิธีการแก้ปัญหานี้ควรจะพิจารณาสถานะที่ไม่ใช่สถานะสุดท้าย นั่นคือ q_3 การที่ไฟฟ้าในต่ออโตมาตา M จะழุดอยู่ที่สถานะ q_3 ได้ ออโตมาตาจะต้องรับอินพุตสตริงเข้ามาเป็น 001 ส่วนอินพุตสตริงรูปแบบอื่นจะยอมรับโดยอโตมาตา M ทั้งหมด ดังนั้นภาษาที่รองรับโดยอโตมาตา M คือ

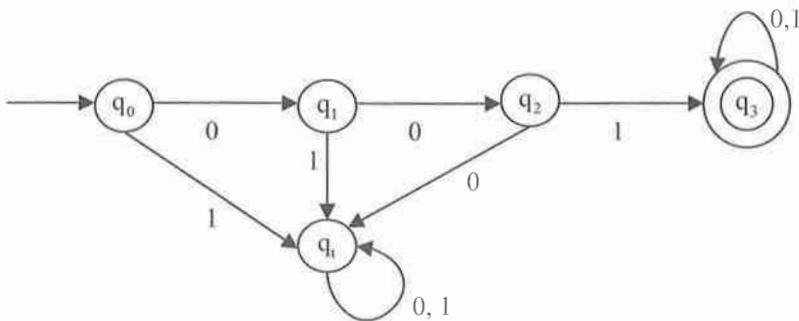
$$L(M) = \{w \in \{0, 1\}^* \mid w \text{ ไม่มีสตริงย่อย } 001 \text{ อยู่ภายใน}\}$$

ตัวอย่างที่ 2.8 กำหนดให้ไฟฟ้าในต่ออโตมาตา M นิยามดังนี้

$$L(M) = \{w \mid w \text{ เริ่มต้นด้วย } 001 \text{ และ } w \text{ ประกอบด้วยตัวอักษร } \{0, 1\} \text{ เพ่านั้น}\}$$

จงหา DFA ของ M ?

เราสามารถใช้ DFA อธิบาย $L(M)$ ได้ 2 วิธี คือ อธิบายตามนิยามของอัตโนมัติ หรือวัดทราบชิ้นกราฟ เพื่อให้ง่ายต่อการเข้าใจ จึงนำกราฟมาอธิบาย $L(M)$ ดังรูป



รูปที่ 2.13 ไฟไนต์อัตโนมัติ M

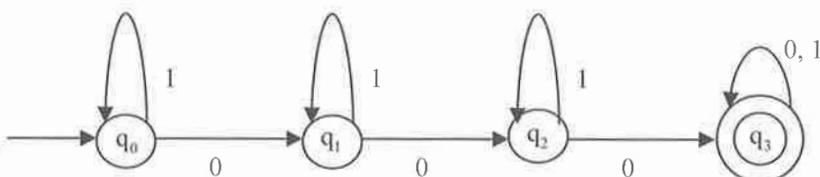
จากรูป สามารถสรุปได้ว่า q_i คือสถานะกับดัก ทั้งนี้เราใช้สถานะกับดักเพื่อใช้เป็นสถานะที่แสดงว่าสตริงที่รับเข้ามานั้นไม่ถูกยอมรับโดย DFA M

นิยาม 2.2.3 : ภาษา L จะถูกเรียกว่า ภาษาเรกูลาร์ (Regular Language) ก็ต่อเมื่อ L เป็นไฟไนต์อัตโนมัติที่คาดเดาได้ (DFA) ที่ยอมรับภาษา L

ตัวอย่างที่ 2.9 จงพิสูจน์ว่าภาษา L ดังต่อไปนี้เป็นภาษาเรกูลาร์

$$L = \{w \mid w \in \{0, 1\}^* \text{ และ } w \text{ มีจำนวนของ } 0 \text{ มากกว่า } 2 \text{ ตัวเสมอ}\}$$

พิสูจน์ : พิสูจน์โดยการสร้าง DFA สำหรับภาษา L ดังนี้

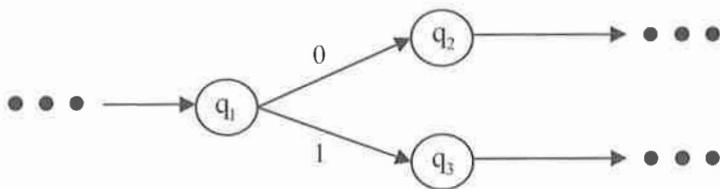


รูปที่ 2.14 DFA สำหรับภาษา L

สำหรับภาษาเรกูลาร์จะอธิบายโดยละเอียดอีกครั้งในบทต่อไป

2.3 ไฟไนต์อัตโนมัติที่คาดเดาไม่ได้

จากไฟไนต์อัตโนมัติที่ผ่านมา จะสังเกตได้ว่าที่สถานะใด ๆ สำหรับอินพุต 1 ค่า อัตโนมัติจะเขื่อนโยงไปยังอีกสถานะเพียงสถานะเดียวเท่านั้น เช่น



รูปที่ 2.15 สักขณะการทำงานของ DFA

ที่ q_1 เมื่ออินพุตเข้ามาเป็น 0 ออโตมาตาจะเปลี่ยนสถานะไปยัง q_2 เท่านั้น ทำนองเดียวกัน ที่ q_1 สำหรับอินพุต 1 ก็จะเปลี่ยนสถานะไปยัง q_3 เท่านั้น นี่คือสิ่งที่เป็นเอกลักษณ์ของไฟโนต์อโตมาตาที่คาดเดาได้ (Deterministic Finite Automata : DFA)

นิยาม 2.3.1 : ไฟโนต์อโตมาตาที่คาดเดาไม่ได้ (Nondeterministic Finite Automata : NFA) แทนด้วยองค์ประกอบห้อง 5 คือ $(Q, \Sigma, \delta, q_0, F)$ โดยที่ Q, Σ, q_0 และ F มีความหมายเช่นเดียวกับนิยามของ DFA นั่นคือ

Q คือ เซตของสถานะ (State) ของออโตมาตา ซึ่งเป็นเซตที่จำกัด

Σ คือ เซตของตัวอักษร (Alphabet) ซึ่งเป็นเซตที่จำกัด

q_0 คือ สถานะเริ่มต้นของออโตมาตา ซึ่ง $q_0 \in Q$

F คือ เซตของสถานะสุดท้าย (Final State) โดยที่ $F \subseteq Q$ และ F เป็นสถานะที่ออโตมาตาพยายามรับอินพุตที่ป้อนเข้ามา

สำหรับ δ คือ เซตของทรานซิชันฟังก์ชัน (Transition Function) ซึ่งนิยามโดย

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \text{เพาเวอร์เซตของ } Q \text{ หรือ } 2^Q$$

ความแตกต่างระหว่าง δ ของ DFA กับ δ ของ NFA คือ

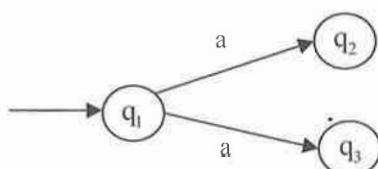
1. NFA สามารถเปลี่ยนสถานะได้โดยไม่จำเป็นต้องรับอินพุตใด ๆ เช่น

$$q_1 \times \{\epsilon\} \rightarrow q_2 \quad (\text{เรียกทรานซิชันแบบนี้ว่า } \epsilon\text{-transition})$$

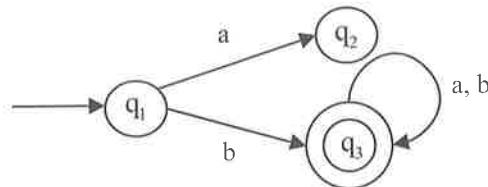


2. จำนวนสถานะปลายทางของ NFA (เพาเวอร์เซตของ Q) มีมากกว่าจำนวนสถานะปลายทางของ DFA (ซึ่งมีได้เพียง 1 สถานะเท่านั้น) หรือกล่าวไก่อกนัยหนึ่งว่าสถานะใด ๆ ของ NFA สามารถเปลี่ยนไปยังสถานะอื่น ๆ ได้มากกว่า 1 สถานะสำหรับอินพุตเดียวกัน เช่น

$$q_1 \times \{a\} \rightarrow \{q_2, q_3\}$$

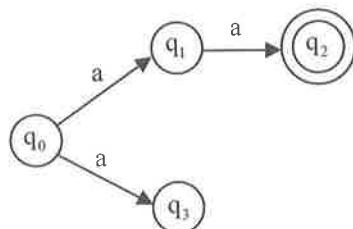


3. สำหรับสถานะใด ๆ ของ NFA เราไม่จำเป็นต้องกำหนดทรานซิชันให้ครบสำหรับทุก ๆ อินพุต ภายในเซต Σ เช่น ให้ $\Sigma = \{a, b\}$



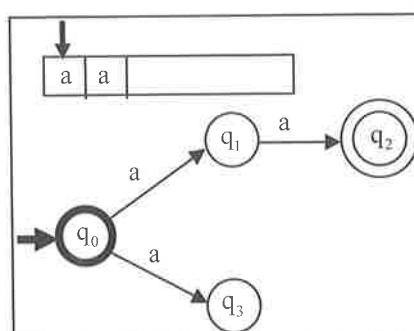
สังเกตว่าสถานะ q_2 ไม่ได้กำหนดทรานซิชันสำหรับอินพุตใด ๆ ตั้งนั้นหากมีอินพุตเกิดขึ้นในขณะที่ NFA ทำงานที่สถานะ q_2 ออโตมาตาจะหยุดทำงานและปฏิเสธอินพุตสตริงทันที

ตัวอย่างที่ 2.10 กำหนดให้ไฟน์ในต่ออโตมาตาที่คาดเดาไม่ได้ (NFA) นิยามดังทรานซิชันกราฟ

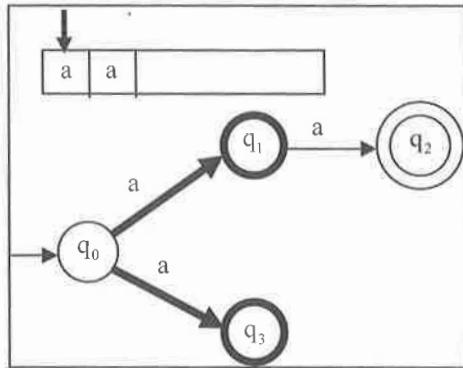


รูปที่ 2.16 ไฟน์ในต่ออโตมาตาที่คาดเดาไม่ได้

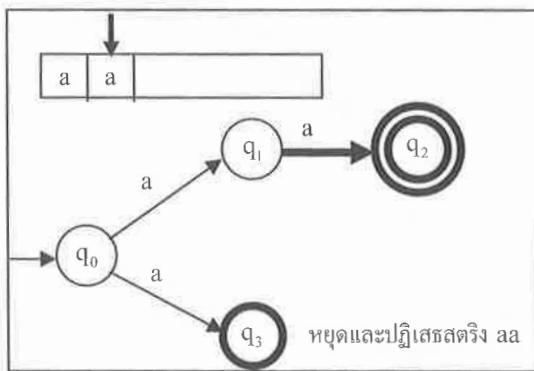
จงแสดงการทำงานของออโตมาตานี้สำหรับอินพุตสตริง aa ?



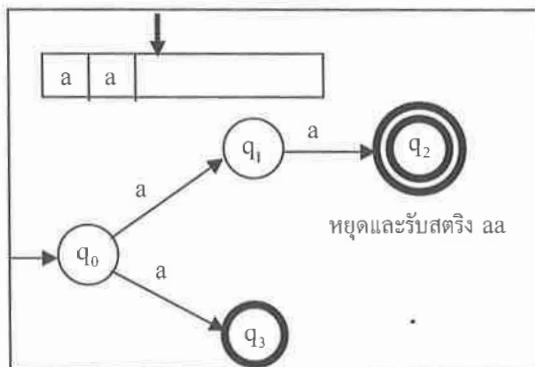
ขั้นตอนแรก ออโตมาตารับตัวอักษร a ตัวแรกเข้ามา โดยเริ่มทั้งที่สถานะเริ่มต้น q_0 เนื่องจากสถานะ q_0 มีการตอบสนองกับอินพุต a 2 ทาง คือสามารถเปลี่ยนสถานะไปยังสถานะ q_1 หรือ q_3 ได้ การเปลี่ยนสถานะจึงแยกออกเป็น 2 ทางดังรูป



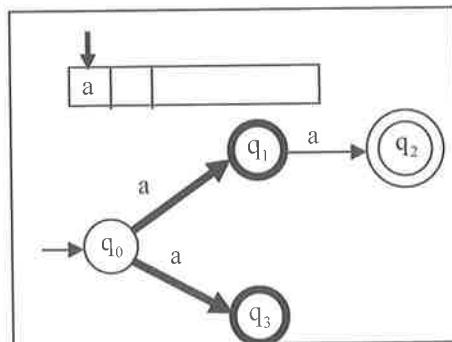
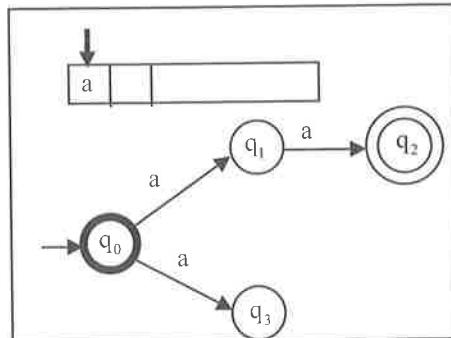
หลังจากนั้นตัวอักษร a ตัวที่สองถูกอ่านเข้ามา ซึ่งทำให้สถานะ q_1 เปลี่ยนสถานะไปเป็น q_2 แต่ไม่สามารถทำให้สถานะ q_3 เปลี่ยนสถานะได้ เนื่องจากสถานะ q_3 ไม่รองรับอินพุตได ๆ ดังนั้นจึงถือได้ว่าอินพุตสตริง aa ไม่ถูกยอมรับโดยสถานะ q_3



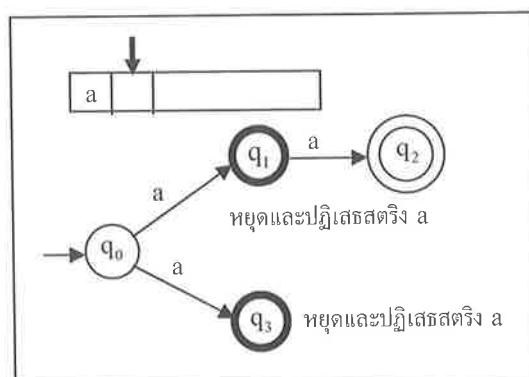
เมื่ออินพุตสตริงถูกอ่านหมด สถานะสุดท้ายของออโตมาตามี 2 สถานะ คือ q_2 และ q_3 โดยเราทราบอยู่แล้วว่าสถานะ q_3 ไม่ยอมรับสตริง aa อย่างไรก็ตาม สถานะ q_2 เป็นสถานะสุดท้าย (Final State) ของออโตมาตา ดังนั้นจึงทำให้ออโตมาตานี้ยอมรับอินพุตสตริง aa



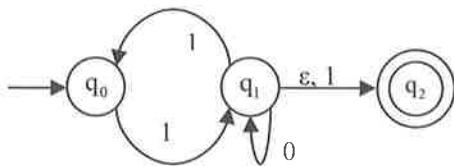
ตัวอย่างที่ 2.11 จงแสดงการทำงานของอัลกอริทึมในตัวอย่างก่อนหน้านี้สำหรับอินพุตสตริง a ? ออโตมาตารับตัวอักษร a เข้ามา โดยเริ่มต้นที่สถานะเริ่มต้น q_0 และเปลี่ยนสถานะไปยัง q_1 และ q_3



เมื่ออินพุตสตริงถูกอ่านหมด สถานะปลายทางของอัลกอริทึมมี 2 สถานะ คือ q_1 และ q_3 เนื่องจากทั้งสถานะ q_1 และ q_3 ไม่ใช่สถานะสุดท้ายของอัลกอริทึม ดังนั้นอินพุตสตริง a จึงถูกปฏิเสธโดยอัลกอริทึมนี้



ตัวอย่างที่ 2.12 กำหนดให้ไฟฟ้าในต่ออโตมาตา M_2 มีลักษณะดังรูป



รูปที่ 2.17 ไฟฟ้าในต่ออโตมาตา M_2

จงยกตัวอย่างสตริงที่ M_2 ยอมรับ

จากรูป สิ่งที่แสดงให้เห็นว่า M_2 เป็น NFA มีดังนี้

- ที่ q_1 นิยามทรานซิชันสำหรับอินพุต 1 ໄว้ 2 ทาง คือ

$$\delta(q_1, 1) = \{q_0, q_2\}$$

- ที่ q_1 นิยามทรานซิชันสำหรับสตริงว่าง (ϵ)

$$\delta(q_1, \epsilon) = \{q_2\}$$

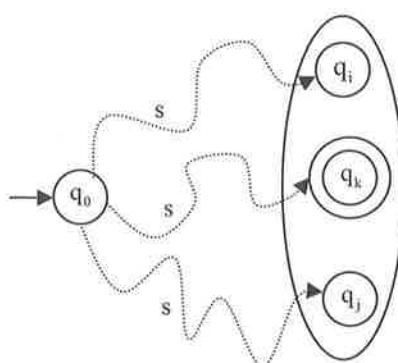
- ที่ q_0 และ q_2 ไม่ได้นิยามทรานซิชันครบถ้วนพุต

สำหรับสตริงที่ M_2 ยอมรับ คือ $\{1, 11, 111, 1111, 10, 100, 1011, 10111, \dots\}$ เป็นที่น่าสังเกตว่า สตริง 11 สามารถนำ M_2 ไปหยุดอยู่ที่ 2 สถานะคือ q_0 และ q_2 แต่เมื่องจาก q_2 เป็นสถานะสุดท้าย ดังนั้น M_2 จึงหยุดและยอมรับสตริง 11

นิยาม 2.3.2 : กำหนดให้ NFA $M = (Q, \Sigma, \delta, q_0, F)$ ภาษาที่ M ยอมรับ อธิบายด้วย

$$L(M) = \{s \in \Sigma^* \text{ โดยที่ } \delta^*(q_0, s) \cap F \neq \emptyset\}$$

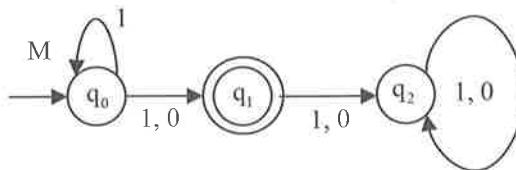
นั่นคือ ภาษา $L(M)$ ประกอบด้วยสตริง s ซึ่งเมื่อป้อนเข้า NFA M ที่สถานะเริ่มต้น q_0 ผลลัพธ์ของสถานะที่ได้จะต้องเป็นสถานะที่อยู่ภายใต้ชุด F (Final State) ดังรูป



รูปที่ 2.18 ภาษา $L(M)$

(ที่มา : Konstantin Busch (<http://www.cse.lsu.edu/~busch/>)

ตัวอย่างที่ 2.13 จงหาภาษาที่ยอมรับโดย NFA ดังต่อไปนี้



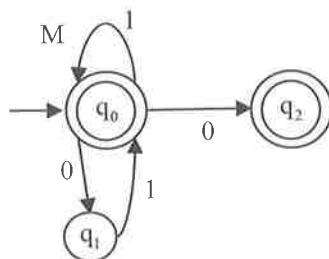
รูปที่ 2.19 NFA

จากรูป จะเห็นได้ว่าสตริงที่ NFA นี้ยอมรับได้นั้น จะต้องเป็นสตริงที่ขึ้นต้นด้วย 1 ติดกันกี่ตัวก็ได้ จากนั้นจะต้องปิดท้ายด้วย 1 หรือ 0 เพียงหนึ่งตัว เช่น $\{1, 0, 11, 10, 111, 110, 1111, 1110, \dots\}$ ดังนั้นจึงสรุปได้ว่า

$$L(M) = \{1^n(1+0) : n \geq 0\}$$

หมายเหตุ : เครื่องหมาย + เมื่อสัญลักษณ์ของนิพจน์เรกูลาร์ (Regular Expression) หมายถึง “หรือ (OR)” ซึ่งผู้อ่านจะได้ศึกษาในบทต่อไป

ตัวอย่างที่ 2.14 จงหาภาษาที่ยอมรับโดย NFA ดังต่อไปนี้



รูปที่ 2.20 NFA

NFA นี้ดูเหมือนจะไม่มีความซับซ้อนอะไรมากนัก แต่การหาภาษาที่ NFA นี้รองรับนั้นไม่ใช่เรื่องง่ายเลย โดยเฉพาะสำหรับผู้เริ่มต้น วิธีการพื้นฐานที่สุดในการหาภาษาที่ NFA รองรับ ในกรณีที่ไม่สามารถจะวิเคราะห์จาก NFA ได้โดยตรง ก็คือการหาสตริงซึ่งถูกสร้างขึ้นจาก NFA ทีละตัว และสังเกตลักษณะของภาษาจากสตริงเหล่านั้น เช่นของสตริงที่ถูกสร้างขึ้นจาก NFA นี้คือ

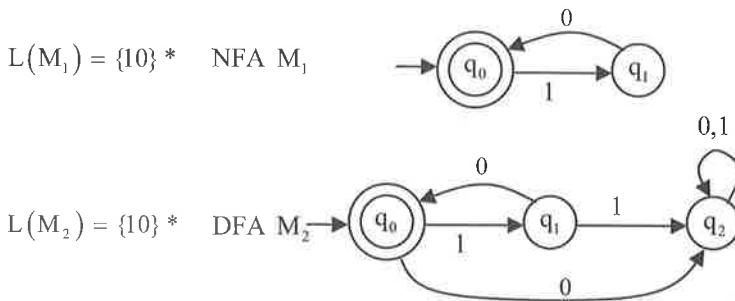
$$L(M) = \{\epsilon, 0, 1, 11, 111, 1111, 01, 011, 0101, 10, 110, 1110, 010, 0110, 01010, 0101101010, \dots\}$$

หากเราสร้างสตริงขึ้นมาหลายตัว เราจะสังเกตเห็นได้ว่าสตริงเหล่านี้ไม่มี 0 อัญเชิญกันเลย ซึ่งหากย้อนกลับไปวิเคราะห์ที่ NFA จะพบว่าการจะเข้าสู่สถานะสุดท้าย q_0 และ q_2 ได้นั้น สตริงที่รับเข้ามานั้นจะต้องไม่มี 0 อัญเชิญกันเลย ดังนั้นจึงสรุปได้ว่าภาษาที่ NFA นี้รองรับคือ

$$L(M) = \{w : w \in \{0, 1\}^* \text{ และ } w \text{ ไม่มี } 0 \text{ อัญเชิญกันเลย}\}$$

2.4 ความเท่าเทียมกันของ DFA และ NFA

ในการเปรียบเทียบความเท่าเทียมกันของอัตโนมາต้าได ๆ จะนำเซตของภาษาที่ได้จากการอัตโนมາต้าทั้งสองมาเปรียบเทียบกัน ถ้าเซตของภาษาที่ได้จากการอัตโนมາต้าทั้งคู่เหมือนกัน ก็แสดงว่าอัตโนมາต้าทั้งสองมีความเท่าเทียมกัน สำหรับการเปรียบเทียบความเท่าเทียมกันของ DFA และ NFA ก็เช่นเดียวกัน จะสามารถสร้าง DFA ที่มีความเท่าเทียมกันกับ NFA ได้เสมอ เช่น



เราจะเห็นได้ว่า NFA M_1 มีความเท่าเทียมกับ DFA M_2 เนื่องจาก $L(M_1) = L(M_2)$

ทฤษฎี : กำหนดให้ $M = (Q, \Sigma, \delta, q_0, F)$ เป็น NFA ที่รองรับภาษา L เราสามารถสร้าง DFA N ซึ่งรองรับภาษาเดียวกันกับ M ได้เสมอ นั่นคือ

$$L(M) = L(N)$$

พิสูจน์ : ดังที่เราทราบมาแล้วว่าภาษา L จะถูกเรียกว่า ภาษาเรกูลาร์ (Regular Language) ก็ต่อเมื่อมีไฟโนต์อัตโนมາต้าที่คาดเดาได้ (DFA) รองรับภาษา L ดังนั้นการพิสูจน์ความเท่าเทียมกันระหว่าง NFA และ DFA ก็คือการพิสูจน์ว่าภาษาที่รองรับโดย NFA เท่าเทียมกับภาษาเรกูลาร์ นั่นคือ

ภาษาที่รองรับโดย NFA = ภาษาเรกูลาร์ (ภาษาที่รองรับโดย DFA)

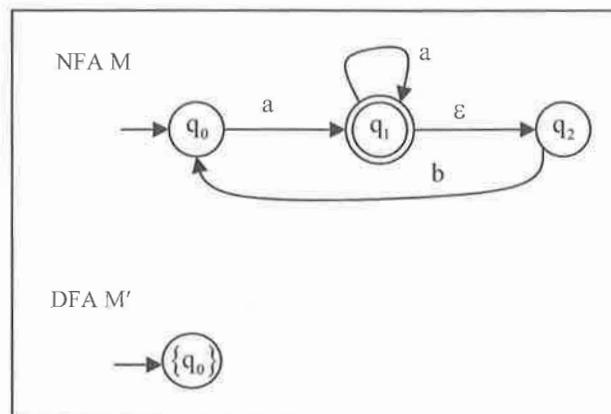
แบ่งการพิสูจน์ออกเป็น 2 ส่วน ดังนี้

1. ภาษาที่รองรับโดย NFA \supseteq ภาษาเรกูลาร์
2. ภาษาที่รองรับโดย NFA \subseteq ภาษาเรกูลาร์

1. ภาษาที่รองรับโดย NFA \supseteq ภาษาเรกูลาร์ เนื่องจาก DFA เป็นเซตย่อยของ NFA ดังนั้นภาษาที่ถูกสร้างจาก DFA (ภาษาเรกูลาร์) จึงถูกยอมรับโดย NFA โดยอัตโนมัติ การพิสูจน์ในกรณีนี้จะไม่น่าสนใจนัก

2. ภาษาที่รองรับโดย NFA \subseteq ภาษาเรกูลาร์ จะพิสูจน์โดยการแปลงทรานซิชันกราฟจาก DFA M ไปเป็น DFA M' ที่เท่าเทียมกัน ตามขั้นตอนดังต่อไปนี้

- (1) นำสถานะเริ่มต้นของ NFA มาเป็นสถานะเริ่มต้นของ DFA เช่น



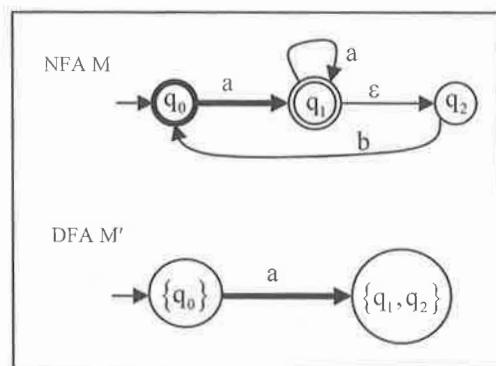
- (2) สำหรับทุก ๆ สถานะ q_i ของ NFA ที่ยังไม่มีทราบซึ้งไปยังสถานะอื่น สำหรับทุก ๆ อินพุต $a \in \Sigma$

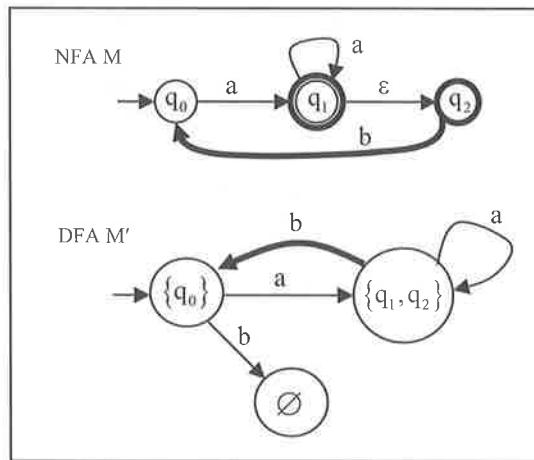
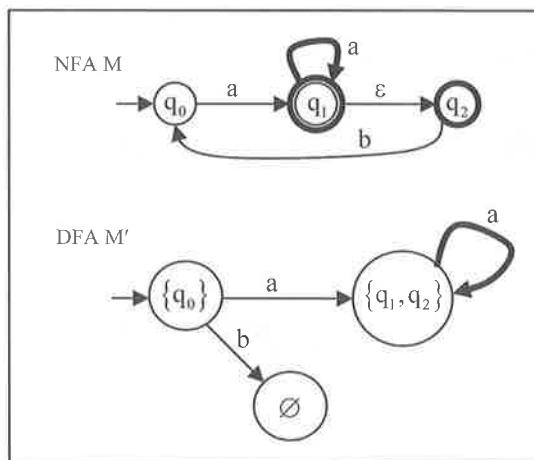
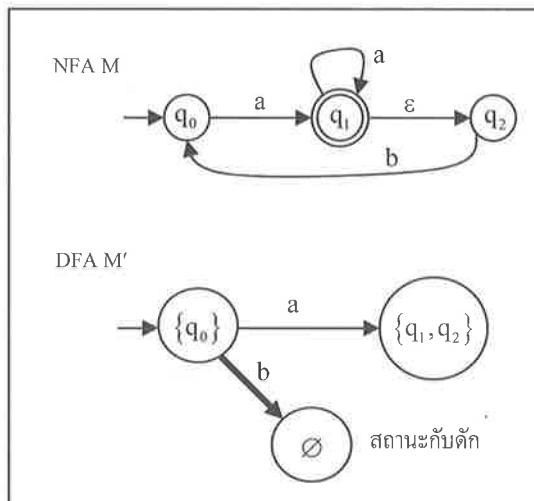
- ค่านวน $\delta^*(q_i, a)$ ของ NFA จากนั้นนำสถานะที่ได้มาญี่เนียนกัน เพื่อให้ได้เป็นสถานะใหม่ของ DFA เช่น

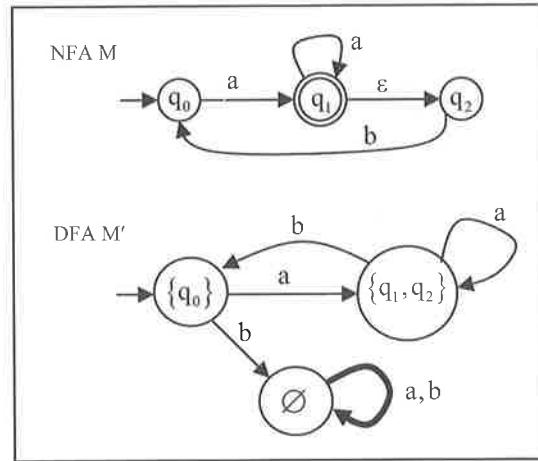
$$\delta^*(q_0, a) = \{q_1, q_2\}$$
 สถานะใหม่ของ DFA จะมีชื่อว่า $\{q_1, q_2\}$
 หาก q_i อยู่ในรูป $\{q_1, q_2, \dots, q_n\}$ ให้ค่านวน

$$\delta^*(q_1, a) \cup \delta^*(q_2, a) \cup \dots \cup \delta^*(q_n, a)$$
- เชื่อมทราบซึ้งจากสถานะ q_i มายังสถานะใหม่ที่สร้างขึ้น จากนั้นกำกับเส้นทางนั้นด้วยอินพุต a

จากตัวอย่าง สามารถแปลง NFA M ไปเป็น DFA M' ตามวิธีในขั้นตอนที่ 2 ได้ดังนี้



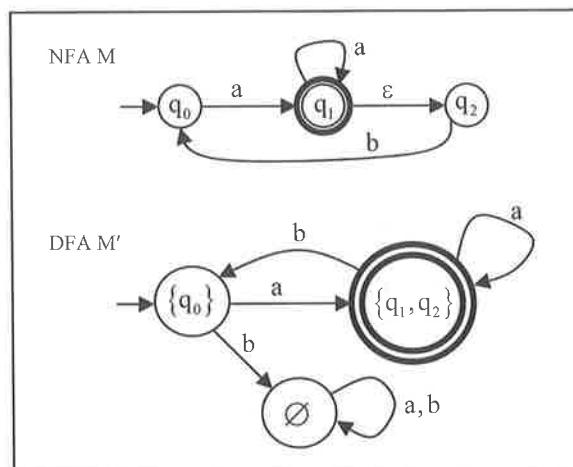




(3) สำหรับทุก ๆ สถานะ q_j ของ DFA ที่มีสถานะสุดท้าย F (Final State) ของ NFA เป็นสมาชิกภายในชื่อสถานะนั้น เช่น จากตัวอย่างนี้ $F = \{q_1\}$ และ

$$Q \text{ ของ DFA } = \{\{q_0\}, \{q_1, q_2\}, \emptyset\}$$

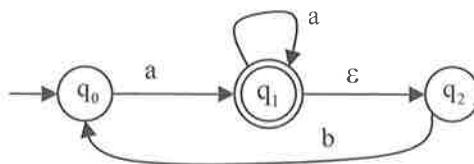
ให้กำหนดสถานะภายใน Q ที่มี q_1 ปรากฏอยู่เป็นสถานะสุดท้ายของ DFA ดังนั้นเซตของสถานะสุดท้ายจากตัวอย่างคือ $\{\{q_1, q_2\}\}$



(4) หาก M ยอมรับสตริงว่าง (ϵ) สถานะเริ่มต้นของ DFA จะต้องถูกกำหนดให้เป็นสถานะสุดท้ายด้วย

จากการพิสูจน์ทั้งสองกรณี สามารถสรุปได้ว่าภาษาที่ยอมรับโดย NFA = ภาษาเรกูลาร์ (ภาษาที่ยอมรับโดย DFA) ดังนั้น NFA จึงมีความสามารถเท่าเทียมกันกับ DFA

สำหรับการแปลงอโตมาตา NFA ไปเป็น DFA จะต้องใช้ความรอบคอบและการตรวจสอบทราบชิ้นๆ ที่เป็นไปได้ทั้งหมดอย่างละเอียด ซึ่งอาจเกิดข้อผิดพลาดขึ้นได้ในระหว่างการแปลงอโตมาตา วิธีการหนึ่งที่จะช่วยอำนวยความสะดวกในการแปลง NFA ไปเป็น DFA ก็คือ การสร้างตารางสำหรับทราบชิ้นๆ ที่เป็นไปได้ของอโตมาตา NFA ดังจะอธิบายโดยใช้ตัวอย่างเดิมดังนี้

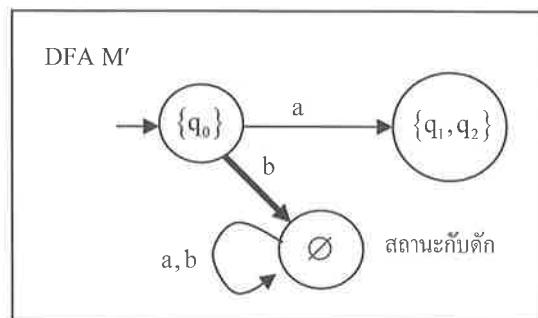


รูปที่ 2.21 NFA M

จาก NFA M สามารถสร้างตารางทราบชิ้นๆ ได้ดังนี้

	a	b
q_0	$\{q_1, q_2\}$	\emptyset
q_1	$\{q_1, q_2\}$	$\{q_0\}$
q_2	\emptyset	$\{q_0\}$

หลังจากนั้น สามารถใช้ตารางทราบชิ้นๆ แทน NFA M โดยที่ไม่ต้องพิจารณา NFA M อีกต่อไป เริ่มต้นจากการสร้างสถานะเริ่มต้น $\{q_0\}$ ก่อน จากนั้นพิจารณาจากตารางทราบชิ้นๆ จะเห็นได้ว่าหากอินพุตเข้ามาเป็น a สถานะจะถูกเปลี่ยนไปเป็น $\{q_1, q_2\}$ และหากอินพุตเข้ามาเป็น b สถานะจะถูกเปลี่ยนไปเป็น \emptyset ดังนั้นจึงเริ่มสร้าง DFA M' ดังนี้



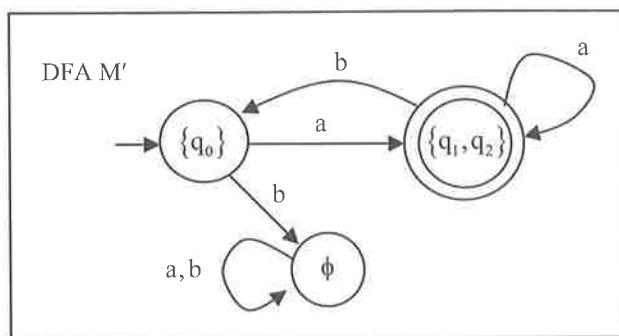
ขั้นตอนต่อไป พิจารณาที่สถานะ $\{q_1, q_2\}$ ควบคู่กับตารางทราบชิ้นๆ จะทราบได้ว่าสำหรับอินพุต a สถานะ q_1 จะเปลี่ยนไปเป็น $\{q_1, q_2\}$ และสถานะ q_2 จะถูกเปลี่ยนไปเป็น \emptyset นำสถานะทั้งสองที่ได้มา�ูเนียชน (Union) กันเป็นสถานะปลายทางสำหรับทราบชิ้นๆ นั่นคือ

$$\begin{aligned}\delta(\{q_1, q_2\}, a) &= \{q_1, q_2\} \cup \phi \\ &= \{q_1, q_2\}\end{aligned}$$

สำหรับอินพุต b สามารถใช้วิธีเดียวกันเพื่อหาtranซิชันที่เกิดขึ้น

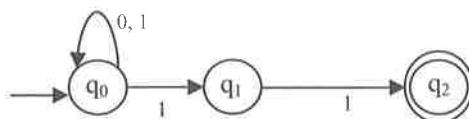
$$\begin{aligned}\delta(\{q_1, q_2\}, b) &= \{q_0\} \cup \{q_0\} \\ &= \{q_0\}\end{aligned}$$

หลังจากนั้น นำtranซิชันที่ได้ไปสร้าง DFA ต่อจนเสร็จ ดังนี้



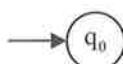
ดังนั้นจะเห็นได้ว่าการสร้างตารางtranซิชันสำหรับ NFA ประกอบการแปลง NFA ไปเป็น DFA ช่วยอำนวยความสะดวกและลดข้อผิดพลาดที่อาจเกิดขึ้นได้

ตัวอย่างที่ 2.15 จงแปลง NFA ดังต่อไปนี้ให้เป็น DFA

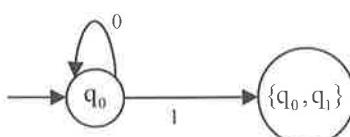


รูปที่ 2.22 NFA

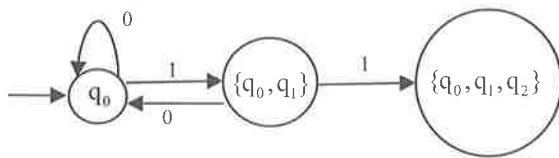
ขั้นตอนที่ 1 : สร้างสถานะเริ่มต้น q_0 ของ DFA



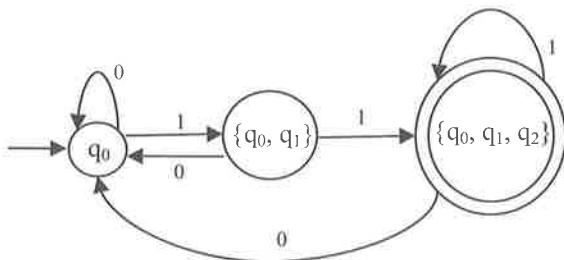
ขั้นตอนที่ 2 :



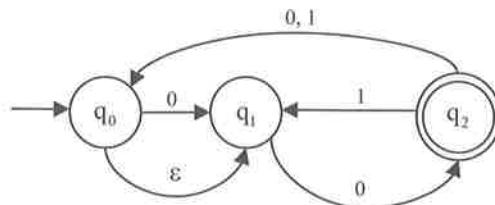
ขั้นตอนที่ 3 :



ขั้นตอนที่ 4 :

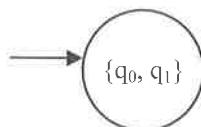


ตัวอย่างที่ 2.16 จงแปลง NFA ดังต่อไปนี้ให้เป็น DFA



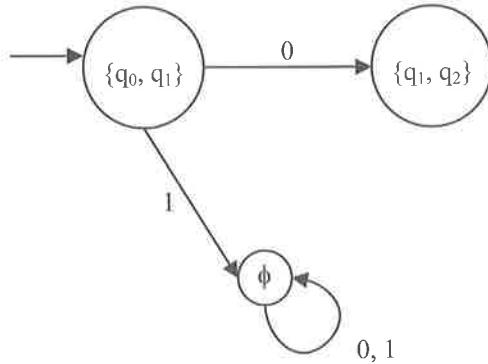
จําบลัง 2.23 NFA

ขั้นตอนที่ 1 : สร้างสถานะเริ่มต้นของ DFA

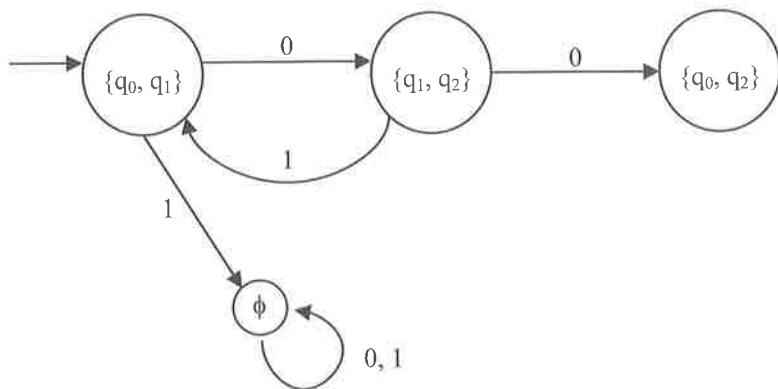


เนื่องจากสถานะ q_0 มีทางเชื่อมไปยังสถานะ q_1 โดยใช้สตริงว่าง (ϵ) ดังนั้นเราจึงสร้างสถานะเริ่มต้นของ DFA เป็น $\{q_0, q_1\}$

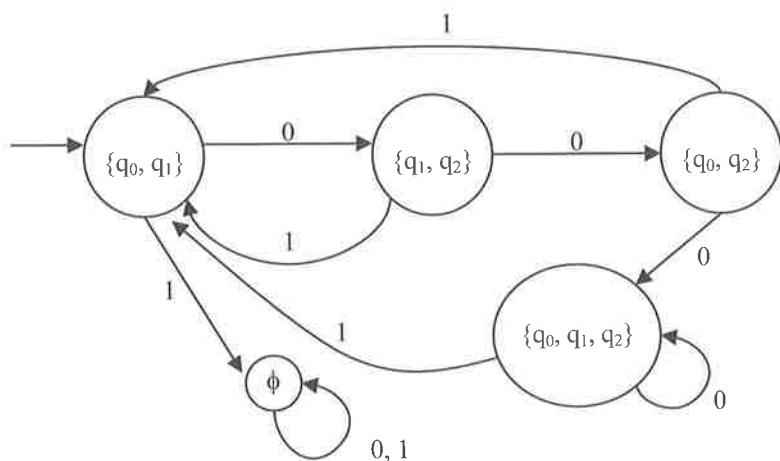
ขั้นตอนที่ 2 : สำหรับแต่ละอินพุตของสถานะ q_0 และ q_1 คำนวณหาสถานะปลายทางสำหรับทั้งสองสถานะ



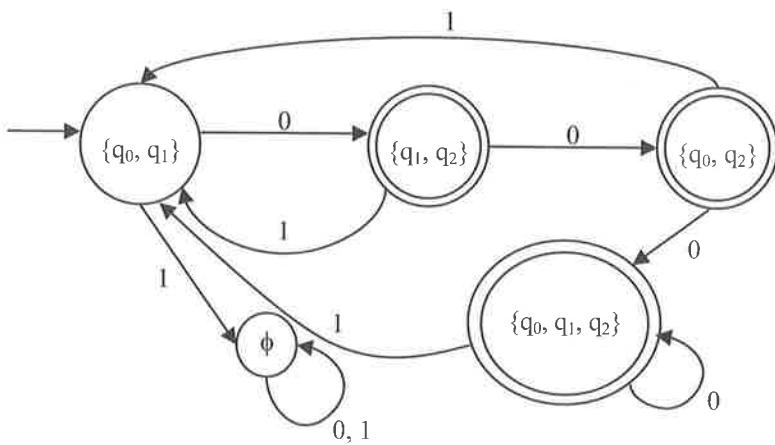
ขั้นตอนที่ 3 :



ขั้นตอนที่ 4 :



ขั้นตอนที่ 5 :



แบบฝึกหัด

1. ไฟน์ต่ออโตมาตาที่คาดเดาได้แตกต่างจากไฟน์ต่ออโตมาตาที่คาดเดาไม่ได้อย่างไร?

2. จงสร้างไฟน์ต่ออโตมาตาสำหรับภาษา L

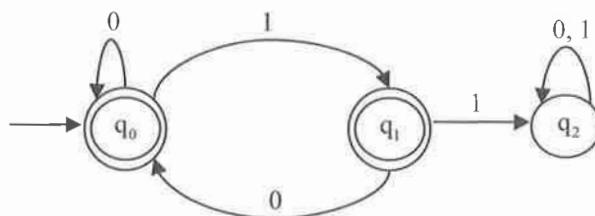
$$L = \{x \in \{0,1\}^* \mid x \text{ ไม่มีสตริงยาว } 100\}$$

3. จงยกตัวอย่างสตริงที่เป็นสมาชิกภายในภาษา L

$$L = \{x \in \{0,1\}^* \mid x \text{ ลงท้ายด้วย } 01\}$$

4. จงเขียนไฟน์ต่ออโตมาตาเพื่อรับคำส่วน (Reserve Words) ในภาษา C คือ include, int, float, main, if โดยสร้างเพียงอโตมาตาเดียวเท่านั้น

5. จงอธิบายภาษาที่รองรับโดยไฟน์ต่ออโตมาตาดังต่อไปนี้



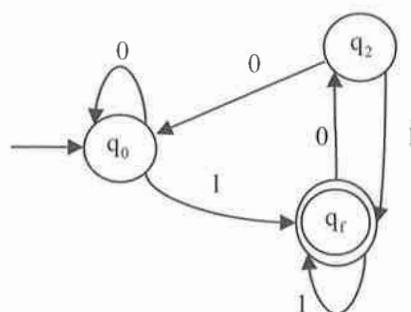
รูปที่ 2.24 ไฟน์ต่ออโตมาตา

6. จงสร้างไฟน์ต่ออโตมาตาที่คาดเดาไม่ได้สำหรับภาษาในข้อ 5

7. จงสร้างไฟน์ต่ออโตมาตาที่คาดเดาได้ สำหรับภาษาดังต่อไปนี้

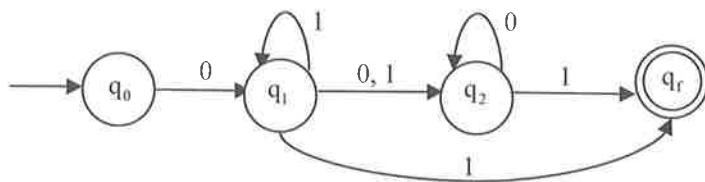
$$L = \{w \in \{0, 1\}^* \mid w \text{ ลงท้ายด้วย } 001\}$$

8. จงอธิบายภาษาที่รองรับโดยไฟน์ต่ออโตมาตาดังต่อไปนี้



รูปที่ 2.25 ไฟน์ต่ออโตมาตา

9. จงแปลง NFA ดังต่อไปนี้ให้เป็น DFA



รูปที่ 2.26 NFA

10. จงให้เหตุผลว่าเราสามารถสร้างไฟน์ต่ออโตมาตาเพื่อรับสตริงของวงเล็บเปิด-ปิด โดยจะต้องมีจำนวนเท่ากันได้หรือไม่ (ตัวอย่างสตริง เช่น “()”, “()”, “(()())” เป็นต้น)

นิพจน์เรกูลาร์

และไวยากรณ์เรกูลาร์

ในบทนี้ผู้อ่านจะได้ศึกษาเพิ่มเติมเกี่ยวกับภาษาที่ไฟฟ้าในต่ออโตมาตาสามารถรองรับได้ นั่นคือ ภาษาเรกูลาร์ (Regular Language) ภาษาเรกูลาร์เป็นประเภทของภาษาที่มีระดับความซับซ้อนที่เรียบง่ายที่สุดเมื่อเทียบกับภาษาประเภทต่าง ๆ ที่มีอยู่ในศาสตร์ทางคอมพิวเตอร์ ผู้อ่านหลายท่านเคยได้สัมผัสถกันภาษาเรกูลาร์มาแล้วโดยไม่รู้ตัว เช่น $+1.4e-12$ เป็นการแทนตัวเลขศูนย์ที่มีรูปแบบทางวิทยาศาสตร์ (หมายถึง จำนวน $+1.4 \times 10^{-12}$) การแทนตัวเลขศูนย์ดังกล่าวมีหลักเกณฑ์ในการเขียนที่แน่นอนเช่นจะเขียนผิดลำดับไม่ได้ เช่น ถ้าเขียนเป็น $+1.4-e12$ จะไม่มีความหมายและผิดหลักไวยากรณ์ ดังนั้นภาษาทุกภาษาจะต้องมีไวยากรณ์รองรับ เช่นเดียวกับภาษาเรกูลาร์ซึ่งมีไวยากรณ์เรกูลาร์ (Regular Grammar) ช่วยอธิบายวิธีการสร้างประโยค

นอกจากนี้ ผู้อ่านยังจะได้ศึกษาถึงทางเลือกในการอธิบายภาษาเรกูลาร์อีกรูปแบบหนึ่ง นอกเหนือจากไฟฟ้าในต่ออโตมาตาและไวยากรณ์เรกูลาร์ นั่นคือ นิพจน์เรกูลาร์ (Regular Expression) ซึ่งมีลักษณะคล้ายคลึงกับนิพจน์ในทางคณิตศาสตร์ แต่ต่างกันตรงที่นิพจน์เรกูลาร์มีสัญลักษณ์อธิบายภาษาที่แตกต่างจากสัญลักษณ์ของนิพจน์คณิตศาสตร์ เนื่องจากเนื้อหาในบทนี้เป็นส่วนเชื่อมระหว่างบทที่ 2 และบทที่ 4 ดังนั้นผู้อ่านจึงควรใช้ความตั้งใจในการอ่านและทำความเข้าใจทั้งในส่วนที่เป็นทฤษฎีและส่วนที่เป็นตัวอย่าง

3.1 นิพจน์เรกูลาร์

นิพจน์ (Expression) ในทางคณิตศาสตร์คือ การนำตัวเลขหรือตัวแปรหลาย ๆ ตัวมากระทำกระบวนการทางคณิตศาสตร์ เช่น

$$x + (2y - 4)$$

ในแง่มุมของภาษา เราใช้นิพจน์เพื่ออธิบายภาษาหนึ่ง ๆ ว่ามีลักษณะเป็นอย่างไร เช่น ภาษาที่ประกอบด้วยเลข 0 และ 1 โดยที่สตริงจะเป็นต้นด้วยเลข 0 และลงท้ายด้วยเลข 1 สามารถเขียนแทนด้วยนิพจน์

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

เราเรียกนิพจน์ที่ใช้อธิบายภาษาว่า นิพจน์เรกูลาร์ (Regular Expression) ซึ่งคำว่า เรกูลาร์ (Regular) เป็นชื่อประเภทของภาษาที่มีระดับความซับซ้อนที่เรียบง่ายที่สุด ความสัมพันธ์ระหว่างนิพจน์เรกูลาร์ ภาษาเรกูลาร์ และไฟฟ้าในต่ออโตมาตาสามารถอธิบายได้ดังรูป



รูปที่ 3.1 ความสัมพันธ์ระหว่างไฟฟ้าในตัวอัตโนมัติ ภาษาเรกูลาร์ และนิพจน์เรกูลาร์

เราสร้างไฟฟ้าในตัวอัตโนมัติเพื่อรับภาษาเรกูลาร์ และใช้นิพจน์เรกูลาร์ในการอธิบายภาษาเรกูลาร์ว่ามีลักษณะอย่างไร นอกจากนี้ ความสัมพันธ์ระหว่างทั้งสามสิ่งยังเป็นความสัมพันธ์แบบสองทาง นั่นคือ

- หากมีไฟฟ้าในตัวอัตโนมัติ τ ภาษาที่ไฟฟ้าในตัวอัตโนมัตินั้นรองรับจะเป็นภาษาเรกูลาร์ และในทางกลับกัน หากมีภาษาเรกูลาร์ τ จะสามารถสร้างไฟฟ้าในตัวอัตโนมัติขึ้นมารองรับภาษา τ ได้เสมอ
- หากมีนิพจน์เรกูลาร์ τ ภาษาที่นิพจน์เรกูลาร์นั้นอธิบายจะต้องเป็นภาษาเรกูลาร์ และในทางกลับกัน หากมีภาษาเรกูลาร์ τ จะต้องมีนิพจน์เรกูลาร์ซึ่งสามารถอธิบายภาษา τ ได้

นิยาม 3.1.1 : กำหนดให้ Σ แทนเซตของตัวอักษรที่ใช้ในภาษาเรกูลาร์

นิพจน์ R จะถูกเรียกว่า นิพจน์เรกูลาร์ ถ้า

- R เป็นสตริงว่าง (ϵ) หรือ R เป็นเซตว่าง (ϕ)
- R เป็นตัวอักษรที่เป็นสมาชิกของ Σ เช่น $\{a, b, c, d, e\}$
- R ประกอบด้วย r_1 และ r_2 โดยที่ r_1 และ r_2 เป็นนิพจน์เรกูลาร์ และ
 - $R = r_1 \cup r_2$ (เขียนแทนด้วย $r_1 + r_2$)
 - $R = r_1 \text{ Concatenate } r_2$ (เขียนแทนด้วย $r_1.r_2$)
 - $R = r_1^*$
 - $R = (r_1)$

ข้อสังเกตจากนิยามนี้คือ ความไม่เหมือนกันของ ϵ และ ϕ ใช้ ϵ แทนสตริงว่าง (Empty String) ส่วน ϕ ใช้แทนเซตของภาษาที่ไม่มีสมาชิกอยู่เลย

เนื่องจากเราสามารถใช้นิพจน์เรกูลาร์อธิบายภาษาได้ ดังนั้นภาษาสำหรับนิพจน์เรกูลาร์ขึ้นพื้นฐาน (ϵ , ϕ , ตัวอักษร) สามารถอธิบายได้ดังนี้

$$\text{ดังนั้น } L(\epsilon) = \{\epsilon\}$$

$$L(\phi) = \emptyset$$

$$L(a) = \{a\}$$

สำหรับนิพจน์เรกูลาร์ที่เกิดจากการผสมผสานกันระหว่างนิพจน์เรกูลาร์หลาย ๆ นิพจน์ ภาษาของนิพจน์เรกูลาร์สามารถอธิบายได้ดังนี้

กำหนดให้ r_1 และ r_2 เป็นนิพจน์เรกูลาร์

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2) = L(r_1)L(r_2) = L(r_1) \times L(r_2)$$

$$L(r_1^*) = (L(r_1))^*$$

$$L((r_1)) = L(r_1)$$

ตัวอย่างที่ 3.1 ภาษาที่นิพจน์เรกูลาร์ดังต่อไปนี้สามารถอธิบายได้คืออะไร (กำหนดให้ a และ b คือตัวอักษรของภาษา)

$$L(a + b), \quad L(a^* \cdot b), \quad L((a)), \quad L((ab)^*)$$

- ❖ $L(a + b) = L(a) \cup L(b)$
 $= \{a\} \cup \{b\} = \{a, b\}$
- ❖ $L(a^* \cdot b) = L(a)^* \times L(b)$ (ผลคูณคาร์ทีเซียนของ $L(a)^*$ และ $L(b)$)
 $= \{\varepsilon, a, aa, aaa, \dots\} \times \{b\} = \{b, ab, aab, aaab, \dots\}$
- ❖ $L((a)) = L(a) = \{a\}$
- ❖ $L((ab)^*) = \{\varepsilon, ab, abab, ababab, \dots\}$

ตัวอย่างที่ 3.2 ภาษาที่นิพจน์เรกูลาร์ดังต่อไปนี้สามารถอธิบายได้คืออะไร

$$\begin{aligned} r &= (a + b)^*(a + bb) \\ L(r) &= \{a, b\}^* \times \{a, bb\} \\ &= \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \dots\} \times \{a, bb\} \\ &= \{a, bb, aa, abb, ba, bbb, aaa, aabb, aba, abbb, \dots\} \end{aligned}$$

ตัวอย่างที่ 3.3 ภาษาที่นิพจน์เรกูลาร์ดังต่อไปนี้สามารถอธิบายได้คืออะไร

$$\begin{aligned} r &= (aa)^*(bb)^*b \\ L(r) &= \{aa\}^* \times \{bb\}^* \times \{b\} \\ &= \{\varepsilon, aa, aaaa, aaaaaa, \dots\} \times \{\varepsilon, bb, bbbb, bbbbb, \dots\} \times \{b\} \\ &= \{b, bbb, aab, aabb, bbbb, aaaab, aaaabbbb, \dots\} \\ L(r) &= \{a^{2n}b^{2m}b : n, m \geq 0\} \end{aligned}$$

ตัวอย่างที่ 3.4 ภาษาที่นิพจน์เรกูลาร์ดังต่อไปนี้สามารถอธิบายได้คืออะไร

$$\begin{aligned} r &= (0 + 1)^*00(0 + 1)^* \\ L(r) &= \{0, 1\}^* \cdot \{00\} \cdot \{0, 1\}^* \\ &= \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\} \times \{00\} \times \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\} \\ &= \{00, 000, 100, 001, 0000, 0100, 0001, \dots\} \\ L(r) &= \{w : w \in \{0, 1\}^* \text{ และ } w \text{ มี } 0 \text{ อยู่ติดกันอย่างน้อยหนึ่งครั้ง}\} \end{aligned}$$

ตัวอย่างที่ 3.5 ภาษาที่นิพจน์เรกูลาร์ดังต่อไปนี้สามารถอธิบายได้คืออะไร

$$\begin{aligned} r &= (1 + 01)^*(0 + \varepsilon) \\ L(r) &= \{1, 01\}^* \times \{0, \varepsilon\} \\ &= \{\varepsilon, 1, 01, 11, 101, 011, 0101, \dots\} \times \{0, \varepsilon\} \\ &= \{\varepsilon, 0, 1, 10, 01, 010, 11, 110, 101, 1010, 011, 0110, \dots\} \\ L(r) &= \{w : w \in \{0, 1\}^* \text{ และ } w \text{ ไม่มี } 0 \text{ อยู่ติดกันเลย}\} \end{aligned}$$

การอ่านนิพจน์เรกูลาร์เป็นความชำนาญที่จะต้องผ่านการฝึกฝนและการสังเกต ผู้อ่านที่ยังไม่เข้าใจตัวอย่างนี้ควรย้อนกลับมาอ่านตัวอย่างนี้อีกครั้งหลังจากอ่านบทนี้จบแล้ว เนื่องจากตัวอย่างนี้ถือว่าเป็นตัวอย่างที่ค่อนข้างยาก

- นิพจน์เรกูลาร์อธิบายภาษาเรกูลาร์

นิยาม 3.1.2 : ภาษาใด ๆ จะถูกเรียกว่า ภาษาเรกูลาร์ ถ้ามีนิพจน์เรกูลาร์ (Regular Expression) ที่สามารถอธิบายภาษานั้นได้

ในทางกลับกัน ถ้า r เป็นนิพจน์เรกูลาร์ จะต้องมีไฟล์ในต่อไปนี้มาตราที่ยอมรับ $L(r)$ ซึ่งจะส่งผลให้ $L(r)$ เป็นภาษาเรกูลาร์

พิสูจน์ : อาศัยวิธีการพิสูจน์เดียวกันกับ “NFA เท่าเทียมกับ DFA” ในบทที่ผ่านมา เราจะต้องพิสูจน์ว่า ภาษาที่ถูกสร้างโดยนิพจน์เรกูลาร์ = ภาษาเรกูลาร์

ดังนั้นการพิสูจน์จึงถูกแบ่งออกเป็น 2 กรณี ดัง

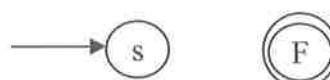
1. ภาษาที่ถูกสร้างโดยนิพจน์เรกูลาร์ \subseteq ภาษาเรกูลาร์
2. ภาษาที่ถูกสร้างโดยนิพจน์เรกูลาร์ \supseteq ภาษาเรกูลาร์

พิสูจน์กรณี 1. ภาษาที่ถูกสร้างโดยนิพจน์เรกูลาร์ \subseteq ภาษาเรกูลาร์

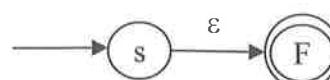
“สำหรับนิพจน์เรกูลาร์ r ใด ๆ ภาษา $L(r)$ เป็นภาษาเรกูลาร์” จะพิสูจน์โดยใช้อินดักชันกับความยาวของนิพจน์เรกูลาร์ r

กรณีพื้นฐาน (Basis Case) : สำหรับนิพจน์เรกูลาร์พื้นฐานคือ ϕ , ϵ , ตัวอักษร

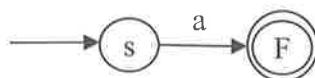
- (1) เมื่อ $r = \phi$, $L(r) = \phi$ สร้าง NFA ได้ดังนี้



- (2) เมื่อ $r = \epsilon$, $L(r) = \{\epsilon\}$ สร้าง NFA ได้ดังนี้

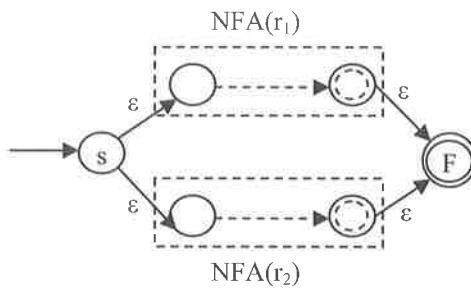
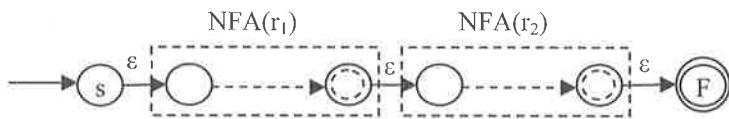
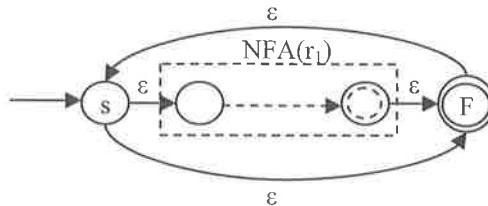


- (3) เมื่อ $r = a$, $L(r) = \{a\}$ สร้าง NFA ได้ดังนี้



เนื่องจากสามารถสร้าง NFA ให้กับนิพจน์เรกูลาร์พื้นฐานได้ทุกรูปแบบ ดังนั้นการพิสูจน์กรณีพื้นฐานจึงถูกต้อง

กรณีเชิงพาณิชย์ (Inductive Case) : สมมุติว่า r_1 และ r_2 เป็นนิพจน์เรกูลาร์ และ $L(r_1)$ และ $L(r_2)$ เป็นภาษาเรกูลาร์ จะพิสูจน์โดยการสร้าง NFA ให้กับกรณีที่นิพจน์เรกูลาร์มีความซับซ้อนขึ้นดังต่อไปนี้

(1) $r = r_1 + r_2$ (2) $r = r_1 \cdot r_2$ (3) $r = (r_1)^*$ (4) $r = ((r_1))$

เนื่องจาก NFA ของ r_1 มีความเหมือนกันกับ NFA ของ $((r_1))$ ดังนี้จึงไม่มีความจำเป็นต้องพิสูจน์ในกรณีนี้

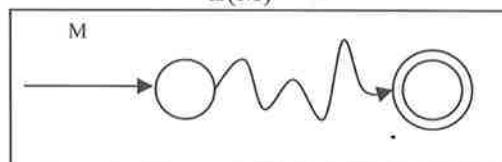
จะเห็นได้ว่าสามารถสร้าง NFA รองรับนิพจน์เรกุลาร์ที่มีความซับซ้อนขึ้นในทุก ๆ กรณีได้ ดังนั้นการพิสูจน์กรณีเฉพาะของการพิสูจน์โดยอินดักชันนี้ได้รับการพิสูจน์แล้ว

พิสูจน์กรณี 2. ภาษาที่ถูกสร้างโดยนิพจน์เรกุลาร์ \subseteq ภาษาเรกุลาร์

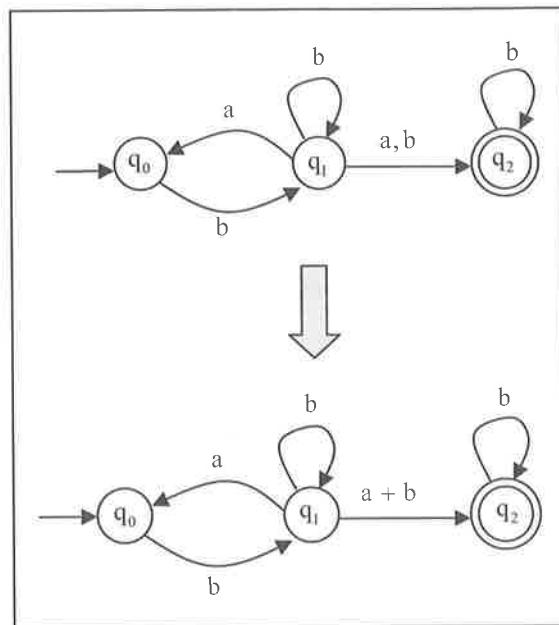
จะพิสูจน์โดยวิธีการสร้างนิพจน์เรกุลาร์จากภาษาเรกุลาร์ (Construction Proof)

กำหนดให้ L เป็นภาษาเรกุลาร์ซึ่งจะต้องมี NFA M รองรับ นั่นคือ

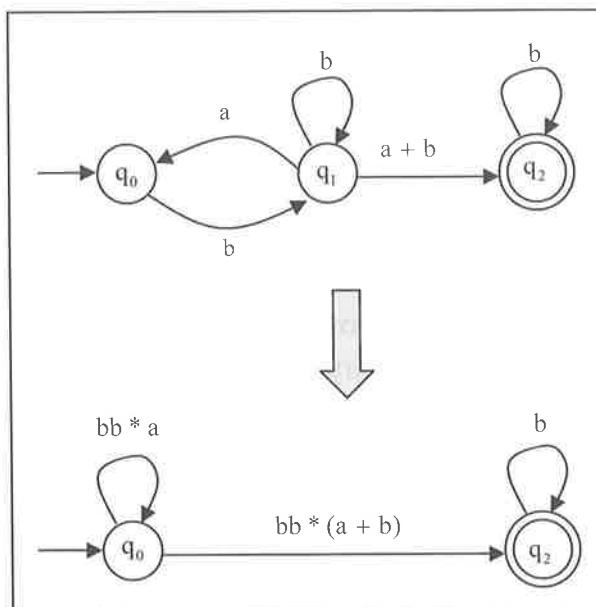
$$L(M) = L$$



จาก NFA M สร้างทรานซิชันกราฟ โดยใช้เทคนิคการรบกวนแต่ละtranซิชันของกราฟ เช่น



จากนั้น ลดรูปสถานะของ NFA M โดยการยุบรวมนิพจน์เรกูลาร์เข้าด้วยกัน (รายละเอียดในการลดรูปสถานะของไฟโนต์อโตมาตา ผู้อ่านสามารถศึกษาเพิ่มเติมได้ในหัวข้อต่อไป) เช่น



ผลที่ได้จะเหลือสถานะภายใน NFA M เพียง 2 สถานะ โดยแต่ละสถานะชั้นภายใน NFA จะถูกกำกับด้วยนิพจน์เรกูลาร์ที่ถูกยุบรวมกัน

จาก NFA ที่ถูกยุบสถานะแล้ว สามารถหานิพจน์เรกูลาร์ที่รับ NFA M ได้ดังนี้

$$r = (bb^*a)^* bb^* (a + b)b^*$$

ดังนั้นจึงสรุปได้ว่า $L(r) = L(M) = L$

ตัวอย่างที่ 3.6 ภาษาที่นิพจน์เรกูลาร์ดังต่อไปนี้อธิบายคืออะไร

$$r = (a + b \cdot c)^*$$

$$L(r) = L(a + b.c)^*$$

$$\text{เมื่อจาก } L(a + b.c)^0 = \{\epsilon\}$$

$$L(a + b.c)^1 = \{a, bc\}$$

$$L(a + b.c)^2 = \{aa, abc, bca, bcba\}$$

$$\text{ดังนั้น } L(r) = L(a + b.c)^0 + L(a + b.c)^1 + L(a + b.c)^2 + \dots$$

$$L(r) = \{\epsilon, a, bc, aa, abc, bca, bcba, \dots\}$$

ตัวอย่างที่ 3.7 ภาษาที่นิพจน์เรกูลาร์ดังต่อไปนี้อธิบายคืออะไร

$$r = (a + b) \cdot a^*$$

$$L((a + b) \cdot a^*) = L(a + b)L(a^*)$$

$$= L(a + b)L(a^*)$$

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

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

$$= \{a, b\} \{\epsilon, a, aa, aaa, \dots\}$$

$$= \{a, aa, aaa, \dots, b, ba, baa, \dots\}$$

ตัวอย่างที่ 3.8 จงพิสูจน์ให้เห็นว่าภาษา L เป็นภาษาเรกูลาร์

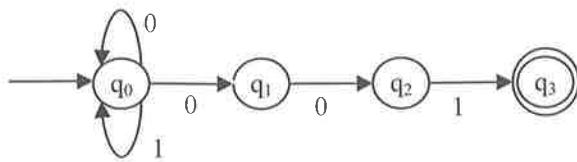
$$L = \{w \mid w \in \{0, 1\}^*\text{ โดยที่ } w \text{ ลงท้ายด้วย } 001 \text{ เสมอ}\}$$

การพิสูจน์ว่าภาษาใด ๆ เป็นภาษาเรกูลาร์ จะต้องสร้างนิพจน์เรกูลาร์หรือสร้างไฟล์โน็ตอโน้ตมาตา (DFA หรือ NFA) ที่รับภาษานั้น ในที่นี้จะแสดงให้ดูทั้งสองวิธี

เนื่องจากสนใจเฉพาะสตริงที่ลงท้ายด้วย 001 โดยที่สตริงส่วนหน้า (Prefix) จะเป็นสตริงอะไรก็ได้ที่ประกอบด้วย 0 หรือ 1 ดังนั้นจึงแทนภาษา L ด้วยนิพจน์เรกูลาร์และ NFA ดังต่อไปนี้

นิพจน์เรกูลาร์สำหรับภาษา L คือ $(0 + 1)^*001$

ไฟล์โน็ตอโน้ตมาตา NFA สำหรับภาษา L คือ

รูปที่ 3.2 NFA สำหรับภาษา L

ตั้งนั้นจึงสรุปได้ว่า ภาษา L เป็นภาษาเรกูลาร์

ตัวอย่างที่ 3.9 จงสร้างนิพจน์เรกูลาร์ที่ใช้อัญญาติประกอบด้วยตัวอักษร a และ b โดยทั้งจำนวนของ a และ b สามารถหารด้วย 3 ลงตัว นอกจากนี้หากสตริงในภาษามี a ปรากฏอยู่ ตัวอักษร a จะต้องนำหน้าตัวอักษร b เสมอ ยกตัวอย่างเช่น $aaa, bbb, aaabb, aaabbbbb, aaaaaabbbb$ เป็นต้น

$$r = (aaa)^*(bbb)^*$$

ตัวอย่างที่ 3.10 จงสร้างนิพจน์เรกูลาร์สำหรับภาษาที่ประกอบด้วยตัวอักษร 0 และ 1 โดยจะต้องมี 01 เป็นสตริงบ่อยเสมอ

$$r = (0 + 1)^* 01 (0 + 1)^*$$

ตัวอย่างของสตริงที่ได้จากนิพจน์เรกูลาร์ r คือ $\{01, 001, 101, 010, 0101, \dots\}$

ตัวอย่างที่ 3.11 กำหนดให้ภาษา L คือ

$$L = \{w \in \{0, 1\}^* : |w| \bmod 5 = 0\}$$

จงหา_nิพจน์เรกูลาร์สำหรับภาษา L

$$r = ((0 + 1)(0 + 1)(0 + 1)(0 + 1)(0 + 1))^*$$

ความเท่าเทียมกันของนิพจน์เรกูลาร์

นิยาม 3.1.3 : นิพจน์เรกูลาร์ 2 นิพจน์ จะเท่าเทียมกันก็ต่อเมื่อภาษาของนิพจน์เรกูลาร์ของทั้งสองนิพจน์ มีความเท่าเทียมกัน นั่นคือ

หาก r_1 และ r_2 เป็นนิพจน์เรกูลาร์

$$r_1 = r_2 \text{ ก็ต่อเมื่อ } L(r_1) = L(r_2)$$

ตัวอย่างที่ 3.12 จงให้เหตุผลว่านิพจน์เรกูลาร์ r_1 และ r_2 มีความเท่าเทียมกัน

$$r_1 = (1 + 01)^* (0 + \varepsilon)$$

$$r_2 = (1 * 011^*)^* (0 + \varepsilon) + 1^* (0 + \varepsilon)$$

สำหรับตัวอย่างนี้ คงเป็นไปไม่ได้เลยที่จะเปรียบเทียบสมาชิกทุกตัวของ $L(r_1)$ และ $L(r_2)$ ทีละตัว เนื่องจากเซตของภาษาทั้งสองมีจำนวนสมาชิกไม่จำกัด วิธีการที่เหมาะสมคือการสังเกตรูปแบบของสตริงที่ถูกสร้างขึ้นมาจากการนิพจน์เรกูลาร์ทั้งสองว่ามีลักษณะเหมือนกันหรือไม่

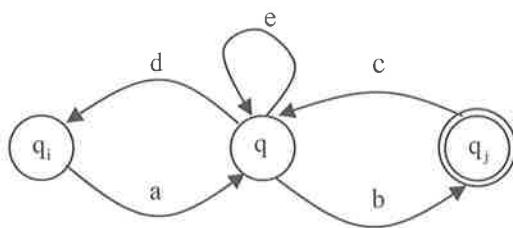
จะสังเกตได้ว่าทั้งนิพจน์เรกุลาร์ r_1 และ r_2 สามารถสร้างสตริงที่ประกอบด้วย 0 และ 1 ซึ่งมีรูปแบบที่หลากหลายมาก สิ่งที่เหมือนกันระหว่างนิพจน์เรกุลาร์ทั้งสองคือ สตริงที่ถูกสร้างออกมามีเมล็ด 0 อยู่ติดกันเลย ดังนั้นจึงสามารถสรุปได้ว่านิพจน์เรกุลาร์ทั้งสองมีความเท่าเทียมกันเนื่องจาก $L(r_1) = L(r_2) = \{w \in \{0,1\}^* \mid w \text{ ไม่มีเลข } 0 \text{ อยู่ติดกันเลย}\}$

3.2 เทคนิคการลดรูปสถาบันไฟฟ้าโนตอมาตา

ในบางครั้งเราอาจต้องเพชญ์กับไฟฟ้าโนตอมาตาที่ยาวและซับซ้อน ทำให้การทำความเข้าใจกับอโตมาตา ดังกล่าวทำได้ยาก สำหรับหัวข้อนี้จะเป็นการนำเสนอวิธีการสร้างไฟฟ้าโนตอมาตาที่สามารถอธิบายภาษาได้เหมือนกับไฟฟ้าโนตอมาตาที่ซับซ้อน แต่สามารถอ่านเข้าใจได้ง่ายกว่า เราเรียกการลดรูปในลักษณะนี้ว่า “ไฟฟ้าโนตอมาตาที่เรียบง่าย” (Simplified Finite Automata)

หลักการลดรูปไฟฟ้าโนตอมาตา

- สำหรับไฟฟ้าโนตอมาตาใด ๆ เช่น

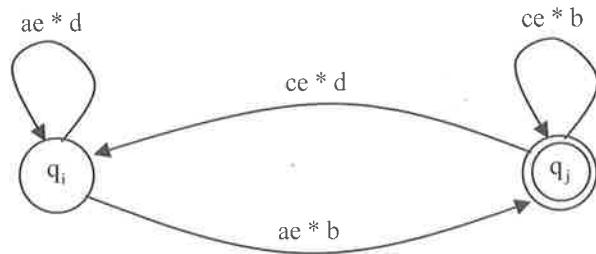


รูปที่ 3.3 ไฟฟ้าโนตอมาตาใด ๆ ที่จะลดรูป

- พยายามกำหนดสถานะที่ไม่ใช่สถานะเริ่มต้นและสถานะสุดท้าย ในที่นี้คือสถานะ q

- สร้างนิพจน์เรกุลาร์ที่ต้องใช้ในการเดินจากสถานะเริ่มต้น q_i ไปยังสถานะสุดท้าย q_j และในทางกลับกัน จากสถานะ q_j กลับไปยังสถานะ q_i จากนั้นเชื่อมสถานะ q_i และ q_j เข้าด้วยกันโดยกำกับด้วยนิพจน์เรกุลาร์ที่ได้ เช่น ใช้ ae^*b ในการเดินจาก q_i ไปยัง q_j และ ce^*d ในการเดินจาก q_j ไปยัง q_i

นอกจากนี้เรายังต้องหา尼พจน์เรกุลาร์สำหรับเส้นทางอื่น ๆ ที่ต้องเดินผ่านสถานะ q อีกด้วย เช่น เราใช้ ae^*d ในการเดินจาก q_i ผ่าน q แล้วกลับมาอีก q_i และใช้ ce^*b ในการเดินจาก q_j ผ่าน q แล้วกลับมาอีก q_j ดังแสดงในรูป

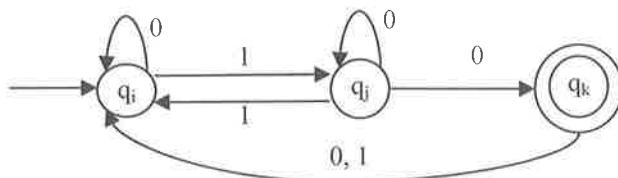


รูปที่ 3.4 ไฟไนต์อัตโนมາตาที่ลดรูปแล้ว

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

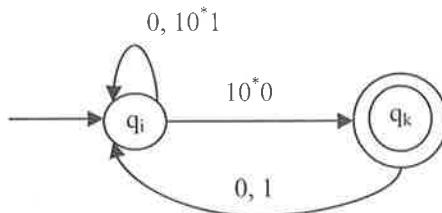
ข้อควรระวังในการลดรูปสถานะ q_i ดีอ จะต้องหานิพจน์เรกุลาร์ที่เป็นไปได้ทั้งหมดที่มีเส้นทางการเดินทางผ่านสถานะ q_i มิฉะนั้นไฟไนต์อัตโนมາตาที่ลดรูปสถานะจะไม่ครอบคลุมภาษาที่อัตโนมາตาเดิมรองรับ

ตัวอย่างที่ 3.13 จงลดรูป NFA ดังต่อไปนี้ พร้อมทั้งเขียนนิพจน์เรกุลาร์สำหรับภาษาที่ NFA รองรับ



รูปที่ 3.5 NFA ที่ยังไม่ได้ลดรูป

สามารถลดรูป NFA ดังกล่าวได้โดยการยุบสถานะ q_j เพื่อไปรวมกับสถานะ q_i ดังนี้



รูปที่ 3.6 NFA ที่ลดรูปแล้ว

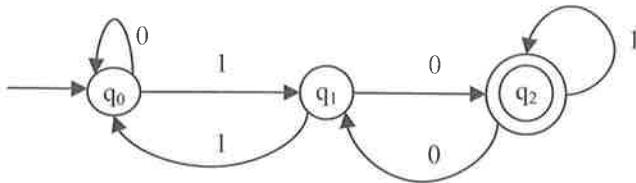
นิพจน์เรกุลาร์สำหรับ NFA ที่ลดรูปแล้วดีอ

$$(0 + 10^*1)^*10^*0((0 + 1)(0 + 10^*1)^*10^*0)^*$$

ตัวอย่างที่ 3.14 จงหานิพจน์เรกุลาร์สำหรับภาษาดังต่อไปนี้

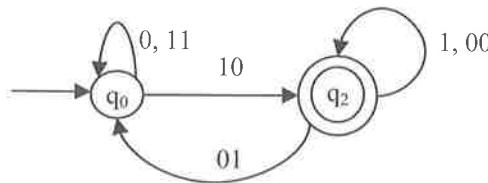
$$L = \{w \in \{0, 1\}^* : w \text{ เป็นเลขฐานสอง และ } w \bmod 3 = 2\}$$

สำหรับปัญหานี้สามารถแก้ได้โดยใช้เทคนิคการแทนซึ่งสถานะด้วยเศษที่ได้จากการหารด้วย 3 ดังนั้นจะมีสถานะทั้งหมดอยู่ 3 สถานะ คือ q_0, q_1, q_2 ซึ่งจะใช้แทนการหารที่มีเศษเป็น 0, 1, 2 ตามลำดับ



รูปที่ 3.7 ไฟน์ต่อโถมตามาสำหรับภาษา L

จากรูป สามารถลดรูปไฟน์ต่อโถมมาได้ดังนี้



รูปที่ 3.8 ไฟน์ต่อโถมตามาสำหรับภาษา L ที่ลดรูปแล้ว

ดังนั้นนิพจน์เรกูลาร์สามารถเขียนได้ดังนี้

$$(0+11)^*10((1+00)^* + 01(0+11)^*10)^*$$

3.3 ไวยากรณ์เรกูลาร์

ภาษาทุกภาษาจะต้องมีไวยากรณ์รองรับเสมอ ไวยากรณ์ทำหน้าที่กำหนดกฎเกณฑ์ในการสร้างภาษา ดังนั้นบทบาทของไวยากรณ์จึงมีความสำคัญไม่น้อยไปกว่าภาษา ตัวอย่างของไวยากรณ์สำหรับภาษาอังกฤษสามารถอธิบายได้ดังนี้

- $\langle \text{sentence} \rangle \rightarrow \langle \text{noun_phrase} \rangle \langle \text{predicate} \rangle$
- $\langle \text{noun_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle$
- $\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle$
- $\langle \text{article} \rangle \rightarrow \text{a}$
- $\langle \text{article} \rangle \rightarrow \text{the}$
- $\langle \text{noun} \rangle \rightarrow \text{cat}$
- $\langle \text{noun} \rangle \rightarrow \text{dog}$
- $\langle \text{verb} \rangle \rightarrow \text{runs}$
- $\langle \text{verb} \rangle \rightarrow \text{walks}$

เรียกสัญลักษณ์ที่ล้อมรอบด้วยเครื่องหมาย “⟨⟩” ว่า ตัวแปร (Variable) เช่น ⟨sentence⟩ และเรียกสัญลักษณ์ที่ไม่ได้ล้อมรอบด้วยเครื่องหมายดังกล่าวว่า เทอร์มินอล (Terminal) เช่น a, the, cat จากตัวอย่าง ไวยากรณ์ของภาษาอังกฤษนี้ เราสามารถนำไปสร้างประโยค (Sentence) หรือสตริงซึ่งเป็นสมาชิกในเซตของภาษา อังกฤษได้ดังนี้

$$\begin{aligned}\langle \text{sentence} \rangle &\Rightarrow \langle \text{noun_phrase} \rangle \ \langle \text{predicate} \rangle \\&\Rightarrow \langle \text{noun_phrase} \rangle \ \langle \text{verb} \rangle \\&\Rightarrow \langle \text{article} \rangle \ \langle \text{noun} \rangle \ \langle \text{verb} \rangle \\&\Rightarrow \text{the} \ \langle \text{noun} \rangle \ \langle \text{verb} \rangle \\&\Rightarrow \text{the dog} \ \langle \text{verb} \rangle \\&\Rightarrow \text{the dog walks}\end{aligned}$$

สำหรับภาษาเรกูลาร์ (Regular Language) เรา มีนิพจน์เรกูลาร์ (Regular Expression) และไฟฟอนต์ öd โตามาตา เป็นเครื่องมือในการอธิบายภาษา ดังนี้นี่ถึงกล่าวไว้ว่าหากภาษาใดสามารถอธิบายได้ด้วยนิพจน์เรกูลาร์หรือไฟฟอนต์อโตามาตาแล้ว ภาษานั้นถือว่า เป็นภาษาเรกูลาร์

นอกเหนือจากนิพจน์เรกูลาร์และไฟฟอนต์อโตามาตาแล้ว เรา ยังสามารถอธิบายภาษาเรกูลาร์ได้อีกวิธีหนึ่ง คือ ไวยากรณ์เรกูลาร์ (Regular Grammar) ซึ่งมีนิยามดังนี้

นิยาม 3.3.1 : กำหนดให้ไวยากรณ์ $G = (V, T, S, P)$ ไวยากรณ์ G จะเป็นเรกูลาร์เมื่อ G อยู่ในรูป อย่างใดอย่างหนึ่งดังต่อไปนี้

(1) $V_1 \longrightarrow t_1 V_2, \quad (\text{ไวยากรณ์แบบ Right-linear})$

$V_1 \longrightarrow t_2$

หรือ

(2) $V_1 \longrightarrow V_2 t_1, \quad (\text{ไวยากรณ์แบบ Left-linear})$

$V_1 \longrightarrow t_2$

โดยที่ $V_1, V_2 \in V$ และ $t_1, t_2 \in T^*$

หมายเหตุ :

1. ไวยากรณ์แบบ Right-linear คือ ไวยากรณ์ที่ความมื้อของกฎมีตัวแปร (Variable) เพียงหนึ่งตัว และมีตำแหน่งอยู่ขวามือสุดของกฎ

2. ไวยากรณ์แบบ Left-linear คือ ไวยากรณ์ที่ความมื้อของกฎมีตัวแปรเพียงหนึ่งตัว และมีตำแหน่งอยู่ข้ายมื้อสุดหลังถูกคร

นั่นคือ ไวยากรณ์ G จะเป็นเรกูลาร์ก็ต่อเมื่อมีรูปแบบดังนี้ หรือ 1 หรือ 2 อย่างใดอย่างหนึ่งเท่านั้น หากมีรูปแบบทั้งสองรูปแบบในไวยากรณ์เดียวกัน (มีทั้ง Right-linear และ Left-linear) จะไม่ถือว่าเป็นไวยากรณ์เรกูลาร์

ตัวอย่างของไวยากรณ์เรกูลาร์สามารถแสดงได้ดังนี้

$G = (V, T, S, P)$ โดยที่

$V = \{V_1, V_2\}$

$$T = \{a, b\}$$

$$S = \{V_1\}$$

$$P : V_1 \longrightarrow aV_1 \mid abV_2$$

$$V_2 \longrightarrow a \mid b$$

ไวยากรณ์นี้ถือว่าเป็นเรกูลาร์เนื่องจากมีรูปแบบเป็น Right-linear

ตัวอย่างที่ 3.15 จงหาภาษาเรกูลาร์สำหรับไวยากรณ์ดังต่อไปนี้

$$G = (V, T, S, P) \text{ โดยที่}$$

$$V = \{S\}$$

$$T = \{a, b\}$$

$$S = \{S\}$$

$$P : S \longrightarrow abS$$

$$S \longrightarrow a$$

ไวยากรณ์ G เป็นไวยากรณ์เรกูลาร์เนื่องจากมีรูปแบบเป็น Right-linear เราสามารถสร้างสตริงจากไวยากรณ์ G ได้หลายสตริง เช่น aba, abababa

$$S \rightarrow abS \rightarrow aba$$

$$S \rightarrow abS \rightarrow ababS \rightarrow abababS \rightarrow abababa$$

ไม่ว่าสตริงที่ถูกสร้างออกมานั้นมาจากไวยากรณ์ G จะยาวเพียงใด สตริงนั้นจะต้องลงท้ายด้วยตัวอักษร a เสมอ นอกจากนี้หากมีสตริงส่วนหน้า (Prefix) สตริงส่วนนั้นจะต้องประกอบด้วยคู่ของตัวอักษร ab กี่คู่ก็ได้ จากข้อสังเกตดังกล่าว เราจึงสรุปภาษาของไวยากรณ์เรกูลาร์ G ได้ดังนี้

$$L(G) = (ab)^*a$$

อนึ่ง จากการได้มาของสตริง $S \rightarrow abS \rightarrow ababS \rightarrow abababS \rightarrow abababa$ เรียก abS, ababS, abababS ว่ารูปแบบเชื่นเทินเชียล (Sentential Form) และเรียก abababa ว่าประโยค (Sentence) นอกจากนี้ เพื่อเป็นการกระชับและสะดวกในการเข้าใจ เรายังสามารถแทนกระบวนการได้มาของสตริง $S \rightarrow abS \rightarrow ababS \rightarrow abababS \rightarrow abababa$ ด้วย $S \xrightarrow{*} abababa$

ตัวอย่างที่ 3.16 จงสร้างไวยากรณ์เรกูลาร์สำหรับภาษา L

$$L = aab(ab)^*$$

การสร้างไวยากรณ์สำหรับภาษา L นี้ จะต้องแบ่งออกเป็น 3 ส่วน ส่วนแรกทำหน้าที่สร้างสตริง a ตัวหน้าส่วนที่ 2 ทำหน้าที่สร้างสตริง ab และส่วนสุดท้ายทำหน้าที่ทำซ้ำสตริง ab ในส่วนหลัง ด้วยหลักการดังกล่าว เราสามารถสร้างไวยากรณ์เรกูลาร์สำหรับภาษา L ได้ดังนี้

$$G = (V, T, S, P) \text{ โดยที่}$$

$$V = \{S, A, B\}$$

$$T = \{a, b\}$$

$$S = \{S\}$$

$$\begin{aligned} S &\rightarrow Aab \\ P : \quad A &\rightarrow Aab \mid B \\ B &\rightarrow a \end{aligned}$$

ภาษาของไวยากรณ์

นิยาม 3.3.2 : สำหรับไวยากรณ์ $G = (V, T, S, P)$ ภาษาของไวยากรณ์ G ถูกนิยามโดย

$$L(G) = \{w : S \xrightarrow{*} w\}$$

สตริง w เกิดจาก Σ^* โดยมีเงื่อนไขว่าจะต้องมีกระบวนการได้มา (Derivation) จากลักษณะเริ่มต้น S หมายงสตริง w ได้

ตัวอย่างที่ 3.17 สำหรับไวยากรณ์ $G = (\{S, A\}, \{a, b\}, S, P)$

$$\begin{aligned} S &\rightarrow Ab \\ P : A &\rightarrow aAb \\ A &\rightarrow \epsilon \end{aligned}$$

จงหาภาษาของไวยากรณ์ G

ไวยากรณ์ G สามารถนำมำทำกระบวนการได้มาซึ่งสตริงดังนี้

$$\begin{aligned} S &\rightarrow Ab \rightarrow b \\ S &\rightarrow Ab \rightarrow aAbb \rightarrow abb \\ S &\rightarrow Ab \rightarrow aAbb \rightarrow aaAbbb \rightarrow aabb \\ S &\rightarrow Ab \rightarrow aAbb \rightarrow aaAbbb \rightarrow aaaAbbbb \rightarrow aaabbbb \end{aligned}$$

จากรูปแบบของสตริงที่ถูกสร้างขึ้นมาจากการไวยากรณ์ G เราสามารถสรุปได้ว่าภาษาของไวยากรณ์ G คือ

$$L(G) = \{a^n b^n : n \geq 0\}$$

เพื่อเป็นการอำนวยความสะดวกในการเขียนไวยากรณ์ G สามารถแทน

$$\begin{aligned} A &\rightarrow aAb \\ A &\rightarrow \epsilon \\ \text{ด้วยกฎที่ลื้นลงดังนี้} \\ A &\rightarrow aAb \mid \epsilon \end{aligned}$$

โดยที่เครื่องหมาย ‘ | ’ หมายถึง การเลือกอย่างใดอย่างหนึ่ง

3.4 ความสัมพันธ์ระหว่างภาษาเรกูลาร์และไวยากรณ์เรกูลาร์

ทฤษฎี 3.4.1 : ภาษาที่ถูกสร้างขึ้นจากไวยากรณ์เรกูลาร์ G เป็นภาษาเรกูลาร์

พิสูจน์ : ต้องการพิสูจน์ว่าถ้า G เป็นไวยากรณ์เรกูลาร์แล้ว $L(G) = \text{ภาษาเรกูลาร์}$ การพิสูจน์จะแบ่งออกเป็น 2 ส่วน คือ

1. $L(G) \subseteq$ ภาษาเรกูลาร์ “ไวยากรณ์เรกูลาร์ G ได ๆ สามารถสร้างภาษาเรกูลาร์ได้”

2. $L(G) \supseteq$ ภาษาเรกูลาร์ “ภาษาเรกูลาร์ได ๆ ถูกสร้างขึ้นจากไวยากรณ์เรกูลาร์”

พิสูจน์กรณี 1. $L(G) \subseteq$ ภาษาเรกูลาร์

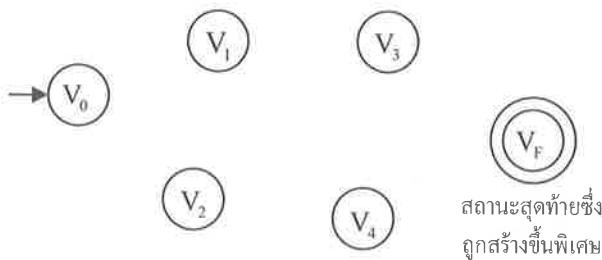
(1) ถ้า G เป็นไวยากรณ์แบบ Right-linear เราก็สามารถโดยการสร้าง NFA M ซึ่งรองรับภาษา $L(G)$ นั้น คือ $L(M) = L(G)$

สำหรับไวยากรณ์ G ซึ่งมีรูปแบบเป็น

$$V_i \rightarrow a_1 a_2 \cdots a_m V_j$$

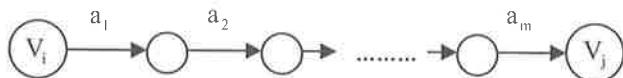
หรือ $V_i \rightarrow a_1 a_2 \cdots a_m$

เราสร้าง NFA M โดยนำตัวแปรทั้งหมดที่อยู่ในไวยากรณ์มาสร้างเป็นโหนดหรือสถานะ สำหรับสถานะสุดท้าย (Final State) ให้สร้างขึ้นมาพิเศษโดยไม่เกี่ยวข้องกับตัวแปรภายในไวยากรณ์ เช่น

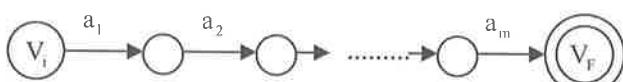


รูปที่ 3.9 สถานะภายใน NFA M

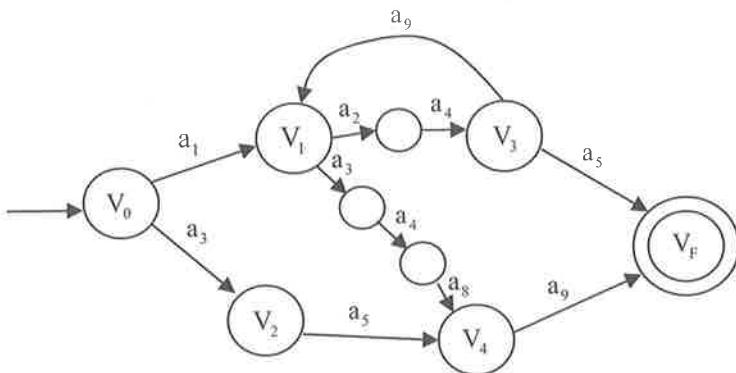
สำหรับกฎที่มีรูปแบบ $V_i \rightarrow a_1 a_2 \cdots a_m V_j$ สร้างทรานซิชันพร้อมทั้งสถานะที่อยู่ระหว่างสถานะ V_i และ V_j ดังนี้



สำหรับกฎที่มีรูปแบบ $V_i \rightarrow a_1 a_2 \cdots a_m$ สร้างทรานซิชันพร้อมทั้งสถานะที่อยู่ระหว่างสถานะ V_i และ V_F ดังนี้



ผลสุดท้ายเราจะได้ NFA M ซึ่งมีลักษณะดังนี้



รูปที่ 3.10 NFA M

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

จาก NFA M ที่เราสร้างได้ เป็นผลให้ $L(G) = L(M)$

(2) ถ้า G เป็นไวยากรณ์แบบ Left-linear

สำหรับไวยากรณ์ G ซึ่งมีรูปแบบดังนี้

$$A \rightarrow B a_1 a_2 \cdots a_k$$

$$\text{หรือ } A \rightarrow a_1 a_2 \cdots a_k$$

หากกระทำโดยเปลี่ยนผันกัน (Reverse) กับไวยากรณ์ G จะได้ไวยากรณ์แบบ Right-linear G_R ซึ่งนี้

$$A \rightarrow B a_1 a_2 \cdots a_k \iff A \rightarrow a_k \cdots a_2 a_1 B$$

$$A \rightarrow a_1 a_2 \cdots a_k \iff A \rightarrow a_k \cdots a_2 a_1$$

เนื่องจากพิสูจน์มานอกกรณีแล้วว่าภาษาที่ถูกสร้างขึ้นมาจากไวยากรณ์ Right-linear เป็นภาษาเรกุลาร์ ดังนั้น $L(G_R)$ จึงเป็นภาษาเรกุลาร์ นอกจากนี้ไฟในต่อไปนี้จะแสดงถึงความคล้ายคลึงกับไฟในตัวอย่างภาษา $L(G_R)$ แต่แตกต่างกันที่ลูกศรของทราบชิ้นจะกลับทิศทางกันและมีการสลับกันระหว่างสถานะเริ่มต้นและสถานะสุดท้าย ดังนั้นภาษาของไวยากรณ์ Left-linear G จึงเป็นภาษาเรกุลาร์

พิสูจน์ครั้น 2. $L(G) \supseteq \text{ภาษาเรกุลาร์}$

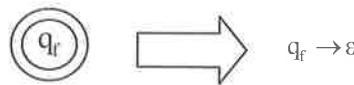
พิสูจน์ดังนี้ กำหนดให้ M เป็น NFA ซึ่งรองรับภาษาเรกุลาร์ L จากนั้นสร้างไวยากรณ์เรกุลาร์ G จาก NFA M ซึ่งทำให้ $L(G) = L(M)$

สำหรับทราบชิ้นใด ๆ เราสร้างกฎซึ่งมีลักษณะดังนี้



โดยที่ q และ p คือ ตัวแปร (Variable) และ a คือ สัญลักษณ์เทอร์มินอล (Terminal)

สำหรับสถานะสุดท้าย q_f เราสร้างกฎเพิ่มเติมดังนี้



เนื่องจากไวยากรณ์ G มีรูปแบบเป็น Right-linear ดังนั้น G เป็นไวยากรณ์เรกูลาร์ซึ่งทำให้

$$L(G) = L(M) = L$$

ตัวอย่างที่ 3.18 จงพิสูจน์ว่าไวยากรณ์ G ดังต่อไปนี้สร้างภาษาอອกมาเป็นภาษาเรกูลาร์

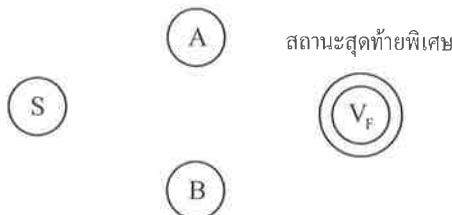
$$S \rightarrow aA \mid B,$$

$$A \rightarrow aaB,$$

$$B \rightarrow bB \mid a$$

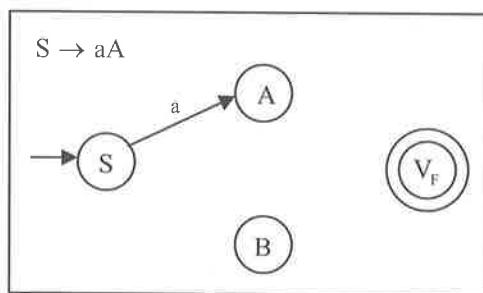
เนื่องจากไวยากรณ์นี้เป็นแบบ Right-linear เราจะพิสูจน์โดยการสร้างไฟโนต์อโตมาตา NFA สำหรับไวยากรณ์นี้

ขั้นตอนที่ 1 : นำตัวแปรที่ปรากฏในไวยากรณ์มาสร้างสถานะ โดยเพิ่มสถานะสุดท้ายพิเศษขึ้นมาอีกหนึ่งสถานะ ดังนี้



รูปที่ 3.11 สถานะภายใน NFA

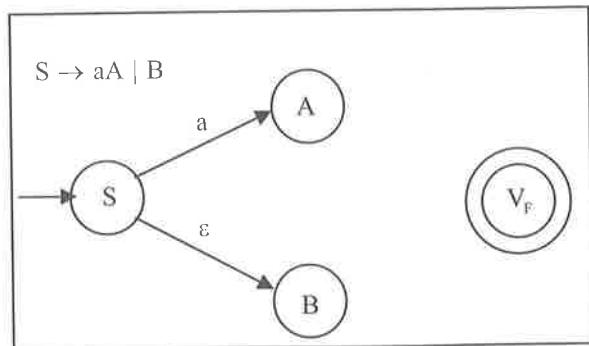
ขั้นตอนที่ 2 : สำหรับแต่ละโปรดักชันของไวยากรณ์ เพิ่มทรานซิชันสำหรับโปรดักชันนั้นลงไว้ใน NFA



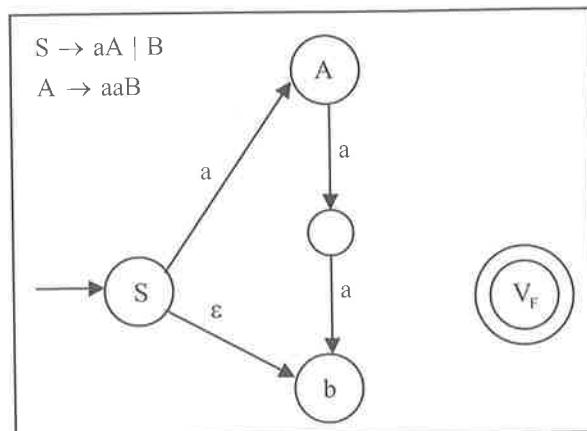
(ก)

รูปที่ 3.12 NFA ที่ได้มาจากการ G

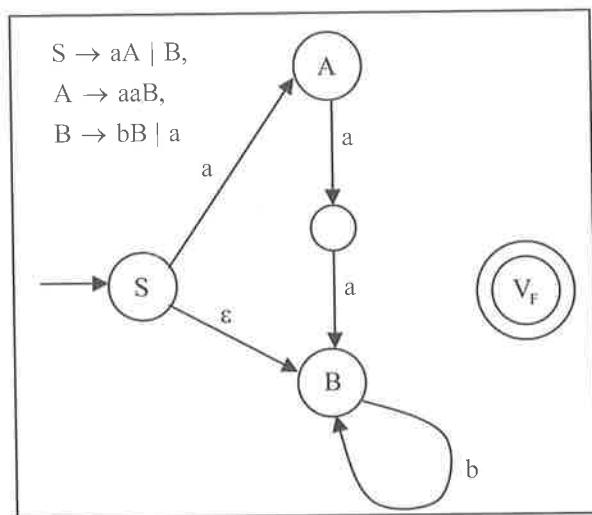
(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)



(ঃ)

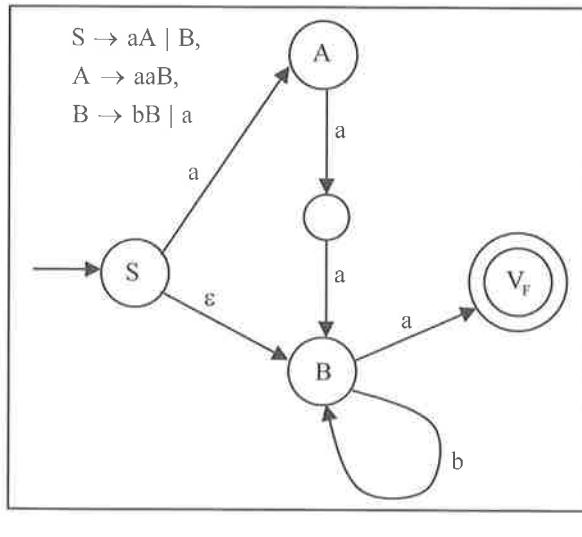


(ং)



(ঃ)

รูปที่ 3.12 (ต่อ)



(จ)

รูปที่ 3.12 (ต่อ)

ไฟในต่ออโตมาตา NFA รูปนี้รองรับภาษาซึ่งถูกสร้างขึ้นมาจากไวยากรณ์ ซึ่งก็คือ

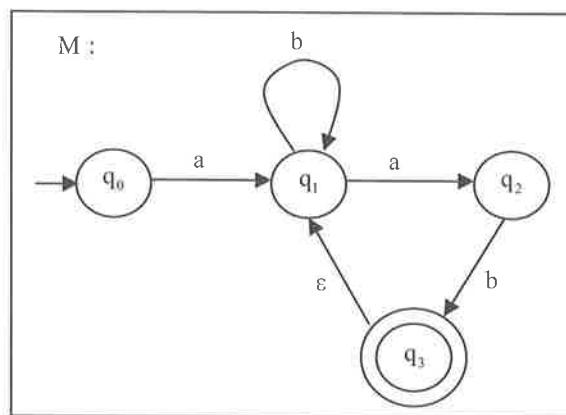
$$L(G) = \{aaab^n a, b^n a : n \geq 0\}$$

ดังนั้นภาษา $L(G)$ จึงเป็นภาษาเรกูลาร์

ตัวอย่างที่ 3.19 จงหาไวยากรณ์ที่รองรับภาษาเรกูลาร์ ดังต่อไปนี้

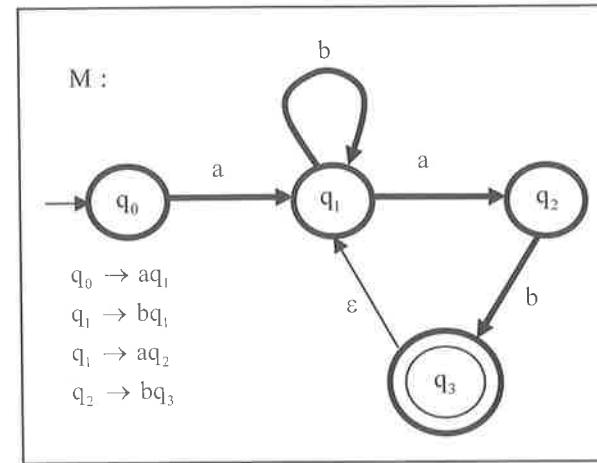
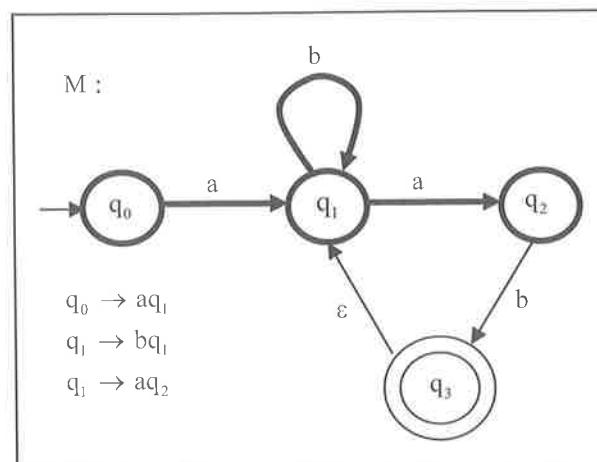
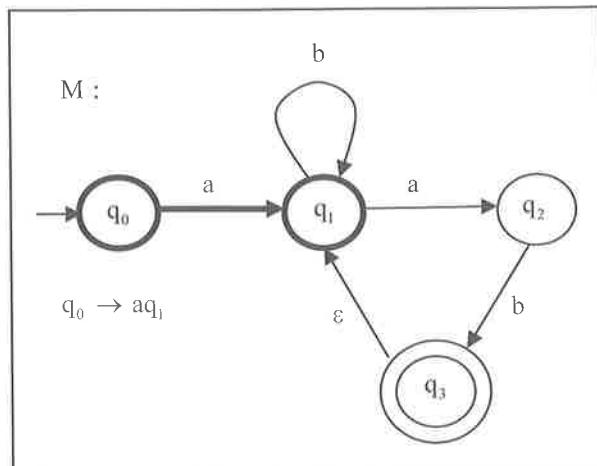
$$L = ab * ab(b * ab) *$$

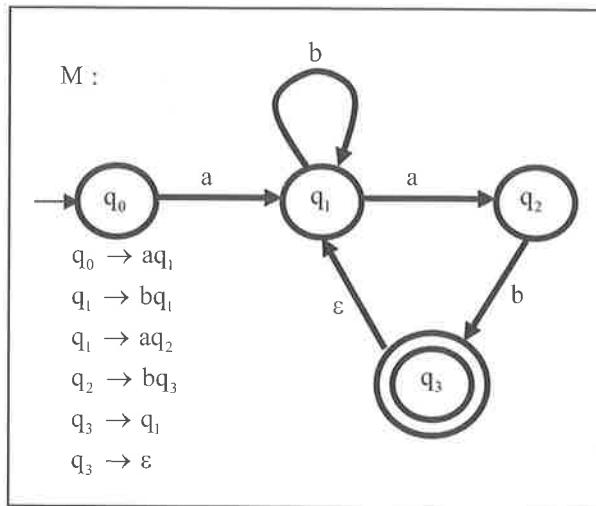
เนื่องจาก L เป็นภาษาเรกูลาร์ สามารถแทน L ด้วยไฟในต่ออโตมาตา NFA M :

รูปที่ 3.13 NFA ที่รองรับภาษา L

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

จาก NFA M สามารถแปลงไปเป็นไวยากรณ์เรกูลาร์แบบ Right-linear ตามขั้นตอนดังต่อไปนี้





ตัวอย่างที่ 3.20 จงพิสูจน์ว่าไวยากรณ์ดังต่อไปนี้สร้างภาษาอອกมาเป็นภาษาเรกุลาร์

$$G : V_1 \longrightarrow xV_2 \mid y$$

$$V_2 \longrightarrow yyV_2 \mid yy$$

ภาษาที่ได้จากไวยากรณ์นี้สามารถแยกแจงในรูปแบบของเซตได้ดังนี้

$$L(G) = \{y, xy, xyy, xyyyy, \dots\}$$

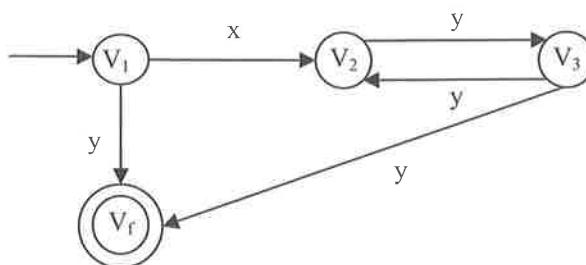
นอกจากจะพิสูจน์ความเป็นเรกุลาร์ของภาษาโดยการสร้างไฟฟ้าในต่อโตามาตรีขั้นมาของรับแล้ว ยังสามารถพิสูจน์ความเป็นเรกุลาร์ของภาษาโดยการสร้างนิพจน์เรกุลาร์สำหรับภาษา $L(G)$

$$\therefore \text{นิพจน์เรกุลาร์สำหรับ } L(G) \text{ คือ } xyy(yy)^*$$

เนื่องจากสามารถสร้างนิพจน์เรกุลาร์สำหรับภาษาที่ถูกสร้างอອกมาจาก G ได้ จึงสรุปได้ว่า $L(G)$ เป็นภาษาเรกุลาร์

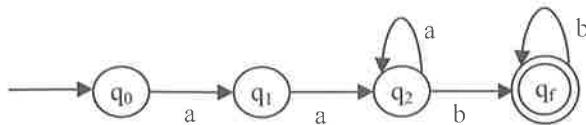
ตัวอย่างที่ 3.21 จงพิสูจน์ว่าไวยากรณ์ในตัวอย่างก่อนหน้านี้สร้างภาษาอອกมาเป็นภาษาเรกุลาร์ โดยใช้วิธีสร้างไฟฟ้าในต่อโตามาตรีเพื่อการพิสูจน์

สามารถสร้าง NFA สำหรับไวยากรณ์ดังกล่าวได้ดังนี้



รูปที่ 3.14 NFA สำหรับไวยากรณ์ G

ตัวอย่างที่ 3.22 สำหรับภาษา $L(G) = \{a^n b^m : n \geq 2, m \geq 1\}$ จงหาไวยากรณ์เรกูลาร์ G ก่อนอื่น เรายังสร้างไฟฟ้าโนตอมาตาสำหรับ $L(G)$ ดังนี้



รูปที่ 3.15 ไฟฟ้าโนตอมาตาสำหรับ $L(G)$

จากนั้น สร้างไวยากรณ์เรกูลาร์ G โดยสังเกตจาก DFA ได้ดังนี้

$$G = (\{q_0, q_1, q_2, q_f\}, \{a, b\}, \{q_0\}, P)$$

$$P : \quad q_0 \longrightarrow aq_1,$$

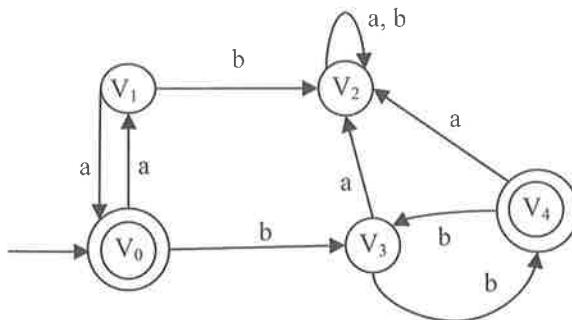
$$q_1 \longrightarrow aq_2,$$

$$q_2 \longrightarrow aq_2 \mid bq_f,$$

$$q_f \longrightarrow bq_f \mid \epsilon$$

แบบฝึกหัด

- จงให้เหตุผลว่าเหตุใดนิพจน์เรกุลาร์ทั้งสองนิพจน์ดังต่อไปนี้จึงเท่าเทียมกัน
 $(aaaa^*)^* = (aaa + aa)^*$
- จงยกตัวอย่างสตริงที่ถูกสร้างขึ้นมาจากนิพจน์เรกุลาร์ดังต่อไปนี้
 - $1^*(0+10)^*1^*$
 - $(0^*+1^*)(0^*+1^*)(0^*+1^*)$
 - $a^*(baa^*)^*b^*$
 - $(a+b)^*(a+bb)$
 - $(aa)^*(bb)^*b$
 - $(0+1)^*00(0+1)^*$
 - $(1+01)^*(0+\varepsilon)$
- กำหนดให้ภาษาประกอบด้วยตัวอักษร $\{a, b\}$ จงสร้างนิพจน์เรกุลาร์สำหรับภาษาที่สตริงไม่ลงท้ายด้วย ab
- จงสร้างไฟโนต์อโตมาตาเพื่อรับนิพจน์เรกุลาร์ดังต่อไปนี้
 $(0+1)^*(1+00)(01+10)^*$
- จงหา_nipun_rekularrที่สอดคล้องกับไฟโนต์อโตมาตาดังต่อไปนี้



รูปที่ 3.16 ไฟโนต์อโตมาตา

- จงยกตัวอย่างสตริงที่ถูกสร้างขึ้นมาจากไวยากรณ์เรกุลาร์ดังต่อไปนี้

$$\begin{aligned} G : \quad & V_1 \longrightarrow 010V_2 \mid 10 \\ & V_2 \longrightarrow 101V_2 \mid 110 \end{aligned}$$
- จงสร้างไวยากรณ์แบบ Right-linear สำหรับภาษาดังต่อไปนี้

$$L = \{1^n 0^m \mid n \geq 1, m \geq 2\}$$

8. จงสร้างไวยากรณ์แบบ Left-linear สำหรับภาษาในข้อ 7
9. จงสร้างไฟฟ์ในต่อโถมata ที่ร่องรับภาษาที่ถูกสร้างขึ้นจากไวยากรณ์ดังต่อไปนี้

$A \longrightarrow xyB,$

$B \longrightarrow yxC,$

$C \longrightarrow xB \mid yy$

10. จงอธิบายว่าไวยากรณ์เรกูลาร์ ภาษาเรกูลาร์ นิพจน์เรกูลาร์ และไฟฟ์ในต่อโถมata มีความสัมพันธ์กันอย่างไร

คุณสมบัติของภาษาเรกูลาร์

ดังที่ได้ทราบมาจากการที่ 2 และ 3 แล้วว่าภาษา L_1 และ L_2 จะเป็นภาษาเรกูลาร์ ถ้าสามารถสร้างไฟฟ์ในต่อไปนี้

อโตมาตาที่รับภาษา L_1 นี้ได้ ดังนั้นการสร้างไฟฟ์ในต่อไปนี้จะเป็นวิธีการหนึ่งที่ใช้พิสูจน์ความเป็นเรกูลาร์ ของภาษา อย่างไรก็ตาม ถ้าเราไม่สามารถสร้างไฟฟ์ในต่อไปนี้ได้ ภาษา L_1 ก็ไม่ได้เป็นการพิสูจน์ว่าภาษา L_1 นี้ ไม่ใช่ภาษาเรกูลาร์ (เนื่องจากอาจมีไฟฟ์ในต่อไปนี้ได้ แต่เราสร้างไม่ได้) ดังนั้นควรจะมีวิธีการพิสูจน์ว่าภาษา L_1 ไม่ใช่ภาษาเรกูลาร์ ในบทนี้จะกล่าวถึงคุณสมบัติปิดของภาษาเรกูลาร์และวิธีการพิสูจน์ความไม่เป็นภาษาเรกูลาร์

4.1 คุณสมบัติปิดของภาษาเรกูลาร์

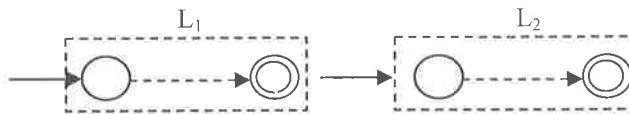
ภาษาเรกูลาร์มีคุณสมบัติปิดภายใต้การกระทำดังต่อไปนี้

- การยูเนียน (Union)
- การต่อกัน (Concatenation)
- การทำซ้ำ (Star)
- การย้อนกลับ (Reverse)
- การคอมพลีเมนต์ (Complement)
- การอินเตอร์เซกชัน (Intersection)
- การหาผลต่าง (Difference)

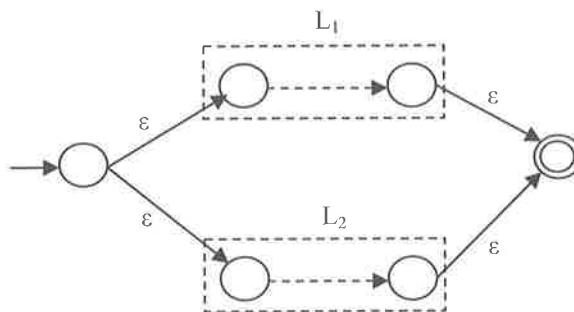
การที่ภาษาเรกูลาร์มีคุณสมบัติปิดภายใต้การกระทำข้างต้นนี้หมายความว่า หากเรากระทำการกับภาษาเรกูลาร์ ด้วยการยูเนียน การต่อกัน การทำซ้ำ การย้อนกลับ การคอมพลีเมนต์ การอินเตอร์เซกชัน และการหาผลต่าง ผลลัพธ์ที่ได้จะยังคงเป็นภาษาเรกูลาร์เสมอ

พิสูจน์ :

1. การยูเนียน (Union) : สมมุติให้ L_1 และ L_2 เป็นภาษาเรกูลาร์ซึ่งสามารถเขียนแทนด้วยไฟฟ์ในต่อไปนี้



โปรดสังเกตว่ารายละเอียดภายในไฟฟ้าในต่อๆ กันของ L_1 และ L_2 จะเป็นอย่างไรก็ได้ เราสนใจเพียงแค่สถานะเริ่มต้นและสถานะสุดท้ายเท่านั้น เมื่อนำ $L_1 \cup L_2$ ผลลัพธ์ที่ได้สามารถเขียนเป็นไฟฟ้าในต่อๆ กันได้ดังนี้

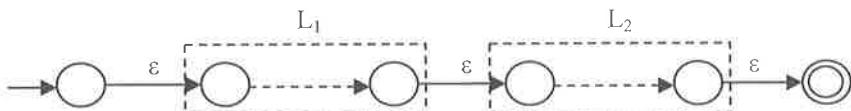


นั่นคือ สร้างสถานะเริ่มต้นใหม่ และโยงเส้นเชื่อมสถานะเริ่มต้นใหม่ไปยังสถานะเริ่มต้นเดิมของอโตมาตาทั้งสองโดยกำกับเส้นเชื่อมด้วยสตริงว่าง จากนั้นสร้างสถานะสุดท้ายใหม่ และโยงเส้นเชื่อมจากสถานะสุดท้ายเดิมของอโตมาตาทั้งสองไปยังสถานะสุดท้ายใหม่โดยกำกับเส้นเชื่อมด้วยสตริงว่าง เช่นกัน

เนื่องจากสามารถสร้างไฟฟ้าในต่อๆ กันของ NFA สำหรับภาษา $L_1 \cup L_2$ ได้ ดังนั้น $L_1 \cup L_2$ จึงเป็นภาษาเรกุลาร์

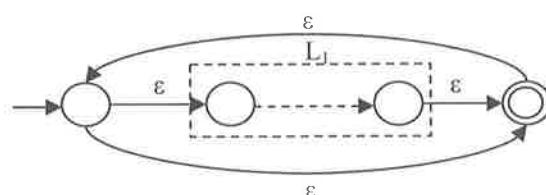
โดยอาศัยหลักการพิสูจน์ที่คล้ายคลึงกัน เราสามารถพิสูจน์กรณีการต่อกันและการทำซ้ำได้ดังนี้

2. การต่อ กัน (Concatenation) : $L_1 L_2$ สามารถเขียนไฟฟ้าในต่อๆ กันได้ดังนี้



ดังนั้น $L_1 L_2$ จึงเป็นภาษาเรกุลาร์

3. การทำซ้ำ (Star) : L_1^* สามารถเขียนไฟฟ้าในต่อๆ กันได้ดังนี้



ดังนั้น L_1^* เป็นภาษาเรกุลาร์

4. การย้อนกลับ (Reverse) : กำหนดให้ L_1 ถูกแทนด้วยไฟฟ์ในต่อไปนี้



สามารถกระทำการย้อนกลับกับไฟฟ์ในต่อไปนี้โดยเปลี่ยนทิศทางของลูกศรเป็นตรงกันข้าม และสลับสถานะกันระหว่างสถานะเริ่มต้นและสถานะสุดท้าย ดังนี้



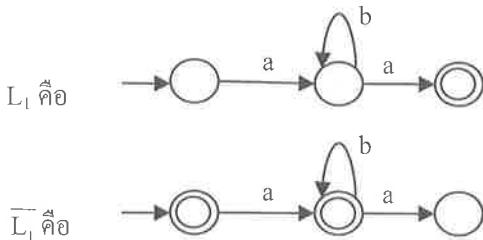
ดังนั้น L_1^R จึงเป็นภาษาเรกูลาร์

5. การคอมพลีเม้นต์ (Complement) : ในการพิสูจน์ว่าภาษา $\overline{L_1}$ เป็นภาษาเรกูลาร์ จะสร้างอัลกอริทึม (Construction Proof) เพื่อใช้ในการเขียนไฟฟ์ในต่อไปนี้

อัลกอริทึม : การสร้างไฟฟ์ในต่อไปนี้โดยรับคอมพลีเม้นต์

- (1) เปลี่ยนสถานะสุดท้ายของอโตมาตาที่รับ L_1 ให้กลายเป็นสถานะที่ไม่ใช่สถานะสุดท้าย
- (2) เปลี่ยนสถานะที่ไม่ใช่สถานะสุดท้ายให้กลายเป็นสถานะสุดท้าย

เช่น



ดังนั้น $\overline{L_1}$ จึงเป็นภาษาเรกูลาร์

6. การอินเตอร์เซกชัน (Intersection) : ในการพิสูจน์ว่า $L_1 \cap L_2$ เป็นภาษาเรกูลาร์ จะเขียน $L_1 \cap L_2$ อยู่ในรูปของการยูเนียนและการคอมพลีเม้นต์ตามกฎของเดอร์มอร์แกนดังนี้

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

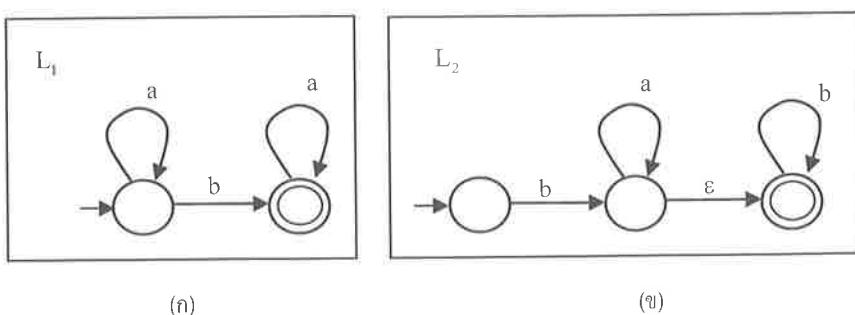
ดังที่ทราบมาแล้วว่าภาษาเรกูลาร์มีคุณสมบัติปิดภายใต้การยูเนียนและการคอมพลีเม้นต์ ดังนั้น $L_1 \cap L_2$ จึงเป็นภาษาเรกูลาร์

7. การหาผลต่าง (Difference) : $L_1 - L_2$ ผลต่างระหว่างภาษา L_1 และ L_2 ถูกนิยามดังนี้

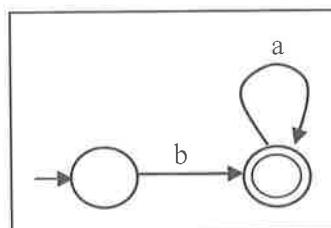
$$L_1 - L_2 = L_1 \cap \overline{L_2}$$

เนื่องจากภาษาเรกูลาร์มีคุณสมบัติปิดภายใต้การอินเตอร์เซกชันและการคอมพลีเม้นต์ ดังนั้น $L_1 - L_2$ จึงเป็นภาษาเรกูลาร์

ตัวอย่างที่ 4.1 กำหนดให้ $L_1 = \{a^m b a^n : m \text{ และ } n \geq 0\}$ และ $L_2 = \{ba^m b^n : m \text{ และ } n \geq 0\}$
 จงหาภาษาซึ่งเกิดจาก $L_1 \cap L_2$
 ภาษา L_1 และ L_2 สามารถแทนได้ด้วย NFA ดังต่อไปนี้

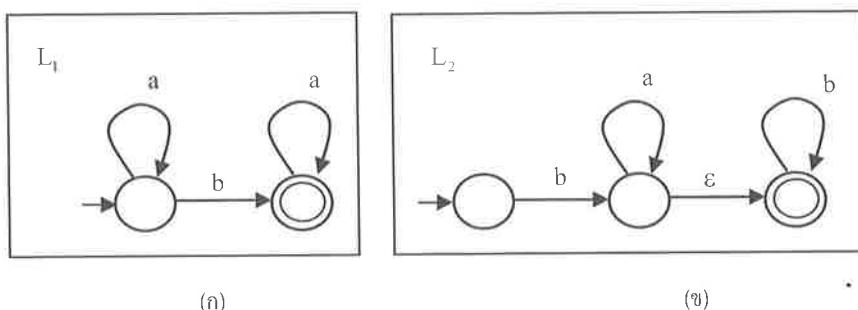
รูปที่ 4.1 NFA สำหรับ L_1 และ L_2

ในการหาไฟน์ต์อโตมาตาที่รับ $L_1 \cap L_2$ เราจะต้องหาส่วนของอโตมาตาที่มีความเหมือนกันและใหญ่ที่สุดของทั้ง L_1 และ L_2 นั่นคือ ba^*

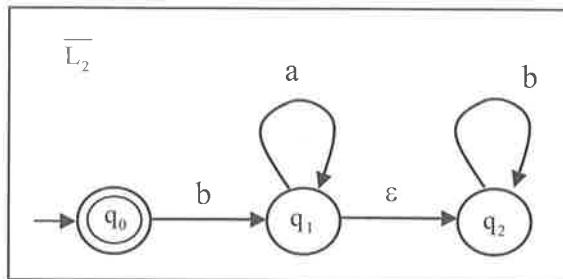
รูปที่ 4.2 ไฟน์ต์อโตมาตาสำหรับ ba^*

ดังนั้นภาษาที่เกิดจาก $L_1 \cap L_2$ คือ $\{ba^m : m \geq 0\}$ ซึ่งเป็นภาษาเรกุลาร์ เนื่องจากสามารถเขียนไฟน์ต์อโตมาตามรูปได้

ตัวอย่างที่ 4.2 กำหนดให้ L_1 และ L_2 ถูกนิยามดังเช่นตัวอย่างก่อนหน้านี้ จงหาภาษาซึ่งเกิดจากภาษา $L_1 - L_2$

รูปที่ 4.3 NFA สำหรับ L_1 และ L_2

เนื่องจาก $L_1 - L_2 = L_1 \cap \overline{L_2}$ สามารถสร้าง NFA สำหรับ $\overline{L_2}$ ได้ดังนี้



รูปที่ 4.4 NFA สำหรับ $\overline{L_2}$

โปรดสังเกตว่าที่สถานะ q_1 เราไม่ได้กำหนดให้เป็นสถานะสุดท้าย เนื่องจากสถานะ q_1 ถูกเชื่อมกับสถานะ q_2 โดยใช้สตริงว่าง ดังนั้นจึงถือได้ว่าสถานะ q_1 และสถานะ q_2 เป็นสถานะเดียวกัน

เนื่องจากสตริงที่เป็นสมาชิกอยู่ใน $\overline{L_2}$ จะไม่สามารถมีตัวอักษร b ได้เลยแม้แต่ตัวเดียว แต่สำหรับภาษา L_1 และ สตริงที่เป็นสมาชิกจะต้องมีตัวอักษร b อยู่หนึ่งตัว ดังนั้นเซตของสตริงที่เกิดจากผลต่างของ L_1 และ L_2 จึงเป็นเซตว่าง

$$L_1 - L_2 = L_1 \cap \overline{L_2} = \emptyset$$

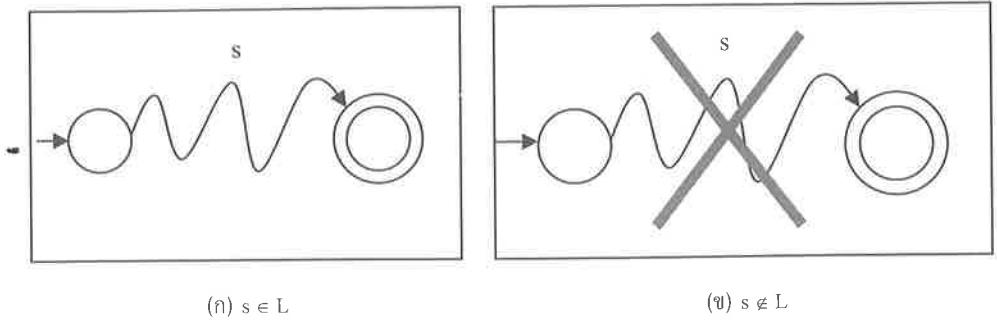
4.2 คำความพื้นฐานเกี่ยวกับภาษาเรกูลาร์

หลังจากที่ศึกษาคุณสมบัติของภาษาเรกูลาร์แล้ว ขั้นตอนต่อไปคือการศึกษาคำความพื้นฐานที่เกี่ยวข้องกับภาษาเรกูลาร์ คำความเหล่านี้จะเป็นประโยชน์อย่างยิ่งในการพิสูจน์ความไม่เป็นเรกูลาร์ของภาษา ซึ่งเราจะได้ศึกษาในหัวข้อต่อไป

1. คำความเกี่ยวกับการเป็นสมาชิก

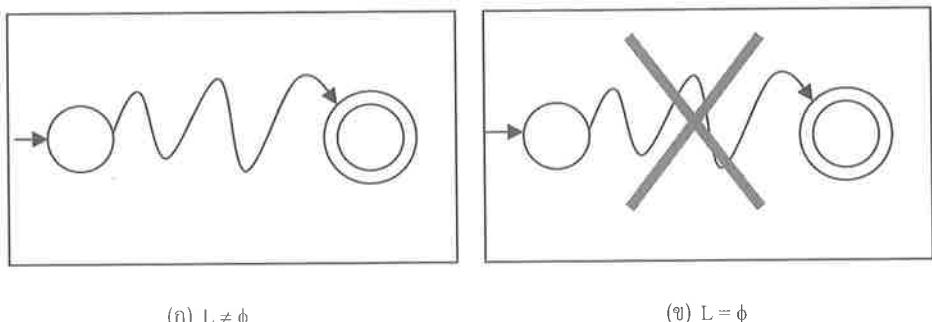
คำความที่ 1 กำหนดให้ L เป็นภาษาเรกูลาร์ และ s เป็นสตริงใด ๆ เราจะตรวจสอบได้อย่างไรว่าสตริง s เป็นสมาชิกในภาษา L ?

คำตอบ : สร้างไฟล์อโตมาตา (DFA หรือ NFA) ที่รองรับภาษา L จากนั้นส่งสตริง s เข้าไปในอโตมาตาโดยเริ่มจากสถานะเริ่มต้น หากสตริง s สามารถทำให้ไฟล์อโตมาตาเปลี่ยนสถานะเข้าสู่สถานะสุดท้ายได้ แสดงว่าสตริง s นั้นเป็นสมาชิกในภาษา L

รูปที่ 4.5 การตรวจสอบว่าสตริง s เป็นสมาชิกในภาษา L

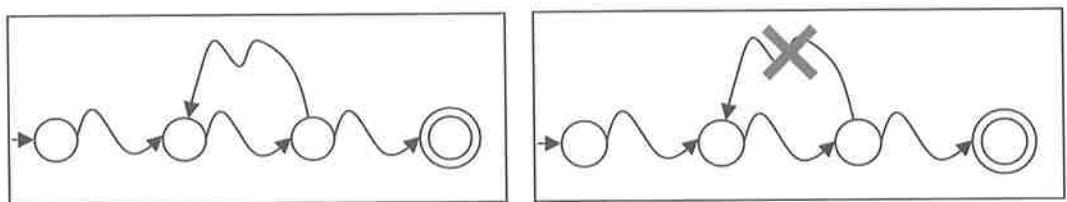
คำถามที่ 2 กำหนดให้ L เป็นภาษาเรกูลาร์ จะทราบได้อย่างไรว่าภาษา L ไม่มีสตริงซึ่งเป็นสมาชิกอยู่เลย ($L = \emptyset$) ?

คำตอบ : สร้างไฟฟ้าในตัวอัตโนมัติ (DFA หรือ NFA) ที่รับภาษา L จากนั้นตรวจสอบดูว่ามีเส้นทางที่เริ่มต้นจากสถานะเริ่มต้น และไปถึงสุดที่สถานะสุดท้ายหรือไม่ หากไม่มี แสดงว่าภาษา L ไม่มีสตริงที่เป็นสมาชิกอยู่เลย

รูปที่ 4.6 การตรวจสอบว่าภาษา L เป็นเซตว่าง

คำถามที่ 3 กำหนดให้ L เป็นภาษาเรกูลาร์ เรายังทราบได้อย่างไรว่าภาษา L เป็นเซตไม่จำกัด ?

คำตอบ : สร้างไฟฟ้าในตัวอัตโนมัติ (DFA หรือ NFA) ที่รับภาษา L จากนั้นตรวจสอบดูว่ามีใช้เคิลที่อยู่ระหว่างเส้นทางจากสถานะเริ่มต้นไปยังสถานะสุดท้ายหรือไม่ ถ้ามีใช้เคิล แสดงว่าภาษา L เป็นเซตไม่จำกัด

(g) ภาษา L เป็นเซตไม่จำกัด(x) ภาษา L เป็นเซตจำกัดรูปที่ 4.7 การตรวจสอบว่าภาษา L เป็นเซตไม่จำกัด

คำถามที่ 4 กำหนดให้ L_1 และ L_2 เป็นภาษาเรกูลาร์ เราจะทราบได้อย่างไรว่าภาษา $L_1 = L_2$?

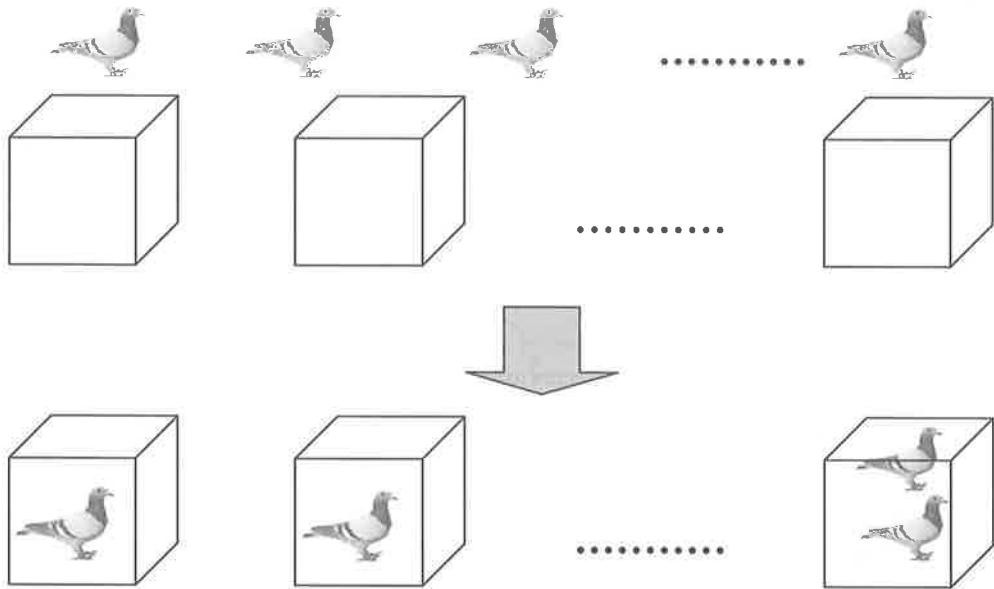
คำตอบ : พิสูจน์ว่า $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$ หากเป็นจริง แสดงว่า $L_1 = L_2$

2. ภาษาที่ไม่ใช่ภาษาเรกูลาร์

นอกจากภาษาเรกูลาร์แล้ว ภาษาที่ใช้ในการเขียนโปรแกรมคอมพิวเตอร์ยังมีภาษาที่ไม่ใช่ภาษาเรกูลาร์ เช่น ภาษา $\{a^n b^n : n \geq 0\}^4$ ซึ่งไม่ใช่ภาษาเรกูลาร์ เนื่องจากไฟฟ้าในต่อๆ กันไม่สามารถไม่เพียงพอที่จะจำได้ว่ามีจำนวนตัวอักษร a กี่ตัว เพื่อจะนำมาตรวจสอบตัวอักษร b ให้มีจำนวนเท่ากับ a อย่างไรก็ตาม เหตุผลนี้ไม่ถือว่าเป็นเหตุผลที่เป็นทางการ จึงไม่สามารถใช้พิสูจน์ความไม่เป็นเรกูลาร์ของภาษาได้

วิธีการพิสูจน์ความไม่เป็นเรกูลาร์ของภาษาอยู่บนพื้นฐานของหลักการรังนกพิราน (Pigeonhole Principle)

“มีรังนกพิรานอยู่ n ตัว และมีรังนกพิรานอยู่ m รัง โดยที่ $n > m$ หากนกพิรานทั้ง n ตัวสามารถบินเข้าไปอยู่ในรังได้ทุกตัว แสดงว่าจะต้องมีรังอย่างน้อยหนึ่งรังที่มีนกพิรานเข้าไปอยู่เกิน 1 ตัว”

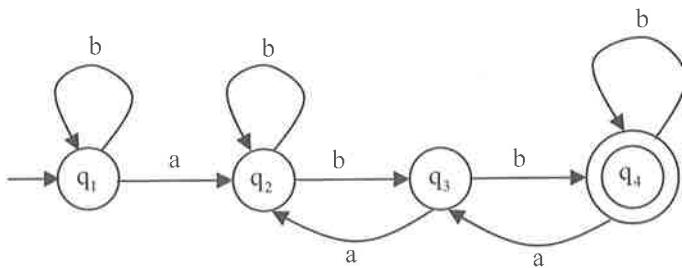


รูปที่ 4.8 หลักการรังนกพิราน (Pigeonhole Principle)

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

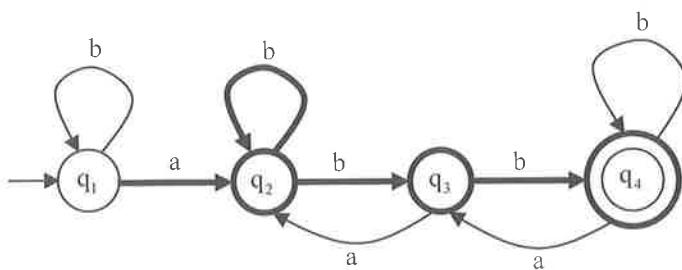
จากหลักการนี้ เมื่อนำมาเปรียบเทียบกับการทำงานของไฟฟ้าในต่อๆ กันมา สามารถอธิบายได้ดังนี้ สมมุติว่ามีไฟฟ้าในต่อๆ กันมา DFA คือ

⁴ ตัวอักษร a เปรียบเสมือนสัญลักษณ์เปิดขอบเขต เช่น $\{ ($ ส่วนตัวอักษร b เปรียบเสมือนสัญลักษณ์ปิดขอบเขต เช่น $\}$) ตั้งนั้นสติง $aabb$ จึงเปรียบเสมือน $\{\} \}$ หรือ $(())$ นั่นเอง



รูปที่ 4.9 ไฟฟ้าในต่อโถมata DFA

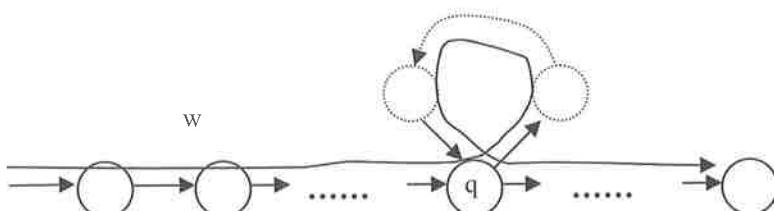
หากแทนนกพิราบด้วยตัวอักษรที่อยู่ภายในอินพุตสตริง และแทนรังของนกพิราบด้วยสถานะ เหตุการณ์ที่คล้ายกันก็จะเกิดขึ้นกับไฟฟ้าในต่อโถมata นั่นคือ ถ้าจำนวนตัวอักษรภายในอินพุตสตริงมีมากกว่าหรือเท่ากับจำนวนสถานะภายในไฟฟ้าในต่อโถมata จะต้องมีบางสถานะที่ถูกวนซ้ำจากการเกิดทราบซึ่งกันของอินพุตสตริงนั้น เช่น สำหรับอินพุตสตริง abbb มีจำนวนตัวอักษรเท่ากับจำนวนของสถานะ (ในที่นี้มีจำนวนเท่ากับ 4) จะทำให้เกิดการวนซ้ำตั้งนี้



รูปที่ 4.10 การวนซ้ำภายในไฟฟ้าในต่อโถมata

สถานะ q_2 เป็นสถานะที่ถูกวนซ้ำเนื่องจากตัวอักษร b ที่เกินมา 1 ตัว นอกจ้านี้ยังสามารถให้ q_4 เป็นสถานะที่ถูกวนซ้ำได้หากเรามาเลือกที่จะวนซ้ำที่สถานะ q_2

สามารถสรุปหลักการรังนกพิราบได้ดังนี้ “สำหรับไฟฟ้าในต่อโถมata ใด ๆ หากอินพุตสตริง w มีจำนวนตัวอักษรมากกว่าหรือเท่ากับจำนวนสถานะภายในไฟฟ้าในต่อโถมata จะต้องมีบางสถานะ q ถูกวนซ้ำจากการผ่านสตริง w เข้าไปในอตโนมata”



รูปที่ 4.11 รูปแบบทั่วไปในการวนซ้ำ

หลักการนี้จะเป็นพื้นฐานสำคัญที่ใช้ในการพิสูจน์ความไม่เป็นเรกูลาร์ของภาษา ซึ่งกำลังจะได้ศึกษาในหัวข้อต่อไป

4.3 ปั๊มปิงเลมมาสำหรับภาษาเรกูลาร์

ปั๊มปิงเลมมา (Pumping Lemma) เป็นวิธีการพิสูจน์โดยการขัดแย้ง (Contradiction Proof) ว่าภาษาที่กำลังพิจารณา $\text{ไม่เป็นภาษาเรกูลาร์}$ สำหรับในบทนี้จะศึกษาเฉพาะปั๊มปิงเลมมาของภาษาเรกูลาร์เท่านั้น ส่วนปั๊มปิงเลมมาสำหรับภาษาคอนเทกซ์ฟรีจะศึกษาในบทที่ 7

ทฤษฎี 4.3.1 : กำหนดให้ L เป็นภาษาเรกูลาร์ที่สามารถเขียนໄไฟในต่อโถมตามาที่มีจำนวนของสถานะเท่ากับ m และสำหรับทุก ๆ สตริง $s \in L$ โดยที่ $|s| \geq m$ สามารถแบ่งสตริง s ออกมาเป็นสตริงย่อย ๆ ได้ดังนี้

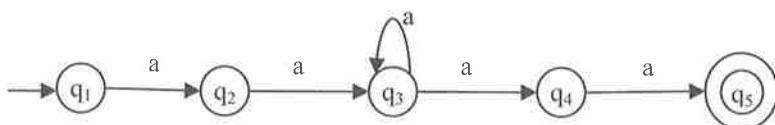
$$s = xyz \quad ; \text{ โดยที่ } |xy| \leq m \text{ และ } |y| = k ; k \geq 1$$

หากทำซ้ำ (Pump^r) สตริงย่อย y ตั้งแต่ 0 ครั้งขึ้นไป ผลที่ได้จะได้เป็นสตริงซึ่งยังคงเป็นสมาชิกอยู่ภายใน L :

$$xy^iz \in L \quad ; i = 0, 1, 2, \dots \quad \#$$

เพื่อให้เกิดความเข้าใจในปั๊มปิงเลมมา พิจารณาดูໄไฟในต่อโถมตามาของภาษาดังต่อไปนี้

$$L = \{a^n : n \geq 4\}$$



รูปที่ 4.12 ໄไฟในต่อโถมตามาของภาษา L

จากรูป ໄไฟในต่อโถมตามาที่มีจำนวนสถานะเท่ากับ 5 ซึ่งหากไม่มีการวนลูปภายใน จะสามารถรับสตริง $aaaaa$ ได้เท่านั้น แต่เนื่องจากໄไฟในต่อโถมตามาต้องกล่าวสามารถรับสตริง $aaaaaa$ ได้ด้วย ดังนั้นจะต้องมีการวนลูปกับสตริงบางส่วนอย่างแน่นอน หากแบ่งสตริง $aaaaaa$ ออกเป็น 3 ส่วน คือ

$$x = aa, \quad y = a, \quad z = aa$$

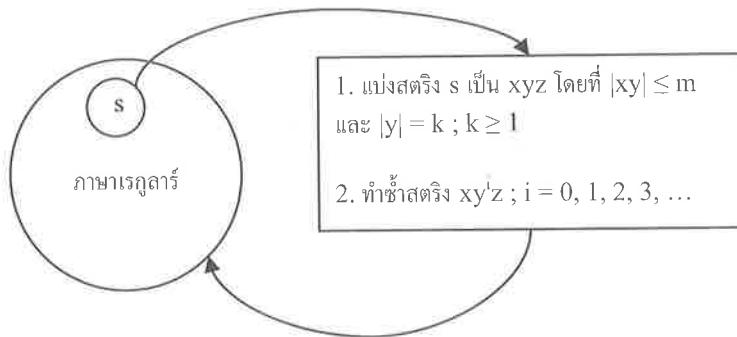
จากนั้นทำซ้ำส่วน y เป็นจำนวน i ครั้ง

$$xy^iz \quad ; \text{ โดยที่ } i = 0, 1, 2, 3, \dots$$

ผลที่ได้คือ $aa(a)^ia ; i = 0, 1, 2, 3, \dots$ ซึ่งสตริงดังกล่าวก็ยังคงเป็นสมาชิกอยู่ภายในภาษา L ดังนั้นมีอเลือกทำซ้ำส่วน y ด้วย $i = 2$ จะได้ผลลัพธ์ออกมานี้เป็นสตริง $aaaaaa$ ซึ่งยังคงเป็นสมาชิกอยู่ภายในภาษา L

^๑ ในที่นี่เราว่าใช้คำว่า Pump ในความหมายของการทำซ้ำ (Repetition)

สรุป : สำหรับภาษา L ที่เป็นภาษาเรกูลาร์ และสตริง $s \in L$ สามารถนำ s มาแบ่งเป็น 3 ส่วน xyz โดยที่ $|xy| \leq m$ มีขนาดน้อยกว่าหรือเท่ากับจำนวนของสถานะภายในไฟฟ้าในต่ออโตมาตาของ L และ $|y| \geq 1$ จากนั้นทำซ้ำสตริง $xy^iz ; i = 0, 1, 2, 3, \dots$ ผลที่ได้จากการทำซ้ำจะต้องเป็นสตริงที่เป็นสมาชิกอยู่ภายในภาษา L



รูปที่ 4.13 การทำซ้ำสตริง s ภายในภาษาเรกูลาร์

ข้อควรระวัง สำหรับการใช้ปั๊มปิงเลมมาในการพิสูจน์ในบทนี้คือ ห้ามใช้ปั๊มปิงเลมมา กับภาษาที่ เป็นเรกูลาร์ มิฉะนั้นผลที่ได้จากการพิสูจน์จะไม่สามารถเชื่อถือได้ ดังนั้นเราจะต้องผึกสร้างไฟฟ้าในต่ออโตมาตาจนชำนาญเพื่อจะได้ทราบล่วงหน้าว่าภาษาที่กำลังจะพิสูจน์นั้นไม่มีไฟฟ้าในต่ออโตมาตาใด ๆ รองรับ การใช้ปั๊มปิงเลมมาเป็นเพียงเครื่องมือที่ใช้ในการยืนยันความไม่เป็นเรกูลาร์ของภาษาและจะต้องเป็นการพิสูจน์โดยการขัดแย้งเท่านั้น

4.4 วิธีการพิสูจน์ความไม่เป็นเรกูลาร์ของภาษาโดยใช้ปั๊มปิงเลมมา

การพิสูจน์โดยปั๊มปิงเลมมา มีลักษณะ เช่นเดียวกับการพิสูจน์โดยการขัดแย้ง นั่นคือจะต้องหาความขัดแย้งระหว่างสมมุติฐานที่ตั้งขึ้น กับผลที่ได้จากการพิสูจน์ โดยการพิสูจน์ดังกล่าวจะคล้ายกับการเล่นเกมระหว่างสองฝ่าย ฝ่ายหนึ่งต้องการพิสูจน์ว่าภาษา L ไม่ใช่ภาษาเรกูลาร์ แต่อีกฝ่ายหนึ่งต้องการพิสูจน์ว่าภาษา L เป็นภาษาเรกูลาร์ การพิสูจน์มีขั้นตอนดังนี้

ขั้นตอนในการพิสูจน์ว่าภาษา L ไม่ใช่ภาษาเรกูลาร์

1. เริ่มต้นด้วยการให้ฝ่ายตรงข้ามเล่นเกมก่อน โดยสมมุติว่าภาษา L เป็นภาษาเรกูลาร์ พร้อมกันให้ฝ่ายตรงข้ามบอกจำนวนสถานะที่ใช้ในการสร้างไฟฟ้าในต่ออโตมาตาสำหรับภาษา L (สมมุติให้จำนวนของสถานะมีค่าเท่ากับ m)
2. เลือกสตริง $s \in L$ โดยที่ $|s| \geq m$ (ถึงขั้นตอนนี้ เราหวังว่าเมื่อทำซ้ำสตริง s แล้ว จะได้สตริงใหม่ เป็น s' โดยที่ $s' \notin L$ ซึ่งก็จะขัดแย้งกับสมมุติฐานที่ตั้งไว้ในข้อ 1 และขัดแย้งกับปั๊มปิงเลมมา เนื่องจากหากภาษา L เป็นภาษาเรกูลาร์ เมื่อนำสตริงได ๆ ที่เป็นสมาชิกของ L มาทำซ้ำ ผลที่ได้จะต้องได้สตริงที่ยังคงเป็นสมาชิกอยู่ภายใน L)

3. ฝ่ายตรงข้ามแยกสตริง s ออกเป็น 3 ส่วน xyz โดยที่ $|xy| \leq m$ และ $|y| \geq 1$ (ขั้นตอนนี้ฝ่ายตรงข้าม มีความหวังว่าเมื่อทำซ้ำสตริง (เฉพาะส่วน y) xy^iz ; $i = 0, 1, 2, \dots$ แล้ว ผลที่ได้จะได้สตริง s' โดยที่ $s' \in L$ เพื่อจะได้สอดคล้องกับสมมุติฐานที่ตั้งไว้ในข้อที่ 1 และสอดคล้องกับปั๊มปิงเลมมา)

4. ทำซ้ำส่วน y ของสตริง s โดยเลือกค่า i ที่ทำให้ $xy^iz \notin L$; $i = 0, 1, 2, \dots$ หากมี i แม้เพียงแค่ ค่าเดียวที่ทำให้ $xy^iz \notin L$ ก็จะเกิดความขัดแย้งกับปั๊มปิงเลมมาทันที ซึ่งก็ทำให้สมมุติฐานที่ตั้งไว้ในข้อ 1 ผิด และทำให้เราสามารถสรุปได้ว่าภาษา L ไม่ใช่ภาษาเรกูลาร์

ตัวอย่างที่ 4.3 จะใช้ปั๊มปิงเลมมาพิสูจน์ว่า $L = \{a^n b^n : n \geq 0\}$ ไม่ใช่ภาษาเรกูลาร์

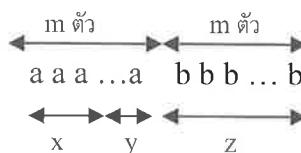
พิสูจน์ : ในที่นี้เราต้องการพิสูจน์ว่า L ไม่ใช่ภาษาเรกูลาร์ แต่ฝ่ายตรงข้ามยืนยันว่า L เป็นภาษาเรกูลาร์

ขั้นตอนที่ 1 ฝ่ายตรงข้ามเริ่มก่อน โดยสมมุติว่า L เป็นภาษาเรกูลาร์ โดยมีไฟโนต์อโตมาตาที่มีจำนวนสถานะเท่ากับ m รองรับ

ขั้นตอนที่ 2 เลือกสตริง $s \in L$ โดยที่ $|s| \geq m$

$$s = a^m b^m$$

ขั้นตอนที่ 3 ฝ่ายตรงข้ามแบ่งสตริง s ออกเป็น 3 ส่วน xyz ดังนี้



เนื่องจากเงื่อนไขในการแบ่งสตริงของปั๊มปิงเลมมาบอกว่า $|xy| \leq m$ และ $|y| \geq 1$ ดังนั้นสตริง xyz ของ y จะต้องประกอบด้วยตัวอักษร a เท่านั้น ห้ามมีตัวอักษร b มาปนอยู่ด้วย นั่นคือ $xyz = a^{m-k} a^k b^m$; $1 \leq k \leq m$

ขั้นตอนที่ 4 เลือกค่า i ที่ทำให้ $xy^iz \notin L$ โดย i มีค่าได้ตั้งแต่ $0, 1, 2, \dots$

เลือก $i = 0$ ซึ่งจะทำให้ $a^{m-k}(a^k)^0 b^m = a^{m-k} b^m$; $1 \leq k \leq m$

จะเห็นได้ว่าจำนวนตัวอักษร a ไม่มีทางเท่ากับจำนวนตัวอักษร b เนื่องจาก k มีค่าไม่เท่ากับ 0 ดังนั้น $a^{m-k} b^m \notin L$ ซึ่งขัดแย้งกับสมมุติฐานที่ฝ่ายตรงข้ามได้ตั้งไว้ในขั้นตอนที่ 1 ว่า L เป็นภาษาเรกูลาร์ เนื่องจากหากภาษา L เป็นภาษาเรกูลาร์ เมื่อทำซ้ำสตริงได้ ๆ ที่เป็นสามาชิกอยู่ภายใน L จะต้องได้ออกมาเป็นสตริงที่ยังคงเป็นสามาชิกอยู่ภายใน L ไม่ว่าจะทำซ้ำสตริงด้วยค่า i ใด ๆ ก็ตาม

ดังนั้นจึงสรุปได้ว่าสมมุติฐานที่ฝ่ายตรงข้ามตั้งไว้ว่า L เป็นภาษาเรกูลาร์นั้นไม่ถูกต้อง จึงพิสูจน์ได้แล้วว่า $L = \{a^n b^n : n \geq 0\}$ ไม่ใช่ภาษาเรกูลาร์

ตัวอย่างที่ 4.4 จงพิสูจน์ว่าภาษา $L = \{s \in \{0, 1\}^* : \text{จำนวนเลข } 0 \text{ ของ } s \text{ เท่ากับจำนวนเลข } 1 \text{ ของ } s\}$ ไม่ใช่ภาษาเรกูลาร์

พิสูจน์ : ตัวอย่างนี้คล้ายกับตัวอย่างที่แล้ว ซึ่งสามารถใช้สตริงที่มีลักษณะเดียวกันในการพิสูจน์ตามวิธีของปั๊มปิงเลมมาได้

- (1) สมมุติว่าภาษา L เป็นเรกูลาร์ โดยมีจำนวนสถานะของไฟฟ้าในต่อโถมาต้าที่รับภาษา L เท่ากับ m
 (2) เลือกสตริง $s \in L$ และ $|s| \geq m$

$$s = 0^m 1^m$$

- (3) จากตัวอย่างที่แล้ว ได้พิสูจน์แล้วว่าเมื่อเลือก $i = 0$ จะได้ $0^{m-k}(0^k)^0 1^m \notin L$ เนื่องจากจำนวนเลข 0 ไม่เท่ากับจำนวนเลข 1

ดังนั้นภาษา L จึงไม่ใช่ภาษาเรกูลาร์

ตัวอย่างที่ 4.5 กำหนดให้ $\Sigma = \{a, b\}$ จงพิสูจน์ว่า

$$L = \{vv : v \in \Sigma^*\} \text{ ไม่ใช่ภาษาเรกูลาร์}$$

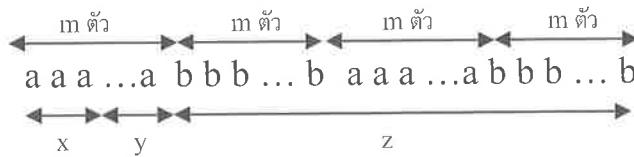
พิสูจน์ : โดยใช้ปั๊มปิงเลมมา

- (1) สมมุติว่า L เป็นภาษาเรกูลาร์โดยมีจำนวนสถานะของไฟฟ้าในต่อโถมาต้าที่รับภาษา L เท่ากับ m

- (2) เลือกสตริง $s \in L$ และ $|s| \geq m$

$$s = a^m b^m a^m b^m$$

- (3) แบ่งสตริง s ออกเป็น 3 ส่วน xyz ดังนี้



โดยที่ $|xy| \leq m$ และ $|y| = k ; k \geq 1$

$$xyz = a^{m-k} a^k b^m a^m b^m$$

- (4) เลือกทำซ้ำสตริง $xy^i z$ โดยกำหนดให้ $i = 0$

$$a^{m-k} (a^k)^0 b^m a^m b^m$$

$$a^{m-k} b^m a^m b^m \notin L$$

เนื่องจากผลที่ได้จากการทำซ้ำไม่อยู่ในภาษา L ดังนั้นจึงขัดแย้งกับปั๊มปิงเลมมา จึงสรุปได้ว่า L ไม่ใช่ภาษาเรกูลาร์

ข้อควรระวัง สิ่งหนึ่งที่สำคัญมากในการพิสูจน์ปั๊มปิงเลมมา คือ การเลือกสตริง s หากเลือกสตริง s ที่ไม่เหมาะสม ฝ่ายตรงข้ามจะสามารถแบ่งสตริง s ออกเป็น xyz โดยที่ไม่เปิดโอกาสให้ทำซ้ำ (Pump) สตริง $xy^i z$ ออกจากภาษา L ได้เลย ไม่ว่าจะเลือกค่า i ใด ๆ ก็ตาม ตัวอย่างของสตริง s ที่ไม่เหมาะสม เช่น

$$s = a^m a^m \text{ (สังเกตว่า } s \in L \text{ และ } |s| \geq m \text{ ตามนิยามของปั๊มปิงเลมมา)}$$

หากฝ่ายตรงข้ามเลือกแบ่งสตริง s โดยกำหนดให้ $y = aa$

$$s = a^{m-2} a^2 a^m$$

ไม่ว่าจะทำซ้ำสตริง $a^{m-2} (a^2)^i a^m$ ด้วยค่า i เท่าใดก็ตาม จะไม่สามารถทำให้สตริงที่ถูกทำซ้ำหลุดออกจากภาษา L ได้เลย

เช่น $i = 0$ จะได้สตริง $a^{m-2}a^m$ ซึ่งเทียบเท่ากับสตริง $a^{m-1}a^{m-1} \in L$

$i = 2$ จะได้สตริง $a^{m+2}a^m$ ซึ่งเทียบเท่ากับสตริง $a^{m+1}a^{m+1} \in L$

$i = 3$ จะได้สตริง $a^{m+4}a^m$ ซึ่งเทียบเท่ากับสตริง $a^{m+2}a^{m+2} \in L$

ดังนั้นสตริง s ที่เราเลือกขึ้นมาจะต้องทำให้ $xy^iz \notin L$ ได้ทุกรูปแบบไม่ว่าฝ่ายตรงข้ามจะกำหนดให้ y เป็นสตริงอะไรก็ตามที่ถูกต้องตามกฎของปีมปิงเลมมา

ตัวอย่างที่ 4.6 กำหนดให้ $\Sigma = \{a, b\}$ พิสูจน์ว่า

$$L = \{vv^R : v \in \Sigma^*\}$$
 ไม่ใช่ภาษาเรกูลาร์

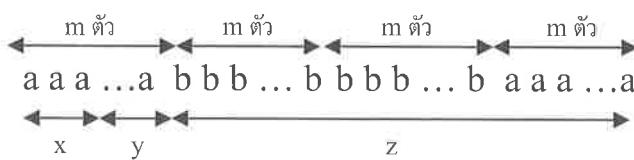
พิสูจน์ : โดยใช้ปีมปิงเลมมา

(1) สมมุติว่า L เป็นภาษาเรกูลาร์ โดยมีจำนวนสถานะของไฟฟ้าในต่อโถมาตาที่รองรับภาษา L เท่ากับ m

(2) เลือกสตริง $s \in L$ และ $|s| \geq m$

$$s = a^m b^m b^m a^m$$

(3) แบ่งสตริง s ออกเป็น 3 ส่วน xyz ดังนี้



โดยที่ $|xy| \leq m$ และ $|y| = k ; k \geq 1$

$$xyz = a^{m-k}a^k b^m b^m a^m$$

(4) เลือกทำซ้ำสตริง xy^iz โดยกำหนดให้ $i = 0$

$$a^{m-k}(a^k)^0 b^m b^m a^m$$

$$a^{m-k}b^m b^m a^m \notin L$$

เนื่องจากผลที่ได้จากการทำซ้ำไม่มีอยู่ในภาษา L จึงขัดแย้งกับปีมปิงเลมมา ดังนั้นสรุปได้ว่า L ไม่ใช่ภาษาเรกูลาร์

ผู้อ่านพยายามท่านอาจสงสัยว่ามีหลักการในการเลือกทำซ้ำสตริง y (ภายในสตริง s) ด้วยค่า i เท่ากับเท่าใด จึงจะทำให้สตริง xy^iz ถูกทำซ้ำจนกระทั้งขัดแย้งกับปีมปิงเลมมา จากตัวอย่างที่แสดงให้ดูส่วนใหญ่มักจะใช้ค่า i เป็น 0 แต่ในหลักการพิสูจน์ปีมปิงเลมมา เราสามารถเลือกค่า i ได้ ๆ อย่างน้อย 1 ค่า ที่สามารถทำให้สตริง xy^iz ถูกทำซ้ำ (Pump) จนกระทั้งขัดแย้งกับปีมปิงเลมมา สำหรับตัวอย่างนี้หากเลือกค่า $i = 2$ ก็สามารถทำให้เกิดการขัดแย้งในการพิสูจน์ได้ เช่นกัน เนื่องจากจะทำให้

$$a^{m-k}(a^k)^2 b^m b^m a^m$$

$$a^{m+k}b^m b^m a^m \notin L$$
 โดยที่ $k \geq 1$

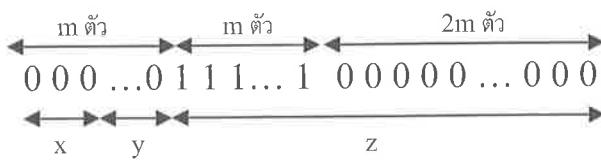
ดังนั้นสมมุติฐานที่ตั้งไว้ว่า L เป็นภาษาเรกูลาร์จึงไม่เป็นความจริง

ตัวอย่างที่ 4.7 กำหนดให้ภาษา $L = \{0^p 1^q 0^{p+q} : p, q \geq 0\}$ จงพิสูจน์ว่าภาษา L ไม่ใช่ภาษาเรกุลาร์ พิสูจน์ : โดยใช้ปั๊มปิงเลมมา

- (1) สมมุติว่า L เป็นภาษาเรกุลาร์ โดยมีจำนวนสถานะของไฟฟ้าในต่อโถมata ที่รองรับภาษา L เท่ากับ m
- (2) เลือกสตริง $s \in L$ และ $|s| \geq m$

$$s = 0^m 1^m 0^{m+m}$$

- (3) แบ่งสตริง s ออกเป็น 3 ส่วน xyz ดังนี้



โดยที่ $|xy| \leq m$ และ $|y| = k ; k \geq 1$

$$xyz = 0^{m-k} 1^k 0^m 0^{2m}$$

- (4) เลือกทำซ้ำสตริง xy^iz โดยกำหนดให้ $i = 0$

$$\begin{aligned} xy^0z &= 0^{m-k} (0^k)^0 1^m 0^{2m} \\ &= 0^{m-k} 1^m 0^{2m} \quad \text{โดยที่ } k \geq 1 \end{aligned}$$

เนื่องจากผลที่ได้จากการทำซ้ำไม่อยู่ในภาษา L การพิสูจน์นี้จึงขัดแย้งกับปั๊มปิงเลมมา ทำให้สมมุติฐานที่ตั้งไว้ว่า L เป็นภาษาเรกุลาร์ไม่เป็นความจริง ดังนั้นสรุปได้ว่า L ไม่ใช่ภาษาเรกุลาร์

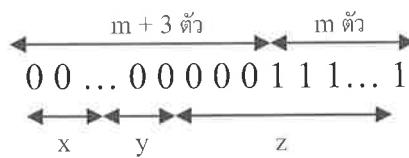
ตัวอย่างที่ 4.8 กำหนดให้ $L = \{v \in \{0, 1\}^* : \text{จำนวนของเลข } 0 \text{ มากกว่าจำนวนของเลข } 1 \text{ อุ่ 3 ตัว}\}$ จงพิสูจน์ว่าภาษา L ไม่ใช่ภาษาเรกุลาร์

พิสูจน์ : โดยใช้ปั๊มปิงเลมมา

- (1) สมมุติว่า L เป็นภาษาเรกุลาร์ โดยมีจำนวนสถานะของไฟฟ้าในต่อโถมata ที่รองรับภาษา L เท่ากับ m
- (2) เลือกสตริง $s \in L$ และ $|s| \geq m$

$$s = 0^{m+3} 1^m$$

- (3) แบ่งสตริง s ออกเป็น 3 ส่วน xyz ดังนี้



โดยที่ $|xy| \leq m$ และ $|y| = k ; k \geq 1$

$$xyz = 0^{m+3-k} 0^k 1^m$$

(4) เลือกทำคำสัตว์ xy^iz โดยกำหนดให้ $i = 0$

$$\begin{aligned} xy^0z &= a^{m+3-k} (a^k)^0 a^m \\ &= a^{m+3-k} a^m \quad \text{โดยที่ } k \geq 1 \end{aligned}$$

เนื่องจากผลที่ได้จากการทำคำไม่อุปในภาษา L จึงขัดแย้งกับปีมปิงเลมนما ทำให้สมมุติฐานที่ตั้งไว้ว่า L เป็นภาษาเรกูลาร์ไม่เป็นความจริง ดังนั้นสูปได้ว่า L ไม่ใช่ภาษาเรกูลาร์

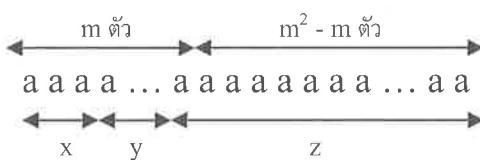
ตัวอย่างที่ 4.9 กำหนดให้ $L = \{ a^{n^2} : n \geq 0 \}$ จะพิสูจน์ว่าภาษา L ไม่ใช่ภาษาเรกูลาร์

พิสูจน์ : โดยใช้มปิงเลมนما

- (1) สมมุติว่า L เป็นภาษาเรกูลาร์ โดยมีจำนวนสถานะของไฟโนต์อโตมาตาที่รองรับภาษา L เท่ากับ m
- (2) เลือกสตริง $s \in L$ และ $|s| \geq m$

$$s = a^{m^2}$$

(3) แบ่งสตริง s ออกเป็น 3 ส่วน xyz ดังนี้



โดยที่ $|xy| \leq m$ และ $|y| = k ; k \geq 1$

$$xyz = a^{m-k} a^k a^{m^2-m}$$

(4) เลือกทำคำสัตว์ xy^iz โดยกำหนดให้ $i = 0$

$$\begin{aligned} xy^0z &= a^{m-k} a^k a^{m^2-m} \\ &= a^{m^2-k} \quad \text{โดยที่ } k \geq 1 \end{aligned}$$

xy^0z จะเป็นสมาชิกของ L ได้ก็ต่อเมื่อจะต้องมีจำนวนเต็มที่ยกกำลังสองแล้วมีค่าเท่ากับ $m^2 - k$ แต่เนื่องจาก $m^2 - k \neq n^2$ (ไม่มีจำนวน n ใด ๆ ที่ยกกำลังสองแล้วมีค่าเท่ากับ $m^2 - k$) สำหรับทุก ๆ ค่าของ m และ $k < m$ และสำหรับค่า n ใด ๆ ซึ่งความสามารถพิสูจน์ได้ดังนี้

จำนวนที่ยกกำลังสองแล้วมีค่าใกล้เคียงกับ $m^2 - k$ ที่สุดคือ $(m - 1)$ ดังนั้นจึงต้องหาความสัมพันธ์ระหว่าง $m^2 - k$ และ $(m - 1)^2$ ว่าสามารถมีค่าเท่ากันได้หรือไม่

$$m^2 - k = (m - 1)^2$$

$$m^2 - k = m^2 - 2m + 1$$

เนื่องจาก k มีค่าสูงสุดได้เพียงแค่ m ดังนั้นจึงทำให้ $m^2 - k$ มีค่ามากกว่า $m^2 - 2m + 1$

$$m^2 - k > m^2 - 2m + 1$$

นั่นคือ $m^2 - k > (m - 1)^2$ ซึ่งมีผลทำให้ $m^2 - k \neq n^2$

ดังนั้นสรุปได้ว่า $a^{m^2-k} \notin L$

เนื่องจากผลที่ได้จากการทำข้อไม่อุปในภาษา L จึงขัดแย้งกับปั้มนปิงเลมมา ทำให้สมมุติฐานที่ตั้งไว้ว่า L เป็นภาษาเรกูลาร์ไม่เป็นความจริง ดังนั้นสรุปได้ว่า L ไม่ใช่ภาษาเรกูลาร์

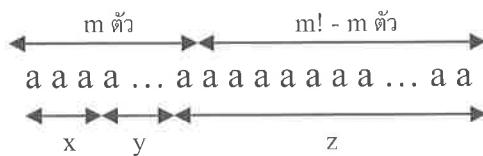
ตัวอย่างที่ 4.10 กำหนดให้ $L = \{ a^{n!} : n \geq 1 \}$ จงพิสูจน์ว่าภาษา L ไม่ใช่ภาษาเรกูลาร์

พิสูจน์ : โดยใช้ปั้มนปิงเลมมา

- (1) สมมุติว่า L เป็นภาษาเรกูลาร์ โดยมีจำนวนสถานะของไฟฟ้าในตัวอัตโนมัติร่องรับภาษา L เพfigcaption m
- (2) เลือกสตริง $s \in L$ และ $|s| \geq m$

$$s = a^{m!}$$

- (3) แบ่งสตริง s ออกเป็น 3 ส่วน xyz ดังนี้



โดยที่ $|xy| \leq m$ และ $|y| = k ; k \geq 1$

$$xyz = a^{m-k} a^k a^{m!-m}$$

- (4) เลือกทำข้อสตริง xy^iz โดยกำหนดให้ $i = 2$

$$xy^2z = a^{m-k} a^{2k} a^{m!-m}$$

$$= a^{m+k} a^{m!-m}$$

$$= a^{m+k} \quad \text{โดยที่ } k \geq 1$$

การจะพิสูจน์ว่า $xy^2z \notin L$ ได้นั้น จะต้องมั่นใจได้ว่า

$$a^{m+k} \neq a^{p!} \quad \text{สำหรับ } 1 \leq k \leq m \text{ และค่า } p \text{ ใด ๆ}$$

นั่นคือ

$$(m! + k) \neq p! \quad \text{สำหรับ } 1 \leq k \leq m \text{ และค่า } p \text{ ใด ๆ}$$

เราสามารถพิสูจน์ความไม่เท่ากันดังกล่าวได้ดังนี้

$$\begin{aligned} (m! + k) &\leq (m! + m) \quad \text{สำหรับ } 1 \leq k \leq m \\ &\leq (m! + m!) \\ &< (m!.m + m!) \\ &< m!(m + 1) \\ &< (m + 1)! \end{aligned}$$

โปรดสังเกตว่า $(m! + k)$ จะมีค่าเท่ากับ $p!$ ได้นั่น ค่า p ควรจะมีค่าเป็น $m + 1$ หรือ $m + 2$ หรือ $m + n$ แต่ผลจากอสมการที่ได้พิสูจน์มาสรุปได้ว่า สำหรับ $1 \leq k \leq m$ ค่าของ $(m! + k)$ จะมีค่าน้อยกว่า $(m + 1)!$ เสมอ ดังนั้นจึงไม่มีทางที่จะหาค่า p ที่ทำให้ $(m! + k) = p!$ เป็นจริงได้เลย ด้วยเหตุผลนี้จึงทำให้ $xy^2z \notin L$ และเป็นผลให้ สมมุติฐานที่ตั้งไว้ตอนต้นนี้เป็นเท็จ ดังนั้นภาษา L จึงไม่ใช่ภาษาเรกูลาร์

4.5 ไฮโอมอร์ฟิซึมฟังก์ชัน

ในการพิสูจน์ความไม่เป็นเรกูลาร์ของภาษา นอกจากรู้ใช้หลักการของปั๊มปิงเลมมาแล้ว ยังสามารถพิสูจน์ ความไม่เป็นเรกูลาร์ของภาษาได้อีกวิธีหนึ่งคือ ไฮโอมอร์ฟิซึมฟังก์ชัน สำหรับนิยามของไฮโอมอร์ฟิซึมฟังก์ชันมีดังนี้

ไฮโอมอร์ฟิซึมฟังก์ชัน (Homomorphism Function) คือ ความสัมพันธ์จากตัวอักษร ๑ ตัวไปยังสตริง เช่น

$$h(a) = bb ; \text{ ตัวอักษร } a \text{ จำนวน } 1 \text{ ตัว สามารถแทนด้วยสตริง } bb$$

กำหนดให้ Σ_1 และ Σ_2 แทนเซตของตัวอักษรใด ๆ ไฮโอมอร์ฟิซึมฟังก์ชันสามารถแทนได้ด้วย ความสัมพันธ์ $\Sigma_1 \rightarrow \Sigma_2^*$ จากนิยามดังกล่าว ไฮโอมอร์ฟิซึมฟังก์ชันจึงหมายถึง การแทนตัวอักษรตัวหนึ่งด้วยสตริง นั่นเอง สำหรับสตริง s ซึ่งประกอบด้วยตัวอักษรหลายตัวมาเขียนต่อ กัน เราสามารถหาไฮโอมอร์ฟิซึมของสตริง s ได้โดย

$$s = a_1 a_2 a_3 \dots a_n$$

$$h(s) = h(a_1) h(a_2) h(a_3) \dots h(a_n)$$

หาก L เป็นภาษาใด ๆ จะเรียก $h(L)$ ว่า ไฮโอมอร์ฟิกอิมเมจ (Homomorphic Image) ของภาษา L เช่น

$$L = \{s_1, s_2, \dots, s_n\}$$

$$h(L) = \{h(s_1), h(s_2), \dots, h(s_n)\}$$

ตัวอย่างที่ 4.11 กำหนดให้ $\Sigma_1 = \{0, 1\}$, $\Sigma_2 = \{a, b\}$

$$L = \{001, 1011, 100\}$$

$$\text{และ } h(0) = aa$$

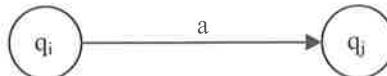
$$h(1) = ab$$

จงหา $h(L)$?

$$\begin{aligned} h(L) &= \{h(001), h(1011), h(100)\} \\ &= \{aaaaab, abaaabab, abaaaa\} \end{aligned}$$

ทฤษฎี 4.5.1 : ถ้า L เป็นภาษาเรกูลาร์ ไฮโอมอร์ฟิกอิมเมจของ L จะต้องเป็นภาษาเรกูลาร์ด้วย นั่นคือ ภาษาเรกูลาร์มีคุณสมบัติปิดภายใต้การกระทำไฮโอมอร์ฟิซึม

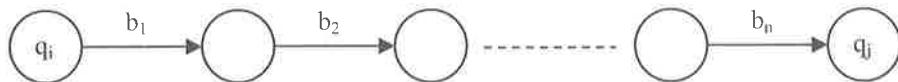
พิสูจน์ : ถ้า L เป็นภาษาเรกูลาร์ จะต้องสร้างไฟโนน์ต่อโตมาตา M ขึ้นมารองรับ L ได้ สำหรับทราบว่า L ได้ ๗ ในไฟโนน์ต่อโตมาตา M



กำหนดให้ไฮโอมอร์ฟิซึมพังก์ชันของตัวอักษร a คือ

$$h(a) = b_1 b_2 \dots b_n$$

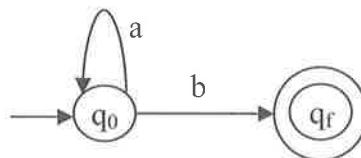
สามารถสร้างทรานซิชันรองรับ $h(a)$ ได้โดยสร้างสถานะใหม่ $n - 1$ สถานะ อยู่ระหว่างสถานะ q_i และ q_j เพื่อรับอินพุต b_1, b_2, \dots, b_n ดังนี้



เนื่องจากสามารถสร้างทรานซิชันให้กับไฮโอมอร์ฟิซึมของแต่ละตัวอักษรที่ปรากฏอยู่ในทรานซิชันเดิมของไฟในต่อๆ กันมาได้ ดังนั้นจึงสามารถสร้างไฟในต่อๆ กันมาให้กับ $h(a)$ ได้ ๆ ได้ ซึ่งส่งผลให้ไฮโอมอร์ฟิกอิมเมจของ L เป็นภาษาเรกูลาร์

ตัวอย่างที่ 4.12 กำหนดให้ $L = \{a^n b : n \geq 0\}$ จะพิสูจน์ว่า $h(L)$ เป็นภาษาเรกูลาร์ กำหนดให้ $h(a) = 00, h(b) = 11$

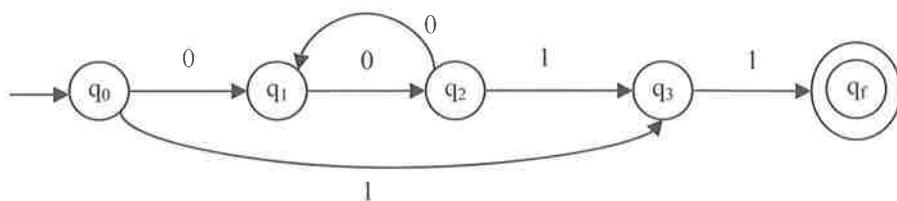
พิสูจน์ : เนื่องจาก L เป็นภาษาเรกูลาร์ ดังไฟในต่อๆ กันมาต่อไปนี้



รูปที่ 4.14 ไฟในต่อๆ กันมาสำหรับภาษา L

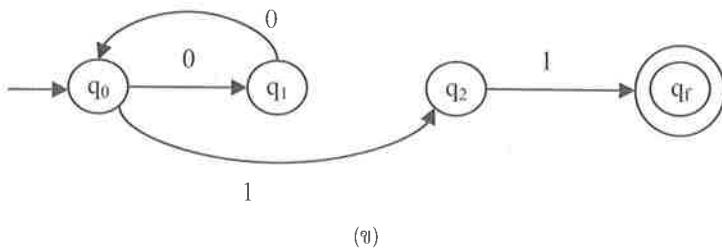
$$h(L) = \{(00)^n 11 : n \geq 0\}$$

$h(L)$ เป็นภาษาเรกูลาร์ เนื่องจากสามารถสร้างไฟในต่อๆ กันมาได้ดังนี้



(ก)

รูปที่ 4.15 ไฟในต่อๆ กันมาสำหรับภาษา $h(L)$



รูปที่ 4.15 (ต่อ)

ตัวอย่างที่ 4.13 จงพิสูจน์ว่าภาษา L ไม่เป็นภาษาเรกูลาร์

$$L = \{a^{n/3}b^{n/3} : n \geq 0 \text{ และ } n \text{ หารด้วย } 3 \text{ ลงตัว}\}$$

พิสูจน์ : สำหรับตัวอย่างนี้ นอกจากจะใช้ปั้มปิงเลมมาว่า พิสูจน์ความไม่เป็นภาษาเรกูลาร์ของภาษา L ได้แล้ว ยังสามารถใช้คุณสมบัติปิดภายใต้ไฮโอมอร์ฟิซึมพิสูจน์ได้อีกด้วย

กำหนดให้ $h(a) = aaa$,

$$h(b) = bbb$$

จากนั้นหาไฮโอมอร์ฟิกอิมเมจของ L

$$h(L) = \{a^n b^n : n \geq 0\}$$

เนื่องจากทราบมาก่อนหน้านี้โดยปั้มปิงเลมมาว่า $\{a^n b^n : n \geq 0\}$ ไม่ใช่ภาษาเรกูลาร์ ดังนั้นภาษา L จึงไม่ใช่ภาษาเรกูลาร์ด้วย เนื่องจากภาษาเรกูลาร์มีคุณสมบัติปิดภายใต้การกระทำไฮโอมอร์ฟิซึม

ผู้อ่านหลายท่านคงสงสัยว่าเหตุใดเราจึงไม่พิสูจน์ต่อโดยใช้ปั้มปิงเลมมาว่า $\{a^n b^n : n \geq 0\}$ ไม่ใช่ภาษาเรกูลาร์ แต่กลับไปอ้างถึงผลการพิสูจน์ก่อนหน้านี้ เหตุผลก็คือว่าในศาสตร์ของทฤษฎีการคำนวณ เป็นที่ทราบกันดีว่า ภาษา $\{a^n b^n : n \geq 0\}$ ไม่เป็นภาษาเรกูลาร์ ดังนั้นหากสามารถแปลงภาษาใด ๆ โดยใช้ไฮโอมอร์ฟิซึมแล้วมีลักษณะของภาษาเหมือนกับ $\{a^n b^n : n \geq 0\}$ ได้ ให้สรุปได้ทันทีว่าภาษาที่กำลังพิจารณาอยู่นั้นไม่เป็นภาษาเรกูลาร์ โดยไม่ต้องพิสูจน์ $a^n b^n$ ต่อ ดังนั้น $a^n b^n$ จึงถือว่าเป็นตัวแทนของภาษาที่ไม่ใช่เรกูลาร์ซึ่งเราควรจะดึงไว้ใช้ในการพิสูจน์ต่อไป

อินเวอร์สไฮโอมอร์ฟิซึม (Inverse Homomorphism)

อินเวอร์สไฮโอมอร์ฟิซึมเป็นความสัมพันธ์ที่เป็นส่วนกลับของไฮโอมอร์ฟิซึม

หาก $h : \Sigma_1 \rightarrow \Sigma_2^*$ และ w คือสตริง

$$h^{-1}(w) = \{x \mid h(x) = w\}$$

และสำหรับภาษา L ใด ๆ $h^{-1}(L)$ หรืออินเวอร์สไฮโอมอร์ฟิกอิมเมจของ L ถูกนิยามโดย

$$h^{-1}(L) = \{y \mid h(y) \in L\}$$

ตัวอย่างที่ 4.14 กำหนดให้ $h(a) = 01$

$$h(b) = 11$$

$$L = (01 + 11)^* 0111$$

$$\therefore h^{-1}(L) = (a + b)^* ab$$

ทฤษฎี 4.5.2 : ถ้า L เป็นภาษาเรกูลาร์ $h^{-1}(L)$ จะต้องเป็นภาษาเรกูลาร์ด้วย

พิสูจน์ : วิธีการพิสูจน์จะส่วนทางกับการพิสูจน์โดยโมมอร์ฟิกอิมเมจ จึงขอละการพิสูจน์ไว้เป็นแบบฝึกหัดสำหรับผู้อ่าน

ตัวอย่างที่ 4.15 จงพิสูจน์ว่าภาษา L ไม่เป็นภาษาเรกูลาร์

$$L = \{a^{4n}b^{2n} : n \geq 0\}$$

พิสูจน์ : สำหรับตัวอย่างนี้ จะใช้คุณสมบัติปิดภายใต้อินเวิร์สโดยโมมอร์ฟิกชื่อพิสูจน์ดังนี้

กำหนดให้ $h(a) = aaaa$,

$$h(b) = bb$$

จากนั้นหาโดยโมมอร์ฟิกอิมเมจของ L

$$h^{-1}(L) = \{a^n b^n : n \geq 0\}$$

เนื่องจากทราบว่า $\{a^n b^n : n \geq 0\}$ ไม่ใช่ภาษาเรกูลาร์ ดังนั้นภาษา L จึงไม่ใช่ภาษาเรกูลาร์ด้วย เพราะภาษาเรกูลาร์มีคุณสมบัติปิดภายใต้การกระทำอินเวิร์สโดยโมมอร์ฟิกชื่อ

4.6 พลหารทางขวา

กำหนดให้ L_1 และ L_2 เป็นภาษาใด ๆ พลหารทางขวา (Right Quotient) L_1 ด้วย L_2 ถูกนิยามโดย

$$L_1/L_2 = \{u : \text{มีบางสตริง } v \text{ ซึ่ง } v \in L_2 \text{ และทำให้ } uv \in L_1\}$$

นั่นคือสามารถทุก ๆ ตัวใน L_1/L_2 เมื่อนำมาเชื่อมต่อกับสตริง v บางตัว ซึ่ง $v \in L_2$ ผลที่ได้จะต้องเป็นสมาชิกใน L_1

ตัวอย่างที่ 4.16 กำหนดให้ภาษา L_1 และ L_2 นิยามดังนี้

$$L_1 = \{aab, abab\}$$

$$L_2 = \{b, ab\}$$

จงหาผลหารทางขวาของ L_1/L_2

$$L_1/L_2 = \{a, aa, ab, aba\}$$

ตัวอย่างที่ 4.17 กำหนดให้ภาษา L_1 และ L_2 นิยามดังนี้

$$L_1 = \{a^n b a^m : n, m \geq 0\}$$

$$L_2 = \{ba^n b : n \geq 0\}$$

จงหาผลหารทางขวาของ L_1/L_2

$$L_1/L_2 = \emptyset$$

เนื่องจากสตริง v ใน L_2 จะต้องมี b อยู่ 2 ตัว ดังนั้นจึงเป็นไปไม่ได้เลยที่สตริง v ใน L_2 จะมีสตริง a มาอยู่ข้างหน้าแล้วทำให้ $uv \in L_1$ คำตอบของตัวอย่างนี้จึงเป็นเท็จว่า

ตัวอย่างที่ 4.18 กำหนดให้ภาษา L_1 และ L_2 ถูกนิยามดังนี้

$$L_1 = \{a^n b a^m : n, m \geq 0\}$$

$$L_2 = \{a^n b : n \geq 0\}$$

จงหาผลหารทางขวาที่ได้จากการหาร L_1 ด้วย L_2 (L_1/L_2)

$$L_1/L_2 = \{a^n : n \geq 0\}$$

ทฤษฎี 4.6.1 : ถ้า L_1 และ L_2 เป็นภาษาเรกูลาร์ L_1/L_2 จะต้องเป็นภาษาเรกูลาร์ด้วย

พิสูจน์ : จะพิสูจน์โดยสร้างอัลกอริทึมในการหาไฟฟ์ในต่อไปนี้

กำหนดให้ L_1 ถูกแทนด้วยไฟฟ์ในต่อไปนี้ $M = (Q, \Sigma, \delta, q_0, F)$

$$L_1/L_2 \text{ ถูกแทนด้วยไฟฟ์ในต่อไปนี้ } M_r = (Q, \Sigma, \delta, q_0, F_r)$$

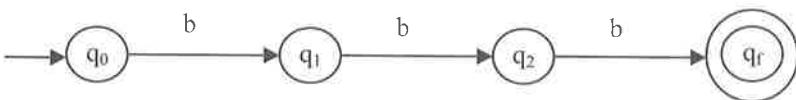
สำหรับการหาเซตของสถานะสุดท้าย (F_r) นั้น เราจะต้องหาดูว่ามีสตริง y ซึ่ง $y \in L_2$ และสามารถทำให้ $\delta^*(q_i, y) = q_f$ โดยที่ $q_i \in Q$ และ $q_f \in F$ ได้หรือไม่ วิธีการหาสตริง y สามารถทำได้โดยการทำอินเตอร์เซกชัน (Intersection) ระหว่าง L_2 และ $L(M_r)$ ถ้าสามารถหาเล้นทางจากสถานะเริ่มต้น q_i ไปยังสถานะสุดท้าย q_f ของ M ได้ ก็แสดงว่า $L_2 \cap L(M_r) \neq \emptyset$ จากนั้นให้นำ q_i ไปปั๊วีในเซต F_r และทำซ้ำในแต่ละสถานะ q_i อีก ๆ ที่สามารถเดินทางมายังสถานะสุดท้ายของ M ได้

ตัวอย่างที่ 4.19 กำหนดให้ภาษา L_1 นิยามดังนี้

$$L_1 = \{a^m b^{m+3} : m \geq 0\}$$

จงพิสูจน์ว่าภาษา L_1 ไม่ใช่ภาษาเรกูลาร์โดยวิธีผลหารทางขวา (Right Quotient)

กำหนด $L_2 = \{bbb\}$ ซึ่งเป็นภาษาเรกูลาร์เนื่องจากสามารถสร้างไฟฟ์ในต่อไปนี้ได้



รูปที่ 4.16 ไฟฟ์ในต่อไปนี้สำหรับภาษา L_2

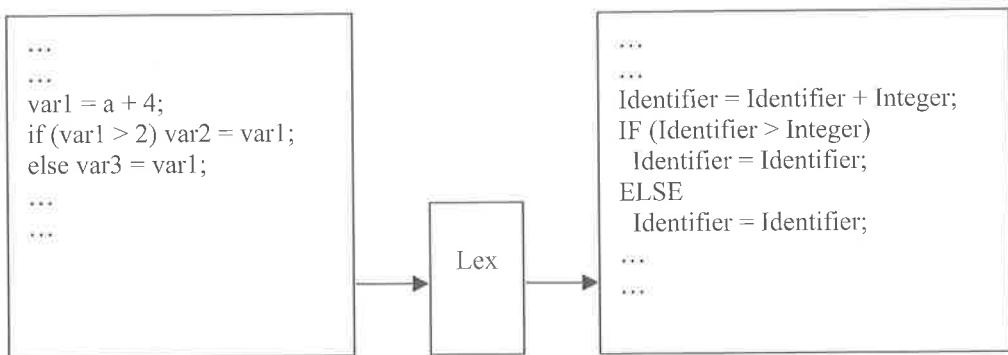
ผลหารทางขวา L_1/L_2 จะได้ว่า

$$L_1/L_2 = \{a^m b^m : m \geq 0\}$$

เนื่องจากทราบมาก่อนแล้วว่า $\{a^m b^m : m \geq 0\}$ ไม่ใช่ภาษาเรกูลาร์ ดังนั้นจึงสรุปได้ว่าภาษา L_1 จะต้องไม่ใช่ภาษาเรกูลาร์ เนื่องจากถ้า L_1 เป็นภาษาเรกูลาร์แล้ว L_1/L_2 ก็จะต้องเป็นภาษาเรกูลาร์ด้วย

4.7 การประยุกต์ใช้งานภาษาเรกูลาร์และไฟไนต์อัตโนมາตาในตัวแปลงภาษา

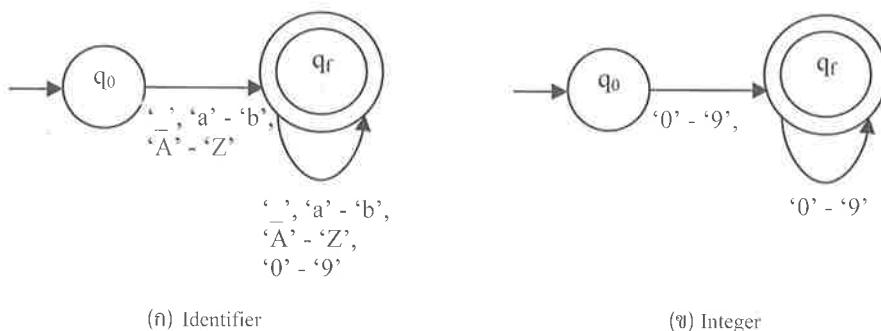
ผู้อ่านหลายท่านคงสังสัยว่าภาษาเรกูลาร์และไฟไนต์อัตโนมາตาที่ได้ศึกษามาทั้งหมดมีประโยชน์อย่างไรบ้าง กับการเขียนโปรแกรม ซึ่งที่จริงแล้วภาษาเรกูลาร์มีความสำคัญอย่างยิ่งกับการสร้างตัวแปลงภาษาสำหรับตรวจสอบความถูกต้องของชอร์สโปรแกรม แต่ละคำสั่งภายในชอร์สโปรแกรมจะถูกมองเป็นสตริงที่อยู่ภายใต้ภาษาคอมพิวเตอร์ ซึ่งสตริงเหล่านี้จะถูกป้อนให้ตัวแปลงภาษาเพื่อแปลงให้กลายเป็นส่วนประกอบย่อย ๆ ที่อิสระจากกันเรียกว่า โทเคน (Token) ส่วนที่ทำหน้าที่แปลงชอร์สโปรแกรมให้กลายเป็นโทเคนถูกเรียกว่า Lex ซึ่งเป็นคำย่อมาจากคำว่า Lexical Analyser ตัวอย่างการทำงานของ Lex สามารถอธิบายได้ดังนี้



รูปที่ 4.17 การทำงานของ Lex

ตัวแปร var1 ถูกแปลงให้เป็นโทเคน Identifier โดยไม่สนใจว่าชื่อตัวแปรเป็นอะไร ขอเพียงให้ตั้งชื่อตัวแปรให้ถูกต้องตามหลักการตั้งชื่อตัวแปรเท่านั้น เช่นเดียวกับตัวเลข 4 ซึ่งถูกแปลงให้เป็นโทเคน Integer โดยไม่สนใจว่าค่าจะเป็นเท่าใด ขอเพียงให้เป็นค่าตัวเลขที่ถูกต้องตามหลักการสร้างตัวเลขจำนวนเต็มเท่านั้น

สำหรับการตรวจสอบว่าโทเคนต่าง ๆ เขียนถูกต้องตามหลักการหรือไม่ เราจะต้องอาศัยการสร้างไฟไนต์อัตโนมາต้าขึ้นมารองรับ นั่นคือ เราจะต้องสร้างไฟไนต์อัตโนมາเพื่อรับโทเคน Identifier, Integer, Keyword-IF, Keyword-Else, วงเล็บเปิด, วงเล็บปิด, เครื่องหมายมากกว่า, เครื่องหมายน้อยกว่า และเครื่องหมายเท่ากับ และเครื่องหมายเชมิโคลอน (;) ตัวอย่างของ NFA สำหรับโทเคน Identifier และ Integer คือ



รูปที่ 4.18 ไฟนิตออโตมาตาสำหรับโดย Identifier และ Integer

จากนั้นป้อนสตริงเข้าไปใน NFA เหล่านี้ หากสตริงถูกยอมรับโดย NFA ได้ ก็ถือว่าโทเค็นของ NFA นั้นใช้แทนสตริงได้ ขั้นตอนในการแปลงชอร์สโปรแกรมให้เป็นโทเค็นเป็นเพียงขั้นตอนแรกในการแปลงภาษา ขั้นตอนต่อไปเป็นการตรวจสอบความถูกต้องของไวยากรณ์ซึ่งจะต้องใช้ความรู้เรื่องอื่น ๆ ที่เราจะได้ศึกษาต่อจากนี้ไป

แบบฝึกหัด

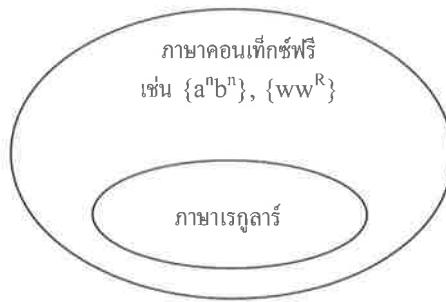
1. การพิสูจน์ความเป็นเรกุลาร์ของภาษาสามารถทำได้กี่วิธี อะไรบ้าง
2. การพิสูจน์ความไม่เป็นเรกุลาร์ของภาษาสามารถทำได้กี่วิธี อะไรบ้าง
จะพิสูจน์ว่าภาษาดังต่อไปนี้ไม่ใช่ภาษาเรกุลาร์
3. $L = \{a^i b^j : j = i \text{ หรือ } j = 2i\}$
4. $L = \{0^p 1^q : p \neq q\}$
5. $L = \{a^n b a^{2n} : n \geq 0\}$
6. $L = \{w \in \{a, b\}^* : \text{จำนวนตัวอักษร } a \text{ ภายในสตริง } w \text{ น้อยกว่าสองเท่าของจำนวนตัวอักษร } b \text{ ภายในสตริง } w\}$
7. $L = \{b^{n,n!} : n \geq 1\}$
8. $L = \{w \in \{a, b\}^* : \text{ไม่มี prefix ใด ๆ ของ } w \text{ ที่มีจำนวนตัวอักษร } b \text{ มากกว่า } a\}$
9. $L = \{0^i 1^j : j \text{ เป็นจำนวนเต่าของ } i\}$
10. $L = \{a^i b^j a^k : k > i + j\}$

ไวยากรณ์คอนเท็กซ์ฟรี

และภาษาคอนเท็กซ์ฟรี

ในโลกของคอมพิวเตอร์ เราสามารถเขียนโปรแกรมที่มีความ слับซับซ้อนของไวยากรณ์ภาษา เช่น สามารถเขียน for loop ซ้อนกันหลาย ๆ ชั้น หรือการมี if - else ชั้น if - else เป็นต้น การเขียนโปรแกรมในลักษณะนี้ ภาษาคอมพิวเตอร์จะต้องมีความสามารถในการจำได้ว่ามีจำนวนลูปซ้อนกันอยู่กี่ชั้น เพื่อจะได้บังคับให้โปรแกรมเมอร์ ใส่สัญลักษณ์ปิดท้ายลูป เช่น end, } ให้มีจำนวนครบตามจำนวนสัญลักษณ์ปิดลูปที่ซ้อนกันอยู่

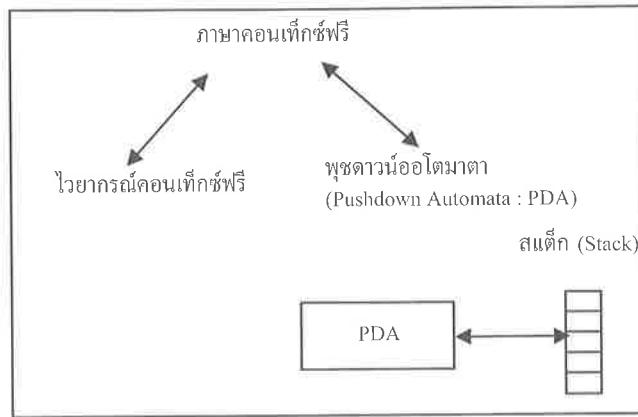
จากลักษณะดังกล่าว ความสามารถของภาษาเรกุลาร์ดูเหมือนจะไม่เพียงพอที่จะจัดการกับเรื่องเหล่านี้ได้ เนื่องจากไฟโนต์อโตมาต้ามีความสามารถจำที่จำกัด (มีจำนวนสถานะที่จำกัด) แต่โปรแกรมเมอร์สามารถทำลูปซ้อนลูปได้ อย่างไม่จำกัด ภาษาหนึ่งที่มีความสามารถมากกว่าภาษาเรกุลาร์และสามารถจัดการกับความซับซ้อนดังกล่าวได้ คือ ภาษาคอนเท็กซ์ฟรี (Context-free Language) เชตของภาษาคอนเท็กซ์ฟรีเป็นเซตที่ใหญ่กว่าและครอบคลุมเชต ของภาษาเรกุลาร์ ดังนั้นไวยากรณ์ของภาษาคอนเท็กซ์ฟรีจึงมีความซับซ้อนกว่าไวยากรณ์ของภาษาเรกุลาร์



รูปที่ 5.1 ความสัมพันธ์ระหว่างภาษาเรกุลาร์และภาษาคอนเท็กซ์ฟรี

จากคำอธิบายในส่วนแรกของหนังสือเล่มนี้ ผู้อ่านได้ทราบมาแล้วว่าไฟโนต์อโตมาต้าเป็นเครื่องมือของรับภาษาเรกุลาร์ สำหรับภาษาคอนเท็กซ์ฟรีนั้นก็มีเครื่องมือรองรับภาษาที่เรียกว่า พุชดาวน์อโตมาต้า (Pushdown Automata : PDA) พุชดาวน์อโตมาต้ามีความสามารถที่เหนือกว่าไฟโนต์อโตมาต้า เนื่องจากมีสแต็ก (Stack) ซึ่งทำหน้าที่เป็นหน่วยความจำของอโตมาต้าจึงทำให้สามารถจัดจำตัวอักษรได้ไม่จำกัด เราจะศึกษาเรื่องพุชดาวน์ อโตมาต้าในบทต่อไป สำหรับในบทนี้เราจะเป็นต้องเรียนรู้ภาษาคอนเท็กซ์ฟรีและไวยากรณ์ที่เกี่ยวข้องก่อน

ความสัมพันธ์ระหว่างภาษาค่อนเท็กซ์ฟรี ไวยากรณ์ค่อนเท็กซ์ฟรี และพุชดาวน์อโตมาตา มีลักษณะ เช่นเดียวกับความสัมพันธ์ที่เกิดขึ้นระหว่างภาษาเรกูลาร์ ไวยากรณ์เรกูลาร์ และไฟฟ์อโนต์อโตมาตาในบทที่ผ่านมา ดัง แสดงในรูป



รูปที่ 5.2 ความสัมพันธ์ระหว่างภาษาค่อนเท็กซ์ฟรี ไวยากรณ์ค่อนเท็กซ์ฟรี และ PDA

เราใช้พุชดาวน์อโตมาตาเป็นเครื่องมือตรวจสอบสตริงที่เป็นสมาชิกในภาษาค่อนเท็กซ์ฟรี และสร้างไวยากรณ์ค่อนเท็กซ์ฟรีเพื่อเป็นกฎเกณฑ์ในการสร้างสตริงในภาษาค่อนเท็กซ์ฟรี โดยความสัมพันธ์ของคำทั้งสามคำ จะเป็นความสัมพันธ์แบบสองทาง นั่นคือ เมื่อเรามีลิ่งหนึ่งก็สามารถสร้างอีกลิ่งหนึ่งที่สัมพันธ์กันได้

5.1 ไวยากรณ์ค่อนเท็กซ์ฟรี

นิยาม 5.1.1 : ไวยากรณ์ค่อนเท็กซ์ฟรี (Context-free Grammar) หรือที่นิยมเรียกว่า CFG นิยามโดย

$$G = (V, T, S, P)$$

โดยที่ V คือ เซตของตัวแปร (Variable)

T คือ เซตของสัญลักษณ์เทอร์มินอล (Terminal Symbol)

นอกจานี้ $V \cap T = \emptyset$

S คือ สัญลักษณ์เริ่มต้น (Start Symbol)

P คือ เซตของโปรดักชัน (Production) ซึ่งมีรูปแบบดังนี้

$$A \rightarrow \alpha$$

โดยที่ A คือ ตัวแปร ($A \in V$) และ α คือ สตริงของ $(V \cup T)^*$

จะเห็นว่าความแตกต่างระหว่างไวยากรณ์เรกูลาร์และไวยากรณ์ค่อนเท็กซ์ฟรีคือ ฝั่งขวาของโปรดักชัน ซึ่งหากเป็นไวยากรณ์เรกูลาร์แล้วฝั่งขวาของโปรดักชันจะถูกบังคับให้มีรูปแบบเป็น Right-linear หรือ Left-linear

เท่านั้น ในขณะที่ไวยากรณ์ค่อนเท็กซ์ฟรีให้ความเป็นอิสระกับฝั่งขามีของโปรดักชันมากกว่า โดยจะมีลักษณะอย่างไรก็ได้ แต่ทางซ้ายมือของโปรดักชันต้องมีตัวแปรเพียงตัวเดียวเท่านั้น ตัวอย่างไวยากรณ์ค่อนเท็กซ์ฟรี เช่น

$$G_1 = (\{S\}, \{a, b\}, S, P_1)$$

$$P_1 : S \longrightarrow aSb$$

$$S \longrightarrow \varepsilon$$

หรือ

$$G_2 = (\{S, A, B\}, \{a, b\}, S, P_2)$$

$$P_2 : S \longrightarrow aAB$$

$$A \longrightarrow bBb$$

$$B \longrightarrow A | \varepsilon$$

สำหรับภาษาที่ได้จากไวยากรณ์ค่อนเท็กซ์ฟรี หรือ $L(G)$ ถูกนิยามดังนี้

$$L(G) = \{w : S \xrightarrow{*} w, w \in T^*\}$$

นั่นคือ ภาษาซึ่งประกอบด้วยเซตของสตริง w โดยที่สตริง w เกิดจากกระบวนการได้มา (Derivation) ซึ่งเริ่มต้นจากสัญลักษณ์เริ่มต้น S นอกจากนี้ สตริง w จะต้องเกิดจากการทำซ้ำของเซตของสัญลักษณ์ท่อร์มินอล T

ภาษาค่อนเท็กซ์ฟรี (Context-free Languages)

นิยาม 5.1.2 : ภาษา L จะถูกเรียกว่าเป็นภาษาค่อนเท็กซ์ฟรีก็ต่อเมื่อมีไวยากรณ์ค่อนเท็กซ์ฟรี G ซึ่งสามารถทำให้ $L(G) = L$

ตัวอย่างที่ 5.1 จงพิสูจน์ว่าภาษา $L = \{a^n b^n : n \geq 0\}$ เป็นภาษาค่อนเท็กซ์ฟรี

พิสูจน์ : จะสร้างไวยากรณ์ G เพื่อทำให้ $L(G) = L$

$$G = (\{S\}, \{a, b\}, S, P)$$

$$P : S \longrightarrow aSb$$

$$S \longrightarrow \varepsilon$$

ไวยากรณ์ดังกล่าวเป็นไวยากรณ์ค่อนเท็กซ์ฟรีและสามารถสร้างภาษา L ได้ ดังนั้น $L(G) = L$ ซึ่งเป็นการพิสูจน์ว่าภาษา L เป็นภาษาค่อนเท็กซ์ฟรี

ต้นไม้แสดงการได้มา (Derivation Tree)

การแสดงการได้มาของสตริงโดยเริ่มต้นจากสัญลักษณ์เริ่มต้นของไวยากรณ์ บางครั้งอาจมีความซับซ้อน และยากต่อการเข้าใจ เช่น การสร้างสตริง $abbbbbbb$ ออกมาจากไวยากรณ์ G

$$S \rightarrow aAB$$

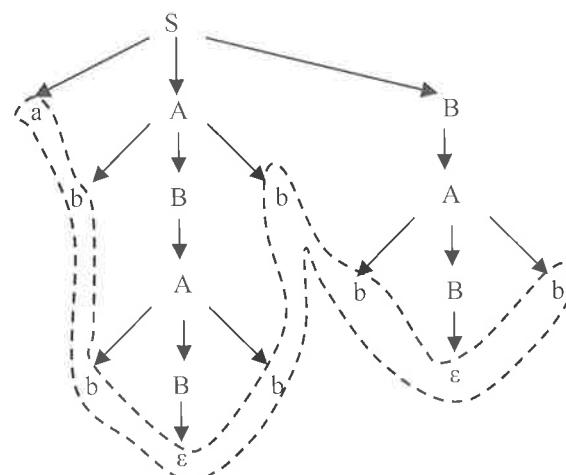
$$G : A \rightarrow bBb$$

$$B \rightarrow A|\varepsilon$$

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB$$

$$\Rightarrow abbbbB \Rightarrow abbbbA \Rightarrow abbbbbBb \Rightarrow abbbbbbb$$

หากสตริงยังมีความยาวเท่าได้ การแสดงการได้มาของสตริงโดยเริ่มต้นจากสัญลักษณ์ S ก็ยังมีความซับซ้อน และอาจทำให้เกิดความผิดพลาดขึ้นได้ ดังนั้นต้นไม้แสดงการได้มาจึงถูกนำเสนอเพื่ืออำนวยความสะดวกในการเข้าใจ กระบวนการได้มาของสตริงจากไวยากรณ์ โดยสามารถแสดงการได้มาของสตริง $abbbbbbb$ จากไวยากรณ์ G ได้ดังนี้



รูปที่ 5.3 ต้นไม้แสดงการได้มาของสตริง $abbbbbbb$

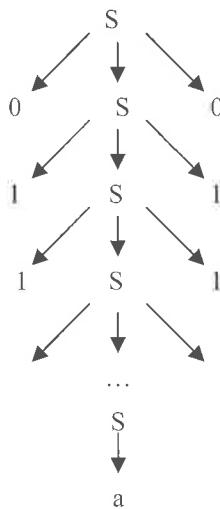
เมื่อนำสัญลักษณ์ท่อร่วมกลับซึ่งกันไปของต้นไม้มาต่อ กันตามเส้นประจากซ้ายไปขวา ผลที่ได้ก็คือสตริง $abbbbbbb$ ซึ่งถูกสร้างขึ้นมาจากไวยากรณ์ G นั้นเอง การใช้ต้นไม้แสดงการได้มาสามารถทำให้เราเข้าใจการได้มาของสตริงจากไวยากรณ์ได้อย่างรวดเร็ว นอกจากนี้ ต้นไม้แสดงการได้มาบังช่วยให้เห็นถึงรูปแบบของสตริงที่ถูกสร้างขึ้นมาจากไวยากรณ์อีกด้วย ทำให้ทราบลักษณะภาษาของไวยากรณ์ได้อย่างรวดเร็ว แต่อย่างไรก็ตาม ต้นไม้แสดงการได้มาไม่ได้แสดงลำดับของการได้มาเอาไว้ เช่น หากต้นไม้มีในระดับที่ 1 เราไม่มีทางทราบเลยว่า $A \rightarrow bBb$ หรือ $B \rightarrow A$ ที่เกิดขึ้นก่อน ดังนั้นหากต้องการสื่อความหมายโดยเน้นไปที่ลำดับการได้มาแล้ว การใช้ต้นไม้แสดงการได้มาอาจจะไม่ใช่ทางเลือกที่ดีนัก

ตัวอย่างที่ 5.2 จงหาภาษา L ที่ถูกสร้างขึ้นมาจากไวยากรณ์ G ดังต่อไปนี้

$$G = (\{S\}, \{0, 1, a\}, S, P)$$

$$P : \quad S \longrightarrow 0S0 \mid 1S1 \mid a$$

เมื่อนำโปรดักชัน P มาสร้างต้นไม้แสดงการได้มา ดังรูป



รูปที่ 5.4 ต้นไม้แสดงการได้มา

จะเห็นได้ว่าไวยากรณ์ G สร้างภาษาที่แบ่งสตริงออกเป็น 3 ส่วน โดยส่วนหน้ากับส่วนหลังจะเป็นลักษณะกลับด้านกัน (Reverse) ส่วนสตริงตรงกลางจะเป็นตัวอักษร a เพียงแค่ตัวเดียวเท่านั้น ดังนี้

$$L(G) = \{vav^R : v \in \{0, 1\}^*\}$$

ตัวอย่างที่ 5.3 จงพิสูจน์ว่าภาษา $L = \{a^n b^m : n > m\}$ เป็นภาษาค่อนเท็กซ์ฟรี

พิสูจน์ : โดยการสร้างไวยากรณ์ค่อนเท็กซ์ฟรี G สำหรับภาษา L

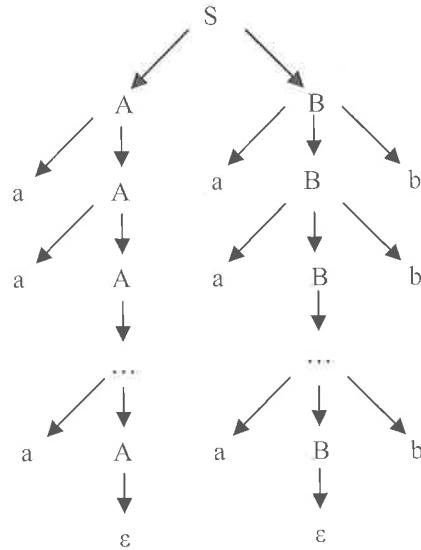
$$G = (\{S, A, B\}, \{a, b\}, S, P)$$

$$P : \quad S \longrightarrow AB$$

$$A \longrightarrow aA \mid a$$

$$B \longrightarrow aBb \mid \epsilon$$

เพื่อเป็นการตรวจสอบว่าไวยากรณ์ G สร้างภาษาเดียวกับภาษา L เราจะสร้างต้นไม้แสดงการได้มาเพื่อศึกษารูปแบบของสตริงที่ถูกสร้างออกมาจากไวยากรณ์ G นอกจากนี้เรายังสามารถสังเกตจากไวยากรณ์ได้ว่าตัวแปร A ไม่สามารถแปลงเป็นสตริงว่างได้ และจะต้องสร้างตัวอักษร a อย่างน้อย 1 ตัวเสมอ ส่วนตัวแปร B สามารถแปลงเป็นสตริงว่างได้ หรือจะเลือกสร้างตัวอักษร a และ b โดยมีจำนวนที่เท่ากัน (a นำหน้า b)



รูปที่ 5.5 ต้นไม้แสดงการได้มา

จากข้อสังเกตนี้และต้นไม้แสดงการได้มาที่สร้างขึ้น ทำให้เราทราบว่าไวยากรณ์ G สร้างสตริงที่มีจำนวนตัวอักษร a มากกว่าตัวอักษร b และตัวอักษร a จะต้องมาก่อนตัวอักษร b

เนื่องจาก $L(G) = L$ ดังนั้น L จึงเป็นภาษาค่อนเท็กซ์ฟรี

ตัวอย่างที่ 5.4 จงพิสูจน์ว่าภาษา $L = \{w \in \{a, b\}^* : \text{จำนวนตัวอักษร } a \text{ และ } b \text{ ภายในสตริง } w \text{ มีจำนวนเท่ากัน}\}$ เป็นภาษาค่อนเท็กซ์ฟรี

พิสูจน์ : โดยการสร้างไวยากรณ์ค่อนเท็กซ์ฟรี G สำหรับภาษา L

$$G = (\{S\}, \{a, b\}, S, P)$$

$$P : \quad S \longrightarrow abS \mid aSb \mid Sab \mid ab \mid ba \mid baS \mid bSa \mid Sba$$

ภาษาที่ถูกสร้างขึ้นมาจากไวยากรณ์ G จะมีจำนวนตัวอักษร a เท่ากับจำนวนตัวอักษร b เสมอ โดยตัวอักษร a จะมาก่อนตัวอักษร b หรือตัวอักษร b จะมาก่อนตัวอักษร a ก็ได้ ซึ่งตรงกับนิยามของภาษา L และทำให้

$$L(G) = L \quad \text{ดังนั้น } L \text{ จึงเป็นภาษาค่อนเท็กซ์ฟรี}$$

ตัวอย่างที่ 5.5 จงอธิบายภาษาค่อนเท็กซ์ฟรีซึ่งถูกสร้างขึ้นมาจากไวยากรณ์ค่อนเท็กซ์ฟรีดังต่อไปนี้

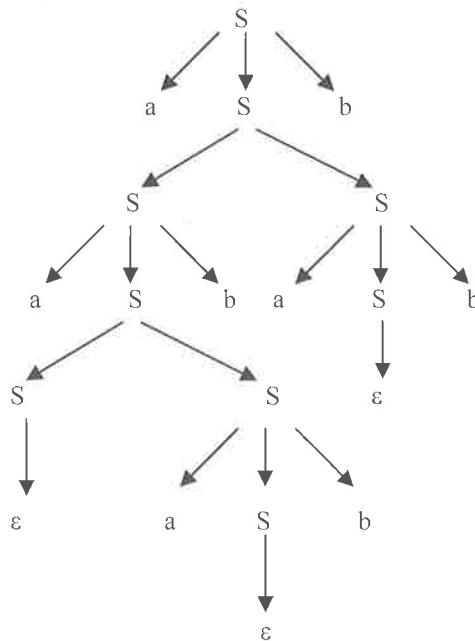
$$G = (\{S\}, \{a, b\}, S, P)$$

$$S \rightarrow aSb$$

$$P : \quad S \rightarrow SS$$

$$S \rightarrow \epsilon$$

ข้อสังเกตของไวยากรณ์ค่อนเท็กซ์ฟรี G ก็คือ ตัวแปร S จะสร้างจำนวนตัวอักษร a เท่ากับจำนวนตัวอักษร b เสมอ นอกจานี้ ตัวอักษร a ไม่จำเป็นต้องอยู่หน้าตัวอักษร b เสมอไป เนื่องจากมีโปรดักชัน $S \rightarrow SS$ เมื่อเราทดลองสร้างต้นไม้แสดงการได้มาจะช่วยอธิบายภาษาที่สร้างโดยไวยากรณ์ G ได้เป็นอย่างดี



รูปที่ 5.6 ต้นไม้แสดงการได้มา

สตริงที่ถูกสร้างจากต้นไม้แสดงการได้มาในรูปนี้คือ $aaabbabb$ ซึ่งผู้อ่านหลายท่านคงยังไม่เข้าใจรูปแบบของสตริงที่ถูกสร้างออกมาจากไวยากรณ์นี้ แต่ถ้าแทนตัวอักษร a ด้วยวงเล็บเปิด "(" และแทนตัวอักษร b ด้วยวงเล็บปิด ")" ปริมาณที่ซ่อนอยู่เบื้องหลังไวยากรณ์ G ก็จะถูกเฉลยออกมายังสตริง $((())()$ ซึ่งก็คือการสร้างวงเล็บเพื่อใช้ในการคำนวนพิจน์ทางคณิตศาสตร์นั่นเอง เช่น $(2*(3 + (8/2)) - (2*3))$ เป็นต้น ด้วยข้อสังเกตและเหตุผลดังกล่าวจึงพอจะสรุปได้ว่า ภาษาที่ถูกสร้างโดยไวยากรณ์ค่อนเท็กซ์ฟรี G คือ เซตของสตริงซึ่งประกอบด้วย เชตของตัวอักษร $\{a, b\}$ โดยที่จำนวนตัวอักษร a จะต้องเท่ากับจำนวนตัวอักษร b นอกจากนี้ สำหรับสตริงส่วนหน้า (Prefix) ใด ๆ จะต้องมีจำนวนตัวอักษร a มากกว่าหรือเท่ากับจำนวนตัวอักษร b ด้วย ทั้งนี้เพื่อป้องกันมิให้เกิดปัญหา เช่น $bbaa$ ซึ่งสตริงนี้ไม่สามารถสร้างจากไวยากรณ์ G ได้

$\therefore L(G) = \{w : w \in \{a, b\}^*\text{ และจำนวนตัวอักษร } a \text{ ใน } w \text{ เท่ากับจำนวนตัวอักษร } b \text{ ใน } w \text{ และสำหรับสตริงส่วนหน้าใด ๆ ของ } w \text{ จำนวนตัวอักษร } a \text{ จะต้องมากกว่าหรือเท่ากับจำนวนตัวอักษร } b \text{ เสมอ}\}$

5.2 วิธีการของไวยากรณ์

วิธีการ (Parsing) คือ การแสดงขั้นตอนการได้มาของสตริงที่ถูกสร้างออกมายจากไวยากรณ์ นั่นคือ หากมีสตริง s และไวยากรณ์ G เราสามารถทดสอบว่าสตริง s ถูกสร้างขึ้นมาโดยไวยากรณ์ G ได้โดยใช้กระบวนการวิธีการ ก่อนที่จะกล่าวถึงวิธีการอย่างละเอียด สิ่งที่จำเป็นต้องศึกษาก่อนก็คือกระบวนการได้มา (Derivation) ของภาษาคอมพิวเตอร์ ซึ่งจะเป็นการขยายความกระบวนการได้มาที่ได้อธิบายมาบ้างแล้วในตอนต้นของบทนี้

กระบวนการได้มา คือ กระบวนการที่ใช้ในการสร้างสตริง โดยเริ่มต้นที่สัญลักษณ์เริ่มต้นของไวยากรณ์ จากนั้นใช้กฎต่าง ๆ ที่อยู่ในโปรดักชันของไวยากรณ์สร้างสตริงของมาทีละขั้นตอน เช่น

กำหนดให้ไวยากรณ์ $G = (\{S\}, \{a, b\}, S, P)$

$$P : \quad S \longrightarrow aSa$$

$$S \longrightarrow bSb$$

$$S \longrightarrow \epsilon$$

สามารถแสดงกระบวนการได้มาของสตริง $abba$ ได้ดังนี้

$$S \longrightarrow aSa \longrightarrow abSba \longrightarrow abba$$

ดังนั้นสตริง $abba \in L(G)$

กระบวนการได้มาซึ่งสตริงนั้น สามารถทำได้ 2 วิธี คือ

1. การได้มาทางซ้าย (Left Derivation) คือ การได้มาโดยเลือกกระจายสัญลักษณ์บนเทอร์มินอล (Nonterminal) ตัวซ้ายมือสุดของสตริงก่อนทุกครั้ง

2. การได้มาทางขวา (Right Derivation) คือ การได้มาโดยเลือกกระจายสัญลักษณ์บนเทอร์มินอล (Nonterminal) ตัวขวา มือสุดของสตริงก่อนทุกครั้ง

เช่น กำหนดให้ไวยากรณ์ G คือ

$$S \longrightarrow aABb$$

$$A \longrightarrow aA \mid a$$

$$B \longrightarrow bBb \mid b$$

สามารถแสดงการได้มาซึ่งสตริง $aaabb$ ของทั้ง 2 วิธีได้ดังนี้

การได้มาทางซ้าย : $S \longrightarrow aABb \longrightarrow aaABb \longrightarrow aaaBb \longrightarrow aaabb$

การได้มาทางขวา : $S \longrightarrow aABb \longrightarrow aAbb \longrightarrow aaAbb \longrightarrow aaabb$

ตัวอย่างที่ 5.6 $S \longrightarrow aABb \mid aABBb$

$$A \longrightarrow aA \mid a$$

$$B \longrightarrow bBb \mid b$$

จะแสดงขั้นตอนการทำวิธีการเพื่อพิสูจน์ว่าสตริง $aaabb$ ถูกสร้างออกมายจากไวยากรณ์ข้างต้น รีเมต้นที่สัญลักษณ์เริ่มต้น S ซึ่งสามารถเลือกกระจายได้ 2 วิธี

$$S \longrightarrow aABb \quad (1)$$

$$S \longrightarrow aABBb \quad (2)$$

ขั้นตอนต่อไป กระจายทางเลือกทั้งสองโดยวิธีการได้มาทางซ้าย

จาก (1) มีทางเลือก 2 วิธี คือ

$$S \longrightarrow aaABb \quad (1.1)$$

$$S \longrightarrow aaBb \quad (1.2)$$

จาก (2) สามารถกระจายได้ 2 วิธี เช่นกัน

$$S \longrightarrow aaABBb \quad (2.1)$$

$$S \longrightarrow aaBBb \quad (2.2)$$

จากนั้น กระจายจากทางเลือกทั้งสี่วิธี (1.1, 1.2, 2.1, 2.2) อีกครั้ง จนกระทั่งได้สตริงอ กมาเป็น aaabb ซึ่งนั่นก็คือ

จาก (1.1) $S \longrightarrow aaABb \longrightarrow aaaBb$

ทำการได้มาทางซ้ายต่ออีกครั้งจะได้

$$aaaBb \longrightarrow aaabb$$

ดังนั้นสตริง aaabb อยู่ในภาษา $L(G)$

การทำจีวิภาคดังตัวอย่างนี้เรียกว่า จีวิภาคจากบันลั่ง (Top-down Parsing) ซึ่งเป็นการทำจีวิภาคที่เรียนรู้ แต่มีข้อเสียตรงที่จำเป็นจะต้องใช้เนื้อที่จำนวนมากในหน่วยความจำ เพื่อใช้ในการเก็บทางเลือกแต่ละทางที่ถูกกระจายออกมานะ ข้อเสียอีกประการของการทำจีวิภาคจากบันลั่ง คือ อาจเกิดเหตุการณ์ที่ทำให้การทำจีวิภาคไม่มีที่สิ้นสุด เหตุการณ์นี้มักจะเกิดขึ้นเมื่อไวยากรณ์มีโปรดักชัน ϵ (ϵ -production) รวมอยู่ด้วย เช่น

$$G : S \longrightarrow SS \mid ab \mid \epsilon$$

เมื่อทำการทำจีวิภาคจากบันลั่งจะได้ผลลัพธ์ดังนี้

$$S \longrightarrow SS \longrightarrow SSS \longrightarrow SSS \longrightarrow SS \longrightarrow SSS \longrightarrow \dots$$

การทำจีวิภาคในตัวอย่างนี้เกิดการกระจาย (yaw ขึ้นและสันลง) ไปเรื่อย ๆ ชั้นรูปแบบเดิม ทำให้มีจบสิ้นชั้นสาเหตุของปัญหาที่คือ การปล่อยให้ไวยากรณ์มีโปรดักชัน ϵ ดังนั้นเพื่อเป็นการป้องกันปัญหาที่จะเกิดขึ้น เราควรสร้างไวยากรณ์ที่ไม่มีโปรดักชัน ϵ

5.3 ความสับสน

ความสับสน (Ambiguity) เป็นสิ่งที่เกิดขึ้นได้กับไวยากรณ์ที่ถูกสร้างขึ้นอย่างไม่ระมัดระวัง ไวยากรณ์ใด ๆ จะถูกเรียกว่า ไวยากรณ์สับสน (Ambiguous Grammar) ก็ต่อเมื่อมีการได้มาที่แตกต่างกันอย่างน้อย 2 วิธี สำหรับสตริงเดียวกัน เช่น

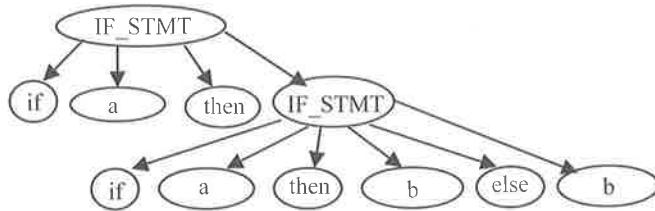
กำหนดให้ไวยากรณ์ $G = (\{IF_STMT, EXPR\}, \{if, then, a, b\}, IF_STMT, P)$ คือ

$IF_STMT \longrightarrow if EXPR \text{ then } IF_STMT |$

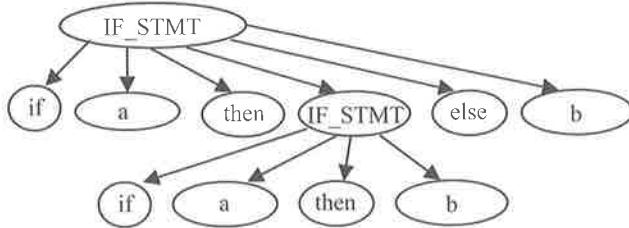
$if EXPR \text{ then } IF_STMT \text{ else } IF_STMT | b$

$EXPR \longrightarrow a$

สามารถสร้างต้นไม้แสดงการได้มาลำחרับสตริง “if a then if a then b else b” ที่แตกต่างกัน 2 ต้น คือ



รูปที่ 5.7 ต้นไม้แสดงการได้มาต้นที่ 1



รูปที่ 5.8 ต้นไม้แสดงการได้มาต้นที่ 1

ความแตกต่างของการได้มาซึ่งความแตกต่างในเรื่องของการได้มาทางซ้าย (Left Derivation) หรือการได้มาทางขวา (Right Derivation) แต่หมายถึงต้นไม้แสดงการได้มาที่แตกต่างกัน เนื่องจากการได้มาทางซ้ายอาจมีต้นไม้ที่เหมือนกันกับการได้มาทางขวาที่เป็นได้ ดังนั้นเพื่อให้เกิดความแตกต่างกันอย่างชัดเจน จึงให้ความสนใจที่ต้นไม้แสดงการได้มาเป็นลำดับ

ตัวอย่างที่ 5.7 กำหนดให้ไวยากรณ์ $G = (\{S\}, \{a, b\}, S, P)$

$$P : S \longrightarrow SabS \mid ab \mid \epsilon$$

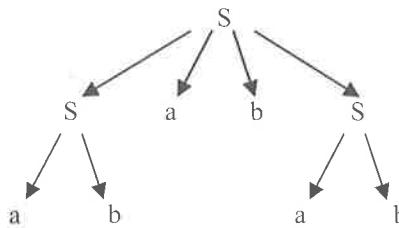
จะพิสูจน์ว่าไวยากรณ์ G เป็นไวยากรณ์ที่สับสน

พิสูจน์ : วิธีการพิสูจน์คือการยกตัวอย่างสตริงซึ่งเป็นสมาชิกอยู่ใน $L(G)$ จากนั้นสร้างต้นไม้แสดงการได้มาที่แตกต่างกัน 2 ต้น

ยกตัวอย่างสตริง $w = ababab$ ซึ่ง $w \in L(G)$

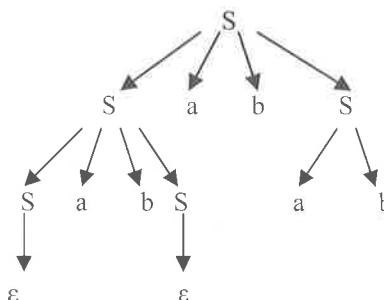
สามารถทำวิวัฒนาการได้แตกต่างกัน 2 วิธีสำหรับสตริง w ดังนี้

$$S \longrightarrow SabS \longrightarrow ababS \longrightarrow ababab$$



รูปที่ 5.9 ต้นไม้แสดงการได้มาต้นที่ 1

หรือ $S \longrightarrow SabS \longrightarrow SabSabS \longrightarrow abSabS \longrightarrow ababS \longrightarrow ababab$



รูปที่ 5.10 ต้นไม้แสดงการได้มาต้นที่สอง

ดังนั้นจึงสรุปได้ว่าไวยากรณ์ G เป็นไวยากรณ์ที่สับสน

ตัวอย่างที่ 5.8 จงแก้ไขไวยากรณ์ในตัวอย่างก่อนหน้านี้ เพื่อไม่ให้เป็นไวยากรณ์ที่สับสน

สาเหตุหนึ่งที่มักจะทำให้ไวยากรณ์เกิดความสับสนคือการอนุญาตให้มีทั้งการเวียนบังเกิดทางซ้าย (Left Recursion) และการเวียนบังเกิดทางขวา (Right Recursion) ของสัญลักษณ์ตัวแปร (Variable Symbol) ประกอบอยู่ภายในไวยากรณ์ นั่นคือ

$$S \longrightarrow SabS$$

จะเห็นได้ว่าตัวแปร S ประกอบอยู่ทั้งทางซ้ายและทางขวาของ ab ซึ่งเป็นสาเหตุหนึ่งของความสับสน (ในบางกรณี แม้ว่าสัญลักษณ์บนเทอร์มินอลจะไม่ได้มีอยู่ทั้งทางซ้ายและทางขวาของสัญลักษณ์เทอร์มินอล ไวยากรณ์นั้นก็อาจจะสับสนได้ดังแสดงในตัวอย่างถัดไป) ดังนั้นเราจะต้องจัดรูปแบบของไวยากรณ์เสียใหม่ ให้มีเพียงแค่การเวียนบังเกิดทางซ้าย หรือการเวียนบังเกิดทางขวาอย่างใดอย่างหนึ่งเท่านั้น จะได้ว่า

$$S \longrightarrow abS \mid ab \mid \epsilon$$

ตัวอย่างที่ 5.9 กำหนดให้ไวยากรณ์ $G = (\{S, X, Y\}, \{u, v\}, S, P)$

$$P : S \longrightarrow XY \mid uuY$$

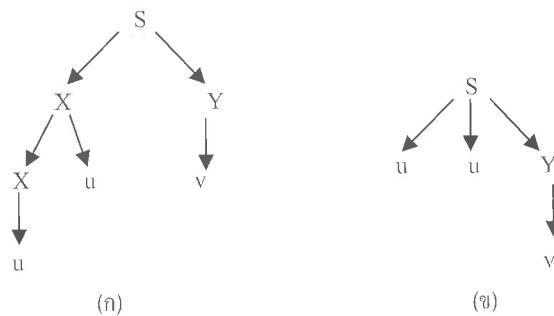
$$X \longrightarrow u \mid Xu$$

$$Y \longrightarrow v$$

จะพิสูจน์ว่าไวยากรณ์ G เป็นไวยากรณ์ที่สับสน

พิสูจน์ : ตัวอย่างนี้เป็นตัวอย่างที่ดีซึ่งแสดงให้เห็นว่า แม้ว่าไวยากรณ์จะไม่มีทั้งการเรียบบังเกิดทางซ้าย และทางขวาพร้อม ๆ กัน แต่ก็ทำให้เกิดความสับสนได้ ดังนั้นความสับสนของไวยากรณ์จึงเกิดได้จากหลายสาเหตุ วิธีการพิสูจน์ที่ดีที่สุดก็คือการยกตัวอย่างสตริงซึ่งเป็นสมาชิกอยู่ใน $L(G)$ และสร้างต้นไม้แสดงการได้มาที่แตกต่างกัน สำหรับสตริงนั้นอย่างน้อยสองต้น

ยกตัวอย่างสตริง $w = \text{uuv}$ เราสามารถสร้างต้นไม้แสดงการได้มาที่แตกต่างกันได้ดังนี้



รูปที่ 5.11 ต้นไม้แสดงการได้มาที่แตกต่างกัน 2 ต้น

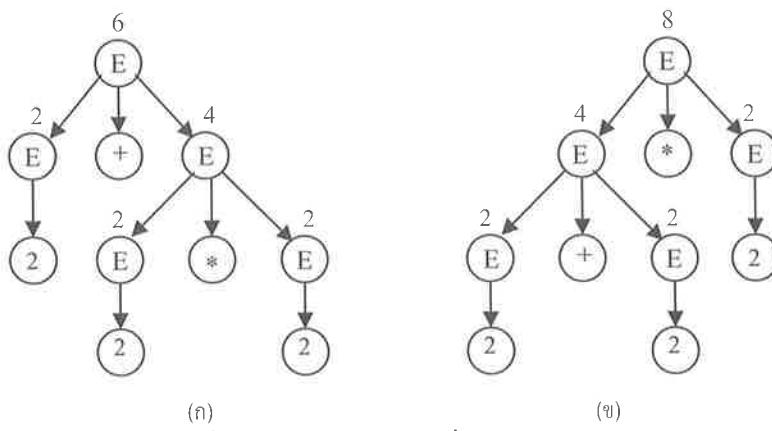
ดังนั้นจึงสรุปได้ว่าไวยากรณ์ G นี้เป็นไวยากรณ์ที่สับสน

เหตุใดเราไม่ต้องการไวยากรณ์ที่สับสน ?

ผู้อ่านคงเกิดความสงสัยว่าทำไมเราต้องศึกษาเรื่องไวยากรณ์ที่สับสน และเหตุใดไวยากรณ์ตั้งกล่าวจึงไม่เป็นที่ต้องการของเรา เหตุผลที่ตอบคำถามนี้ได้ดีที่สุดคือการยกตัวอย่างการคำนวณนิพจน์ทางคณิตศาสตร์ โดยกำหนดให้ไวยากรณ์ $G_1 = (\{E\}, \{2, +, *, (\cdot)\}, E, P)$

$$P : E \longrightarrow E + E \mid E * E \mid (E) \mid 2$$

กำหนดให้สตริง $w = 2 + 2 * 2$ เราสามารถสร้างต้นไม้แสดงการได้มาที่แตกต่างกันได้ 2 ต้นดังนี้



รูปที่ 5.12 ต้นไม้แสดงการได้มาที่แตกต่างกัน 2 ต้น

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

ผลจากการคำนวณนิพจน์สำหรับต้นไม้แสดงการได้มาทางซ้ายเท่ากับ 6 ในขณะที่ผลของต้นไม้ทางขวาเมื่อค่าเท่ากับ 8 ซึ่งแสดงให้เห็นว่าต้นไม้แสดงการได้มาที่แตกต่างกันมีผลทำให้คำตอบแตกต่างกันด้วย ดังนั้นจึงควรสร้างไวยากรณ์ด้วยความระมัดระวัง เนื่องจากผลเสียที่เกิดจากความลับสนของไวยากรณ์สามารถทำลายคุณค่าของภาษาที่ลูกสร้างจากไวยากรณ์นั้นได้

สำหรับไวยากรณ์นี้ เราสามารถแก้ไขไม่ให้เกิดความลับสนได้ดังนี้

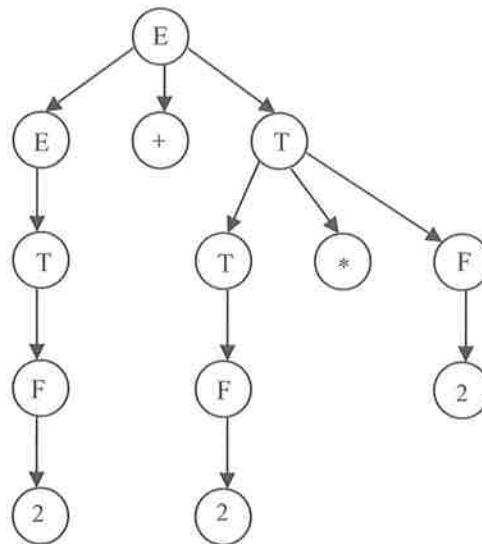
$$G_2 = (\{E, T, F\}, \{2, +, *, (\), \), E, P)$$

$$P : \quad E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (E) \mid 2$$

ไวยากรณ์ G_2 สามารถสร้างภาษาได้เหมือนกับไวยากรณ์ G_1 แต่ไม่มีความลับสนเหมือนกับที่ไวยากรณ์ G_1 มีกับสตริง $2 + 2 * 2$ และรวมถึงสตริงอื่น ๆ ด้วย ต้นไม้แสดงการได้มาที่สร้างจากไวยากรณ์ G_2 สำหรับสตริง $2 + 2 * 2$ สามารถแสดงได้ดังนี้ ดังนี้



รูปที่ 5.13 ต้นไม้แสดงการได้มาซึ่งสร้างได้เพียงต้นเดียวเท่านั้น

5.4 การจัดรูปแบบไวยากรณ์

การปล่อยให้ไวยากรณ์ตอนที่เก็ชฟรีมีอิสระทางฝั่งขวาของโปรดักชันมากเกินไป อาจก่อให้เกิดปัญหา
มากมาย เช่น การทำงานวิภาคที่ไม่วันจบหรือความลับสนของไวยากรณ์ ดังนั้นจึงควรจัดรูปแบบของไวยากรณ์
ตอนที่เก็ชฟรีให้อยู่ในรูปแบบที่เหมาะสม เพื่อหลีกเลี่ยงปัญหาที่จะเกิดขึ้น

วิธีการจัดรูปแบบของไวยากรณ์ค่อนเท็จฟรีมีดังนี้

1. กำหนดโปรดักชัน ϵ โปรดักชัน ϵ เป็นสาเหตุที่ทำให้เกิดกระบวนการทำงานวิเคราะห์ที่ไม่ลื้นสุด ดังนั้น จึงควรแปลงรูปไวยากรณ์ไม่ใหม่เป็นโปรดักชัน ϵ ปรากฏอยู่ สำหรับวิธีการแปลงรูปสามารถอธิบายได้ด้วยตัวอย่างไวยากรณ์ดังนี้

$$A \longrightarrow xyBC$$

$$B \longrightarrow yxB \mid \epsilon$$

$$C \longrightarrow cC \mid c$$

ไวยากรณ์นี้มีโปรดักชัน ϵ ซึ่งเราสามารถกำจัดได้โดยแปลงไวยากรณ์เป็น

$$A \longrightarrow xyBC \mid xyC$$

$$B \longrightarrow yxB \mid yx$$

$$C \longrightarrow cC \mid c$$

2. กำจัดกฎที่ไม่สามารถสร้างสตริงได้ กฎที่ไม่สามารถสร้างสตริงได้มีอยู่ 2 ประเภท คือ

(1) กฎที่ไม่สามารถถูกกระจายมาจากสัญลักษณ์เริ่มต้น เช่น

$$S \longrightarrow aAb$$

$$A \longrightarrow aA \mid a$$

$$C \longrightarrow c$$

กฎ $C \longrightarrow c$ เป็นกฎที่ไม่สามารถถูกกระจายมาจากสัญลักษณ์เริ่มต้น S เราสามารถกำจัดกฎประเภทนี้ได้ด้วยวิธีง่าย ๆ คือ การลบกฎนั้นทิ้งไปจากไวยากรณ์

(2) กฎที่เกิดการวนซ้ำไม่รู้จบ เช่น

$$S \longrightarrow aAb \mid aBb$$

$$A \longrightarrow aA \mid a$$

$$B \longrightarrow bB$$

กฎ $B \longrightarrow bB$ เป็นกฎที่วนซ้ำไม่รู้จบ ดังนั้นจึงไม่สามารถสร้างสตริงได้ เราจะต้องกำจัดทุก ๆ กฎที่มีการอ้างถึงสัญลักษณ์ B ซึ่งจะได้ว่า

$$S \longrightarrow aAb$$

$$A \longrightarrow aA \mid a$$

3. กำจัดกฎที่สร้างสัญลักษณ์ non-terminal (Nonterminal) 1 ตัวเท่านั้น นั่นคือ

$$S \longrightarrow A \text{ โดยที่ } S \text{ และ } A \text{ เป็นสัญลักษณ์ non-terminal}$$

เช่น

$$S \longrightarrow aAb$$

$$A \longrightarrow B$$

$$B \longrightarrow bB \mid b$$

ควรจะกำจัดกฎ $A \longrightarrow B$ ออกไปเนื่องจากไม่มีประโยชน์ ดังนั้นไวยากรณ์ใหม่ คือ

$$S \longrightarrow aBb$$

$$B \longrightarrow bB \mid b$$

4. พยายามแทนสัญลักษณ์ที่สามารถสร้างสัญลักษณ์เทอร์มินอลได้ทันที เช่น

$$S \longrightarrow aAbB$$

$$A \longrightarrow a | b | aa | bb$$

$$B \longrightarrow bB | b$$

จากไปรดักชันในบรรทัดที่ 2 เราสามารถสร้างสัญลักษณ์เทอร์มินอลได้ทันทีจากสัญลักษณ์นอนเทอร์มินอล A ดังนั้นจึงไม่มีประโยชน์ที่จะแยกไปรดักชันในบรรทัดที่ 2 อีกนา ความสามารถรวมไปรดักชันในบรรทัดที่ 2 เข้าไปไว้ในบรรทัดแรกได้ทันที

$$S \longrightarrow aabB | abbB | aaabB | abbbB$$

$$B \longrightarrow bB | b$$

นอกเหนือจากวิธีการจัดรูปแบบไวยากรณ์ทั้ง 4 ข้อ เพื่อหลีกเลี่ยงปัญหาที่อาจเกิดกับไวยากรณ์แล้ว ยังมีการกำหนดรูปแบบที่เป็นมาตรฐานที่เข้าใจกันในหมู่นักวิทยาศาสตร์คอมพิวเตอร์ นั่นคือ รูปแบบชอมสกี (Chomsky) และรูปแบบกรีบาก (Greibach) ซึ่งผู้อ่านกำลังจะได้ศึกษาในหัวข้อต่อไป

5.5 รูปแบบไวยากรณ์แบบชอมสกี

กำหนดให้ S, X, Y เป็นสัญลักษณ์นอนเทอร์มินอล และ a, b เป็นสัญลักษณ์เทอร์มินอล แล้วไวยากรณ์ชอมสกี (Chomsky Normal Form) มีรูปแบบดังนี้

$$S \longrightarrow XY$$

$$X \longrightarrow b$$

$$Y \longrightarrow a$$

ทางขวาของกฎจะต้องเป็นสัญลักษณ์นอนเทอร์มินอลจำนวน 2 ตัว หรือเป็นสัญลักษณ์เทอร์มินอลเพียง 1 ตัว เท่านั้น

ตัวอย่างที่ 5.10 $S \longrightarrow aXY | XY$

$$X \longrightarrow ba | bX | b$$

$$Y \longrightarrow c$$

ไวยากรณ์ดังกล่าวไม่อยู่ในรูปแบบของชอมสกี เนื่องจากมีกฎหลายข้อที่ขัดแย้งกับรูปแบบ นั่นคือ

$$S \longrightarrow aXY,$$

$$X \longrightarrow ba,$$

$$X \longrightarrow bX$$

สามารถจัดรูปแบบไวยากรณ์ให้อยู่ในรูปแบบของชอมสกีได้ดังนี้

$$S \longrightarrow ZY | XY$$

$$Z \longrightarrow AX$$

$$A \longrightarrow a$$

$$X \longrightarrow BA \mid BX \mid b$$

$$B \longrightarrow b$$

$$Y \longrightarrow c$$

5.6 รูปแบบไวยากรณ์แบบกรีบَاค

กำหนดให้ S เป็นสัญลักษณ์อนเทอร์มินอล V เป็นเซตของสัญลักษณ์อนเทอร์มินอล a เป็นสัญลักษณ์เทอร์มินอล และ $Z \in V^*$ ไวยากรณ์แบบกรีบَاค (Greibach Normal Form) มีรูปแบบดังนี้

$$S \longrightarrow aZ$$

นั่นคือ ทางขวาของกฎจะต้องเริ่มต้นด้วยสัญลักษณ์เทอร์มินอล 1 ตัว และตามด้วยสัญลักษณ์อนเทอร์มินอลตั้งแต่ 0 ตัวขึ้นไป

ตัวอย่างที่ 5.11 $S \longrightarrow ab \mid XY$

$$X \longrightarrow aa$$

$$Y \longrightarrow bY \mid b$$

ไวยากรณ์นี้ยังไม่มีอยู่ในรูปแบบของกรีบَاค เนื่องจากกฎบางข้อไม่มีอยู่ในรูปแบบที่กำหนด นั่นคือ

$$S \longrightarrow ab,$$

$$S \longrightarrow XY$$

เราสามารถจัดรูปแบบได้ดังนี้

$$S \longrightarrow aB \mid aAY$$

$$B \longrightarrow b$$

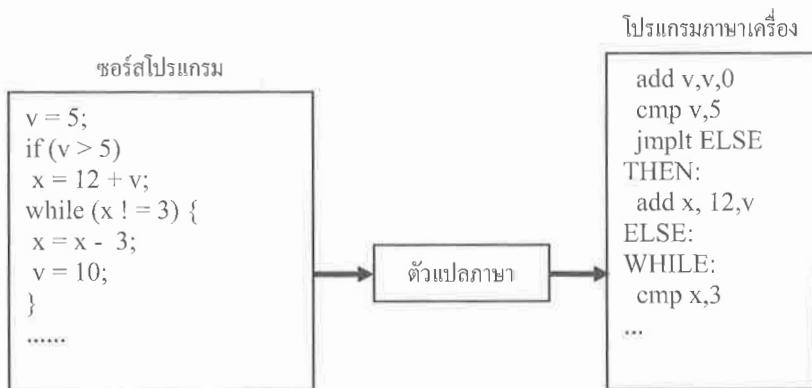
$$Y \longrightarrow bY \mid b$$

$$A \longrightarrow a$$

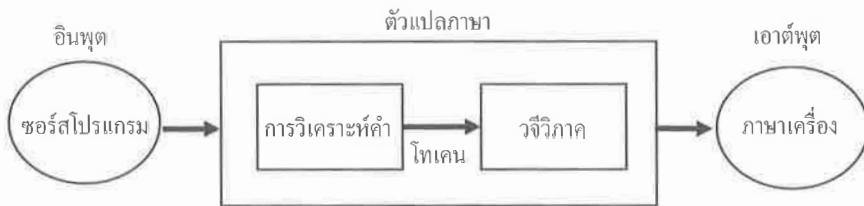
จะเห็นได้ว่า X เป็นสัญลักษณ์ที่สามารถสร้างสัญลักษณ์เทอร์มินอลได้ทันที ดังนั้นเราจึงควรแทนค่า X ลงในกฎทุกข้อที่มีการอ้างถึง X จากนั้นก็จัด X ออกไปจากไวยากรณ์

5.7 ตัวแปลงภาษา

หลังจากที่ได้ศึกษาไวยากรณ์ตอนเทิกซ์ฟรีและภาษาตอนเทิกซ์ฟรีแล้ว ผู้อ่านก็คงพอจะมองภาพของตัวแปลงภาษาคอมพิวเตอร์ออกได้大概 กิจกรรมของตัวแปลงภาษา (Compiler) เริ่มต้นจากนำอินพุตสดริง (ซึ่งก็คือซอฟต์แวร์ที่โปรแกรมเมอร์เขียนขึ้นจากภาษาคอมพิวเตอร์ เช่น ภาษา C, Java เป็นต้น) มาแปลงให้กลายเป็นโทเกน (Token) หรือที่เรารู้จักในชื่อสัญลักษณ์อนเทอร์มินอลโดยกระบวนการวิเคราะห์คำ (Lexical Analyzer) จากนั้นนำไฟล์ที่ประกอบด้วยสัญลักษณ์อนเทอร์มินอลส่งต่อไปยังหน่วยที่ทำงานวิเคราะห์คำ (Parser) เพื่อตรวจสอบว่าสามารถสร้างอินพุตสดริงที่รับเข้ามาได้จากไวยากรณ์ที่ภาษาคอมพิวเตอร์กำหนดไว้หรือไม่ หากไวยากรณ์มีความผิดต้อง ก็จะนำไปสร้างเป็นโปรแกรมภาษาเครื่อง (Machine Code) ต่อไป ดังรูป



รูปที่ 5.14 ตัวแปลภาษา

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)

รูปที่ 5.15 ส่วนประกอบภายในตัวแปลภาษา

สำหรับกระบวนการทำงานวิจิภัคเพื่อตรวจสอบว่าซอฟต์แวร์สโปรограмมที่รับเข้ามาเป็นถูกต้องตามหลักไวยากรณ์ของภาษาหรือไม่ สามารถทำได้หลายวิธี แต่วิธีที่ง่ายและตรงไปตรงมาที่สุดคือ วิธีการค้นหาแบบอึ่งซอสทีฟ (Exhaustive Search) ซึ่งเป็นการค้นหาทุก ๆ วิธีการได้มา (Derivation) ที่เป็นไปได้ โดยเริ่มต้นจากสัญลักษณ์เริ่มต้นไปยังอินพุตสตริงของໂທเคน ตัวอย่างเช่น กำหนดให้ไวยากรณ์ของภาษาคือ

$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow aSb \\ S &\rightarrow bSa \\ S &\rightarrow \epsilon \end{aligned}$$

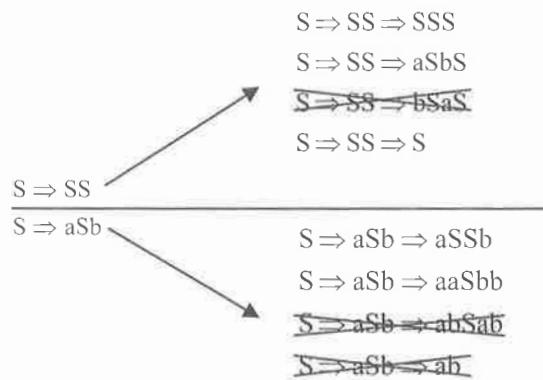
และอินพุตสตริงของໂທเคนคือ aabb การทำงานวิจิภัคโดยการค้นหาแบบอึ่งซอสทีฟสามารถทำเป็นขั้นตอน ทีละขั้นดังนี้

ขั้นตอนที่ 1 : พิจารณาการได้มาในทุก ๆ ทางเลือกของไวยากรณ์

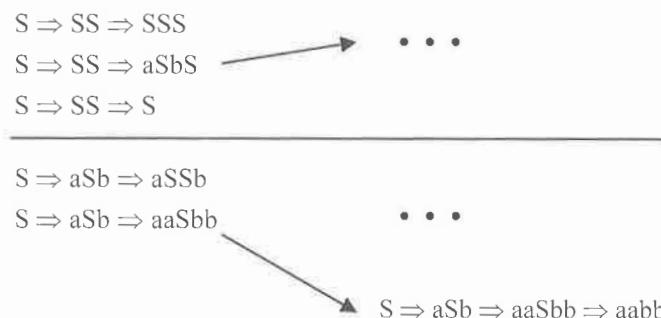
$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow aSb \\ \cancel{S \rightarrow bSa} \\ \cancel{S \rightarrow \epsilon} \end{aligned}$$

เนื่องจากสตริงของเราระมต้นด้วยໂທເຄນ a ດังນັ້ນເຮົາຈຶ່ງຕັດ 2 ທາງເລືອກສຸດທ້າຍອກໄປ

ຫຸ້ນຕອນທີ 2 : ກະຈາຍຕ່ອໄປອຶກເຊັ່ນເດືອກກັບຫຸ້ນຕອນທີ 1 ສໍາຫຼັບໃນແຕ່ລະທາງເລືອກຊື່ຍັງໄໝຄູກຕັດທີ່



ຫຸ້ນຕອນທີ 3 : ກະຈາຍຕ່ອໄປອຶກເຊັ່ນເດືອກກັບຫຸ້ນຕອນທີ 2 ສໍາຫຼັບໃນແຕ່ລະທາງເລືອກຊື່ຍັງໄໝຄູກຕັດທີ່ ຜຶ່ງ
ໜີ່ໃນທາງເລືອກທີ່ພົບຄວາມສໍາເລົງຄື່ອ $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$



ດັ່ງນັ້ນຈຶ່ງສຽບຢືນວ່າສตรິງຂອງໂທເຄນ $aabb$ ຄູກຕັດຕາມໄວຍາກຮົມຂອງພາຍາ ເນື່ອຈາກສາມາຄະກະຈາຍຈາກ
ສີ່ຄູກຄົມທີ່ເລີ່ມຕົ້ນມາຢັງສຕຣິງ $aabb$ ໄດ້ ໂປຣດສັງເກດວ່າໄວຍາກຮົມທີ່ປະກອບດ້ວຍສີ່ຄູກຄົມທີ່ວັນແປຣທີ່ສາມາຄະສວັງ
ສຕຣິງວ່າງໄດ້ ນັ້ນຄື່ອ $S \rightarrow \epsilon$ ດັ່ງນັ້ນຈຶ່ງທຳໃຫ້ສຕຣິງທີ່ກຳລັງກະຈາຍສາມາຄະຫຼັກສັ້ນທີ່ຍ້ອຍວາອຳໄດ້ໂດຍໄມ່ຈຳກັດ ສິ່ງນີ້ຈະ
ກ່ອໄຂເກີດກາຕິດລູປ໌ຈຶ່ງທຳໃຫ້ຄັນຫາຄຳຕອບໄນ່ເຈືອແຫຍຸດກາກໍາວົງວິວາການໄດ້ ຕ້ວອຍ່າງນີ້ຈຶ່ງເປັນຄຳຕອບທີ່ດີຂອງຄຳຄານ
ທີ່ວ່າເຫຼຸດໄວຍາກຮົມທີ່ດີ່ໄມ່ຄວາມມີສີ່ຄູກຄົມທັງແປຣທີ່ກະຈາຍເປັນສຕຣິງວ່າງໄດ້

การทำโจทย์วิธีการค้นหาแบบอีซอสที่ฟันเป็นวิธีที่เรียบง่าย แต่สำหรับไวยากรณ์ที่มีโปรดักชันเป็นจำนวนมาก จะใช้เวลาอย่างนาน ทำให้ไม่เหมาะสมกับการประยุกต์ใช้งานจริง หากกำหนดให้มีจำนวนของโปรดักชันภายในไวยากรณ์เท่ากับ k ก็จะ และสตริงของโทเคนมีความยาว $|w|$ จะต้องใช้เวลาในการค้นหาแบบอีซอสที่ฟันเป็นวิธีเท่ากับ

$$k + k^2 + \cdots + k^{2^{|w|}}$$

พิสูจน์ :

- เวลาที่ใช้ในการค้นหาขั้นตอนที่ 1 เท่ากับ k (สมมุติว่าไม่มีทางเลือกใดถูกตัดทิ้งเลย)
- เวลาที่ใช้ในการค้นหาขั้นตอนที่ 2 เท่ากับ k^2 (เนื่องจากทางเลือก k ทางในขั้นตอนที่ 1 สามารถกระจายต่อได้อีกสูงสุด k วิธี)

หากลังเกตการได้มาซึ่งสตริงโดยวิธีการค้นหาแบบอีซอสที่ฟ ทุกครั้งที่เรากระจายได้สัญลักษณ์เทอร์มินอล เราจะได้สัญลักษณ์ตัวแปรออกมาร่วมด้วย ดังนั้นหากสตริงยาว $|w|$ เราจะต้องใช้จำนวนขั้นตอน $|w|$ ขั้น เพื่อสร้างสัญลักษณ์เทอร์มินอล และใช้อีก $|w|$ ขั้น ในการกำจัดสัญลักษณ์ตัวแปรออกไป ดังนั้นสำหรับสตริงที่มีความยาว $|w|$ เราจะต้องใช้ขั้นตอนในการค้นหาทั้งสิ้น $2|w|$

- เวลาที่ใช้ในการค้นหาขั้นตอนที่ $2|w|$ เท่ากับ $k^{2^{|w|}}$

นั่นคือ การค้นหาจึงใช้เวลาทั้งสิ้น $k + k^2 + \cdots + k^{2^{|w|}} = O(k^{2^{|w|}})$ ซึ่งเวลานี้เป็นเวลาที่นานมาก อย่างไรก็ตาม มีอัลกอริทึมที่มีประสิทธิภาพในการทำงานจำนวนมาก เช่น ไวยากรณ์ที่มีความยาว $|w|^3$ เราสามารถคำนวณได้ในเวลาที่ $O(n^3)$ แต่ต้องใช้เวลาที่มากกว่า $O(n^2)$ ในการคำนวณ

การทำโจทย์วิภาคโดยอัลกอริทึม CYK

อัลกอริทึม CYK (Cocke-Younger-Kasami) เป็นวิธีที่ใช้ตรวจสอบว่าอินพุตสตริงเป็นสมาชิกอยู่ในภาษาที่เราพิจารณาหรือไม่ โดยใช้หลักการของเทคนิคการเขียนโปรแกรมแบบไดนามิก (Dynamic Programming Technique) ทำให้ลดความซ้ำซ้อนของการทำงานลงมาเป็น $|w|^3$ โดยเราต้องป้อนไวยากรณ์ของภาษาในรูปแบบของชอมสกี (Chomsky Normal Form) และอินพุตสตริง w ให้กับอัลกอริทึม จากนั้นอัลกอริทึมจะให้คำตอบการเป็นสมาชิกของสตริง w ก่อนที่จะเริ่มใช้งานอัลกอริทึม เราจะต้องสร้างตารางของ V_{ij} ขึ้นมาก่อน ดังนี้

กำหนดให้อินพุตสตริง $w = aabb$ และไวยากรณ์ G คือ

$$S \rightarrow AB$$

$$A \rightarrow BB$$

$$A \rightarrow a$$

$$B \rightarrow AB$$

$$B \rightarrow b$$

1. นำอินพุตสตริง $aabb$ มาใส่ที่หัวคอลัมน์ที่ลงทะเบียน

2. นำสตริงย่อที่เป็นไปได้ทั้งหมดของ $aabb$ มาใส่ในบรรทัด $j = 2$ โดยให้มีขนาดความยาวของสตริงย่อเท่ากับ 2 จากนั้นสำหรับบรรทัดที่เหลือ ($j = 3$ ถึง 5) ให้ทำด้วยกันบรรทัดที่ 2 แต่เพิ่มความยาวของสตริงย่อขึ้นทีละ 1 ตัวอักษร

V_{ij}	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	
$j = 1$	a	a	b	b	b	I.
$j = 2$	aa	ab	bb	bb		II.
$j = 3$	aab	abb	bbb			
$j = 4$	aabb	abbb				
$j = 5$	aabbb					

รูปที่ 5.16 ตารางของ V_{ij}

อัลกอริทึม CYK

เริ่มต้น

(1) สำหรับ $i = 1$ ถึง $|w|$ ให้ทำดังนี้

(2) $V_{i1} = \{A \mid \text{มีโปรดักชัน } A \rightarrow x \text{ โดยที่ } x \text{ คือสัญลักษณ์เทอร์มินอลในแล้ว } j = 1\}$;

(3) สำหรับ $j = 2$ ถึง $|w|$ ให้ตามบรรทัดที่ (4) - (7)

(4) สำหรับ $i = 1$ ถึง $|w| - j + 1$ ให้ทำดังนี้

$$(5) \quad V_{ij} = \emptyset$$

(6) สำหรับ $k = 1$ ถึง $j - 1$ ให้ทำดังนี้

$$(7) \quad V_{ij} = V_{ij} \cup \{A \mid \text{มีโปรดักชัน } A \rightarrow BC \text{ โดยที่ } B \in V_{ik} \text{ และ } C \in V_{i+k, j-k}\}$$

จบ

เมื่อเราใช้อัลกอริทึม CYK กับตาราง V_{ij} จะได้ผลดังนี้

- สำหรับการทำงานในบรรทัดที่ (1) - (2) ตรวจสอบไวยากรณ์ว่ามีตัวแปรตัวใดที่สามารถกระจายออกมามาเป็นสตริง a และ b ได้โดยตรงบ้าง ซึ่งก็คือตัวแปร A และ B ตั้งนี้ผลจากการทำงานในบรรทัดที่ (1) - (2) คือ

a A	a A	b B	b B	b B
aa	ab	bb	bb	
aab	abb	bbb		
aabb	abbb			
aabbb				

รูปที่ 5.17 ตาราง V_{ij} สำหรับการทำงานในบรรทัดที่ (1) - (2)

- สำหรับการทำงานในบรรทัดที่ (3) - (7) เมื่อ $j = 2$ สามารถพิจารณาเป็นตัวอย่างได้ดังนี้ ช่องที่ ① มีสตริงประจำช่องคือ aa ซึ่งเกิดจาก a ในช่อง V_{11} มาต่อกับ a ในช่อง V_{21} จึงเปรียบเสมือนกับการนำ A กับ A มาต่อกันเป็น AA อย่างไรก็ตาม ไม่มีโปรดักชนใดในไวยากรณ์ที่กระจายออกมานะ AA ได้ ดังนั้นช่อง ① จึงไม่มีตัวแปรอยู่เลย

- สำหรับช่อง ② มีสตริงประจำช่องคือ ab ซึ่งเกิดจาก a ในช่อง V_{21} มาต่อกับ b ในช่อง V_{31} จึงเปรียบเสมือนกับการนำ A กับ B มาต่อกันเป็น AB ซึ่งโปรดักชนที่กระจายออกมานะได้ AB ก็คือ $S \rightarrow AB$ และ $B \rightarrow AB$ นั่นเอง ทำให้ช่อง ② มีตัวแปร S และ B ประจำอยู่ ดังนั้นผลจากการทำงานในบรรทัดที่ (3) - (7) เมื่อ $j = 2$ คือ

a A	a A	b B	b B	b B
aa ①	ab ② S, B	bb A	bb A	
aab	abb	bbb		
aabb	abbb			
aabbb				

รูปที่ 5.18 ตาราง V_{ij} สำหรับการทำงานในบรรทัดที่ (3) - (7) เมื่อ $j = 2$

- สำหรับการทำงานในบรรทัดที่ (3) - (7) เมื่อ $j = 3$ เราสามารถพิจารณาเป็นตัวอย่างได้ดังนี้ ช่องที่ ③ มีสตริงประจำช่องคือ aab ซึ่งเกิดขึ้นได้ 2 กรณี คือ 1. นำ a ในช่อง V_{11} มาต่อกับ ab ในช่อง V_{22} 2. นำ aa ในช่อง V_{12} มาต่อกับ b ในช่อง V_{31} ดังนั้นเมื่อนำตัวแปรของช่องดังกล่าวในทั้งสองกรณีมาเรียงต่อกันจะได้เป็น

$\{AS, AB\}$ และ $\{B\}$ ตามลำดับ ซึ่งตัวแปรที่กระจายออกมานี้ได้เป็นสมาชิกในเซตทั้งสองคือ S และ B เนื่องจากกระจายออกมานี้ได้เป็น AB ฉะนั้น ของ ③ จึงมีตัวแปร S และ B ประจำอยู่

จากวิธีการดังกล่าว ผลจากการทำงานในบรรทัดที่ (3) - (7) เมื่อ $j = 3, 4$ และ 5 คือ

a A	a A	b B	b B	b B
aa	ab S, B	bb A	bb A	
aab ③ S, B	abb A	bbb S, B		
aabb A	abbb S, B			
aabbb S, B				

รูปที่ 5.19 ตาราง V_{ij} สำหรับการทำงานในบรรทัดที่ (3) - (7) เมื่อ $j = 3, 4$ และ 5

เนื่องจากมี S เป็นสมาชิกอยู่ใน $V_{1,5}$ ดังนั้นจึงสรุปได้ว่า $aabbb \in L(G)$
ข้อสังเกต : ในการวิ่งของ V_{ij} จะมีลักษณะเป็นตัว V ดังรูป

a A	a A	b B	b B	b B
aa	ab S, B	bb A	bb A	
aab S, B	abb A	bbb S, B		
aab A	abbb S, B			
aabbb S, B				

รูปที่ 5.20 ลักษณะการวิ่งภายในตาราง V_{ij}

และการจับคู่ภาษาในเส้นทางจะมีลักษณะดังนี้

a A	a A	b B	b B	b B
aa	ab	bb	bb	
aah S, B	abb A	bbb S, B		
aabb Ⓐ	abbb S, B			
aabbb S, B				

รูปที่ 5.21 ลักษณะการจับคู่ภาษาในตาราง V_{ij}

สำหรับความซับซ้อนของอัลกอริทึม CYK นั้นชัดเจนว่ามีลูปทั้งหมด 3 ลูป ซ้อนกันอยู่ โดยแต่ละลูป มีการทำซ้ำเท่ากับ $|w|$ ครั้งโดยประมาณ ดังนั้นความซับซ้อนของอัลกอริทึมนี้จึงมีค่าเป็น $|w|^3$

แบบฝึกหัด

1. การพิสูจน์ว่าภาษาหนึ่งเป็นภาษาคอนเทกซ์ฟรีสามารถทำได้อย่างไรบ้าง
2. จงพิสูจน์ว่าไวยากรณ์ดังต่อไปนี้เป็นไวยากรณ์ที่สัมสน

$$S \rightarrow S + S \mid S - S \mid S^*S \mid S/S \mid c$$

3. กำหนดให้ไวยากรณ์ G_1 และ G_2 คือ

$$G_1 = (\{S\}, \{a, b\}, S, P_1)$$

$$P_1 : \quad S \rightarrow aSb \mid SS \mid \epsilon$$

$$G_2 = (\{S\}, \{a, b\}, S, P_2)$$

$$P_2 : \quad S \rightarrow aSb \mid abS \mid \epsilon$$

จงพิสูจน์ว่า $L(G_1) \neq L(G_2)$

ข้อ 4 - 7 จงพิสูจน์ว่าภาษาดังต่อไปนี้ เป็นภาษาคอนเทกซ์ฟรี

4. $L = \{w \in \{a, b\}^* : \text{จำนวนตัวอักษร } a \text{ ภายในสตริง } w \text{ มีจำนวนเป็น } 2 \text{ เท่าของจำนวนตัวอักษร } b \text{ ภายในสตริง } w\}$

$$5. \quad L = \{a^i b^j \mid i \leq j\}$$

$$6. \quad L = \{a^i b^j c^k \mid k = i + j\}$$

$$7. \quad L = \{a^i b^j c^k \mid j = i + k\}$$

ข้อ 8 - 10 จงอธิบายภาษาที่ลูกสร้างออกแบบจากไวยากรณ์คอนเทกซ์ฟรีดังต่อไปนี้

$$8. \quad S \rightarrow aSb \mid bSa \mid \epsilon$$

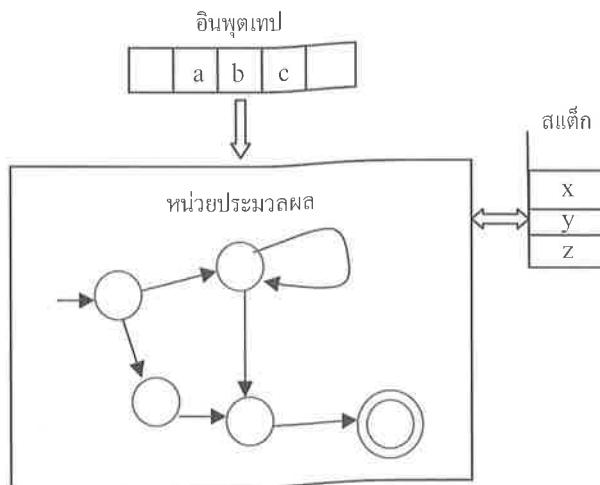
$$9. \quad S \rightarrow aSa \mid bSb \mid a \mid b$$

$$10. \quad S \rightarrow aA \mid bA \mid a \mid b$$

$$A \rightarrow aS \mid bS$$

ພູ້ດາວນ້ອໂຕມາຕາ

ພູ້ດາວນ້ອໂຕມາຕາ (Pushdown Automata : PDA) ເປັນອອໂຕມາຕາທີ່ສາມາຄຣອງຮັບກາຍາໄດ້ກໍວັງກວ່າກາຍາທີ່ໄຟໄຟຕ້ອໂຕມາຕາຮອງຮັບໄດ້ ສ່ວນປະກອບກາຍາໃນພູ້ດາວນ້ອໂຕມາຕາປະກອບດ້ວຍ 3 ສ່ວນ ດື່ອ ອິນພຸດເຫປ່ ຜຶ່ງບຽງຈຸຕ້ວອ້ກຍຣ (Alphabet) ທີ່ຈະຄຸກປົ້ນເຂົ້າມາໃຫ້ປະມວລຜລ ມີວ່າວ່າອິນພຸດເຫປ່ນີ້ເຄີ່ມືນໍ້າຈັກຊ້າຍໄປຂາດໄດ້ ໄນມີສ່າມາຄລັບມາຍັງຕໍ່ແນ່ງທີ່ເຄຍອ່ານແລ້ວໄດ້ ນອກຈາກນີ້ຍັງມີສ່າມາຄບັນທຶກຕ້ວອ້ກຍຣໄດ້ ຖໍ່ ລົງໄປໃນເຫປ່ໄດ້ ສ່ວນທີ່ 2 ດື່ອ ສແຕ໌ກ (Stack) ຜຶ່ງທຳນ້າທີ່ເກີ່ມພລັບພົບຮ້ວ່າຄວາມທີ່ໄດ້ຈາກການປະມວລຜລ ໂດຍສແຕ໌ກຈະມີສັງລັກຍໍລ ເຮັ່ມຕົ້ນເຫັນ z , $\$$ ເປັນຕົ້ນ ສ່ວນທີ່ 3 ດື່ອ ແນ່ວຍປະມວລຜລ ຜຶ່ງຈະມີສຄານະ (State) ທີ່ສ່າມາຄປ່ລືຢັນແປ່ລົງໄດ້ອູ່ງໆກ່າຍໃນການປ່ລືຢັນແປ່ລົງຂອງສຄານະນີ້ຈະຫຼືນ້ອງຢູ່ກັບອິນພຸດແລະສັງລັກຍໍລສ່ວນນັ້ນຂອງສແຕ໌ກ ລຸ່ມ ຂະເວລານີ້



ຮູບທີ່ 6.1 ສ່ວນປະກອບກາຍາໃນພູ້ດາວນ້ອໂຕມາຕາ

ສໍາໜັບກາຍາທີ່ພູ້ດາວນ້ອໂຕມາຕາສາມາຄຣອງຮັບໄດ້ດື່ອ ກາຍາຄອນເທິກ້ອົງພຣີ (Context-free Language) ຜຶ່ງເປັນກາຍາທີ່ໄຟໄຟຕ້ອໂຕມາຕາໄມ່ສ່າມາຄຣອງຮັບໄດ້ ເຫັນ $\{uau : u \in \{a, b\}^*\}$, $\{a^n b^n : n \geq 0\}$ ເປັນຕົ້ນ ສາເຫຫຼື່ຖໍ່ພູ້ດາວນ້ອໂຕມາຕາສາມາຄຣອງຮັບກາຍາຄອນເທິກ້ອົງພຣີໄດ້ ກໍ່ນີ້ອງຈາກມີສແຕ໌ກຈີ່ທຳນ້າທີ່ເກີ່ມຂ້ອງມູນໄດ້ຢ່າງໄມ່ຈຳກັດ

ทำให้สามารถจัดลำดับอักษรภายในสตริงที่ได้อ่านเข้ามาแล้ว จึงไม่ใช่เรื่องยากเลยที่จะสร้างสตริงที่มีลักษณะพิเศษที่ไฟนอลต่อโดยมาตราไม่สามารถกรองรับได้

6.1 พุชดาวน์อโตมาตาที่คาดเดาได้

นิยาม 6.1.1 : พุชดาวน์อโตมาตาที่คาดเดาได้ (Deterministic Pushdown Automata : DPDA) หรือ เรียกย่อ ๆ ว่า DPDA ถูกนิยามดังนี้

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

โดยที่

Q คือ เซตของสถานะภายในหน่วยประมวลผล

Σ คือ เซตของตัวอักษรอนุพต

Γ คือ เซตของสัญลักษณ์ที่ถูกนับรุ่งลงสแต็ก

q_0 คือ สถานะเริ่มต้นของหน่วยประมวลผล โดยที่ $q_0 \in Q$

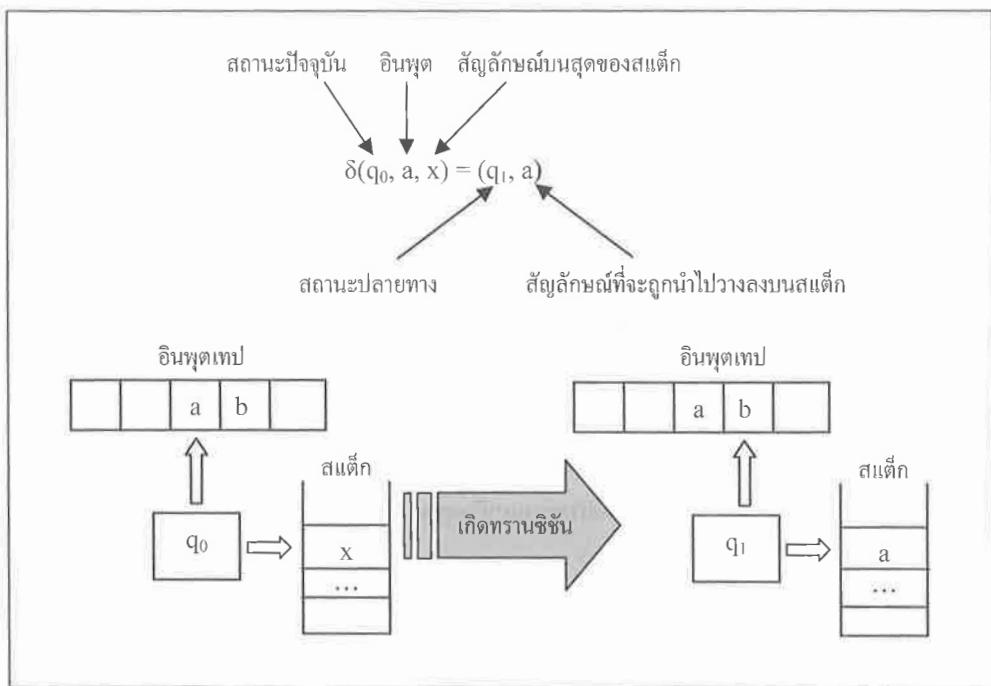
z คือ สัญลักษณ์กำหนดจุดเริ่มต้นของสแต็ก เช่น $z, \$$

F คือ เซตของสถานะสุดท้ายโดยที่ $F \subseteq Q$

δ คือ เซตของทรานซิชันฟังก์ชัน (Transition Function) ซึ่งเป็นความสัมพันธ์ดังนี้

$$Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \longrightarrow Q \times \Gamma^*$$

โดยเมื่อบังคับว่าทางฝั่งขวาของทรานซิชันเดียวกันจะต้องไม่มีทางเลือก เช่น



รูปที่ 6.2 การทำงานของทรานซิชันฟังก์ชัน

จากรูปการทำงานของ DPDA เป็นการบังคับว่าหากสถานะภายในคือ q_0 อินพุตคือ a และส่วนบนสุดของสแต็กคือ x แล้ว DPDA จะต้องเปลี่ยนสถานะไปเป็น q_1 จากนั้นนำ x ออกจากสแต็กและนำตัวอักษร a ใส่ลงในสแต็กแทน แต่ถ้า然是ชิ้นฟังก์ชันมีลักษณะเป็น

$$\delta(q_0, a, x) = \{(q_1, a), (q_1, \epsilon)\}$$

กรณีนี้จะไม่ถือว่าเป็นพุชดาวน์อ็อโตมาตาที่คาดเดาได้ (ไม่ใช่ DPDA นั้นเอง) เนื่องจากที่สถานะเดียวกัน คือ $\delta(q_0, a, x)$ ออกोมาดาสามารถนำตัวอักษร a หรือสตริงว่าง (ϵ) ลงลงในสแต็กได้

นอกจากนี้ หากในเซตของทราบชิ้นฟังก์ชันมีฟังก์ชันซึ่งทางซ้ายมีของเครื่องหมายเท่ากับมีลักษณะคล้ายกัน เช่น

$$\delta(q_0, \epsilon, a) = (q_1, b)$$

$$\delta(q_0, b, a) = (q_2, a)$$

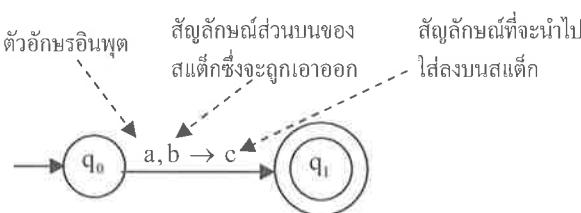
จะเห็นว่าทราบชิ้นแรกไม่จำเป็นต้องอ่านอินพุตจากเทป เนื่องจากสามารถอ่านสตริงว่าง (ϵ) ได้ ส่วนทราบชิ้นที่ 2 จะต้องอ่านอินพุต b จากเทป ดังนั้นออโตมาตาจึงเกิดทางเลือกขึ้น เนื่องจากถ้าสถานะปัจจุบันเป็น q_0 , อินพุตเป็น b และสัญลักษณ์ส่วนบนของสแต็กเป็น a ออโตมาดาสามารถเลือกทำได้ทั้งสองทราบชิ้น เพื่อทำให้ออโตมาดา มีลักษณะเป็นแบบคาดเดาได้ เราจะอนุญาตให้ฟังก์ชันเดียวเท่านั้นที่ข้ามีของทราบชิ้นไม่เป็นเซตว่าง ส่วนฟังก์ชันอื่น ๆ ที่มีสถานะและส่วนบนสุดของสแต็กเหมือนกัน จะต้องถูกบังคับให้ข้ามีของทราบชิ้นเป็นเซตว่าง เช่น

$$\delta(q_0, \epsilon, a) = (q_1, b)$$

$$\delta(q_0, b, a) = \phi$$

การหยุดทำงานของ DPDA สามารถทำได้ 2 วิธี คือ การทำให้สแต็กว่าง และการทำให้หน่วยประมวลผล มีสถานะเป็นสถานะสุดท้าย หนังสือเล่มนี้เน้นเรื่องการทำงานทำให้หน่วยประมวลผลมีสถานะเป็นสถานะสุดท้าย เนื่องจากจะทำความเข้าใจได้ง่ายกว่าการทำให้สแต็กว่าง อีกหนึ่งเหตุผลคือ เมื่อสแต็กว่าง ออโตมาดาจะหยุดทำงานทันที ซึ่งการหยุดทำงานของออโตมาดาจะสามารถดีความໄດ้ 2 กรณี คือ ยอมรับสตริงที่ได้อ่านเข้ามาแล้ว (ในกรณีที่ออโตมาดาได้อ่านอินพุตสตริงเข้ามาแล้วทั้งหมด) หรือไม่อนุรับสตริงที่อยู่บนอินพุตเทป (ในกรณีที่อินพุตสตริงถูกอ่านไม่หมดและยังคงค้างอยู่บนอินพุตเทป)

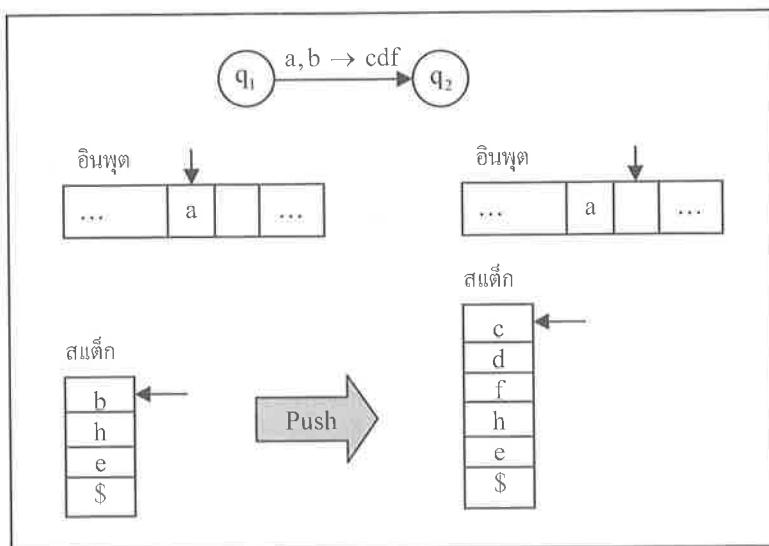
เราสามารถนำทราบชิ้นฟังก์ชันของพุชดาวน์อ็อโตมาตามาเขียนให้อยู่ในรูปทราบชิ้นกราฟ เพื่อให้ง่ายต่อการเข้าใจ (เช่นเดียวกับที่ทำกับทราบชิ้นฟังก์ชันของไฟโนต์อ็อโตมาตา) เช่น $\delta(q_0, a, b) = (q_1, c)$ สามารถแสดงการเปลี่ยนสถานะของพุชดาวน์อ็อโตมาตาที่คาดเดาได้ดังนี้



รูปที่ 6.3 ทราบชิ้นกราฟ

เมื่อสถานะเริ่มต้นคือ q_0 , อินพุตคือ a และลัญลักษณ์บนสุดของสแต็กคือ b พุชดาวน์ออโตมาตาที่คาดเดาได้จะเปลี่ยนสถานะไปเป็น q_1 พร้อมกับนำ b ออกจากสแต็กและใส่ c ลงในสแต็กแทน เพื่อให้เกิดความเข้าใจในการทำงานของออโตมาตา จึงใช้ทราบชิ้นกราฟในการอธิบายตัวอย่างง่ายข้อของบทนี้

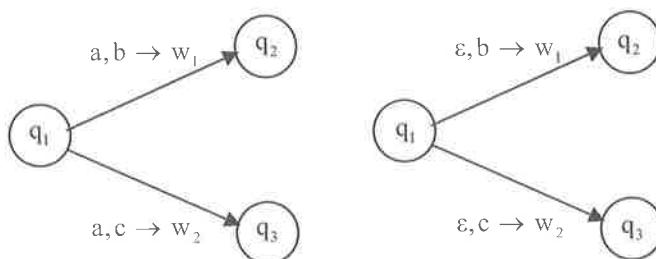
ในการนำลัญลักษณ์ใส่ลงบนสแต็ก สามารถให้ลัญลักษณ์นั้นเป็นสตริงแทนที่จะเป็นตัวอักษร 1 ตัว เช่น



รูปที่ 6.4 การนำสตริงใส่ลงในบันสแต็ก

ทราบชิ้นนี้เป็นการนำตัวอักษร b ออก แล้วใส่สตริง cdf ลงบนสแต็ก ซึ่งตัวอักษรท้ายสุดจะถูกใส่ลงบนสแต็กก่อน แต่อย่างไรก็ตาม สำหรับการนำลัญลักษณ์ออกจากสแต็กนั้นยังคงทำได้เพียงการนำตัวอักษรออกทีละตัว เนื่องจากพุชดาวน์ออโตมาตาไม่มีความสามารถให้มีการนำสตริงออกจากสแต็กทีละหลาย ๆ ตัวอักษร

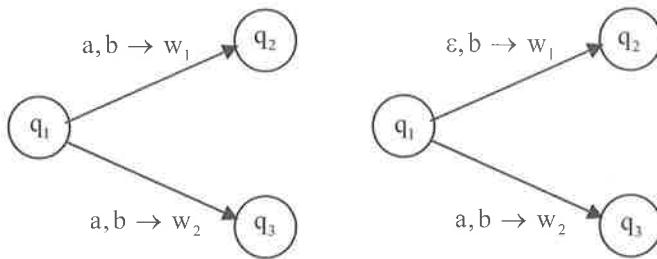
เพื่อเป็นการสร้างความคุ้นเคยกับ DPDA ของผู้เรียนต้นศึกษา ตัวอย่างต่อไปนี้เป็นตัวอย่างของทราบชิ้นที่สามารถเกิดขึ้นได้ใน DPDA



รูปที่ 6.5 ทราบชิ้นที่อนุญาตให้เกิดขึ้นได้ใน DPDA

แม้ว่าจะมีตัวอักษรอินพุต a ที่เหมือนกัน แต่ทรานซิชันกราฟทางซ้ายจะใช้ความแตกต่างกันที่ส่วนบนของสแต็ก โดยหากลัญลักษณ์ที่ส่วนบนของสแต็กเป็น b จะเปลี่ยนสถานะไปเป็น q_2 แต่หากลัญลักษณ์ส่วนบนของสแต็กเป็น c ก็จะเปลี่ยนสถานะไป q_3 สถานการณ์ที่คล้าย ๆ กันเกิดขึ้นกับทรานซิชันกราฟทางขวา โดย DPDA ไม่อ่านอินพุตใด ๆ ระหว่างการเกิดทรานซิชัน ออโตมาตาเลือกที่จะเปลี่ยนสถานะไป q_2 หรือ q_3 โดยสังเกตจากลัญลักษณ์ ณ ส่วนบนของสแต็กซึ่งเป็น b หรือ c

สำหรับตัวอย่างของทรานซิชันที่ไม่อนุญาตให้เกิดขึ้นใน DPDA คือ



รูปที่ 6.6 ทรานซิชันที่ไม่อนุญาตให้เกิดขึ้นใน DPDA

ทรานซิชันกราฟทางซ้ายจะไม่เกิดขึ้นใน DPDA เนื่องจากทั้งสองทางเลือกใช้สถานะ ตัวอักษรอินพุต และลัญลักษณ์ส่วนบนของสแต็กร่วมกัน แต่ให้สถานะปลายทางที่แตกต่างกัน สำหรับทรานซิชันกราฟทางขวาเนื่น ทั้งสองทางเลือกมีสถานะและลัญลักษณ์ส่วนบนของสแต็กเหมือนกัน นอกจากนี้ทรานซิชันกราฟทางขวาสามารถเลือกได้ว่าจะอ่านอินพุต a หรือไม่อ่านก็ได้ (อ่านสตริงว่าง) พฤติกรรมดังกล่าว เป็นพฤติกรรมที่คาดเดาไม่ได้ว่าจะอ่านอินพุต a หรือจะอ่านสตริงว่าง ดังนั้นทรานซิชันกราฟทั้งสองจึงไม่อนุญาตให้มีใน DPDA

ตัวอย่างที่ 6.1 จงสร้าง DPDA เพื่อรับรับภาษา

$$L = \{x^n y^n : n \geq 0\}$$

วิธีทำ : ออกแบบให้ DPDA นี้หยุดทำงานโดยการเข้าสู่สถานะสุดท้าย ซึ่งจะกำหนดให้สถานะสุดท้ายและสถานะเริ่มต้นเป็นสถานะเดียวกัน นั่นคือ q_0 ทั้งนี้เนื่องจากภาษา L ยอมรับสตริงว่าง (ϵ) ดังนั้นหากเริ่มต้นที่ q_0 ถึงแม้ว่าจะไม่มีสตริงป้อนเข้ามาเลย ก็ยังถือว่า DPDA นี้หยุดทำงานที่สถานะสุดท้าย

$$M = (\{q_0, q_1, q_2\}, \{x, y\}, \{z, 1\}, \delta, q_0, z, \{q_0\})$$

$$\delta : \delta(q_0, x, z) = (q_1, 1z),$$

$$\delta(q_1, x, 1) = (q_1, 11),$$

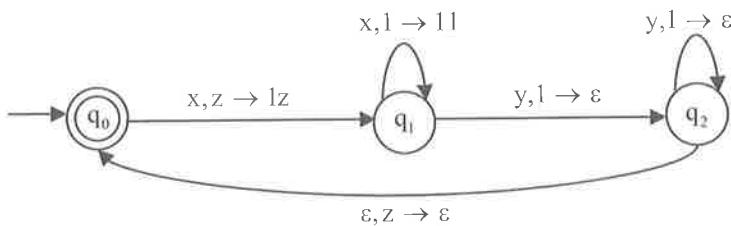
$$\delta(q_1, y, 1) = (q_2, \epsilon),$$

$$\delta(q_2, y, 1) = (q_2, \epsilon),$$

$$\delta(q_2, \epsilon, z) = (q_0, \epsilon)$$

สำหรับหลักการทำงานของ DPDA นี้ เริ่มต้นจากการพบอินพุต x ให้วางสัญลักษณ์ 1 ลงบนสแต็ก (ซึ่งเริ่มต้นด้วยสัญลักษณ์ z) ทำแบบเดียวกันไปเรื่อยๆ จนกว่าจะพบอินพุต y ซึ่งหากส่วนบนของสแต็动能เป็น 1 ก็ให้นำสัญลักษณ์ 1 ออกจากสแต็กพร้อมกับรับอินพุต y ตัวต่อไปเข้ามา ถ้าส่วนบนของสแต็กยังคงเป็น 1 อยู่ก็ให้นำสัญลักษณ์ 1 ออกจากสแต็กส่วนบนของสแต็กเรื่อยๆ จนกว่าอินพุต y จะหมด สุดท้ายหากอินพุตสตริงถูกอ่านจนหมด และสแต็กเหลือเพียงสัญลักษณ์เริ่มต้น z ก็แสดงว่าจำนวนของ x เท่ากับจำนวนของ y ดังนั้น DPDA จึงยอมรับสตริงที่ป้อนเข้ามา

สามารถแทน DPDA ที่ได้โดยใช้กราฟได้อธิบายหนึ่ง ดังนี้



รูปที่ 6.7 DPDA ที่แทนด้วยกราฟ

จากตัวอย่างที่เพิ่งได้ศึกษามาแล้ว ผู้อ่านหลายท่านอาจสงสัยว่ากราฟที่สร้างขึ้นสำหรับกรณีอื่นๆ ที่ไม่ได้เขียนเอาไว้ใน DPDA จะมีความเกี่ยวข้องกับการทำงานของ DPDA หรือไม่ เช่น หากเกิดกรณีที่สถานะเป็น q_1 อินพุตเป็น y และส่วนบนของสแต็กเป็น z จะเกิดอะไรขึ้น นั่นคือ

$$\delta(q_1, y, z) = ?$$

ซึ่งคำตอบก็คือ เชตัวว่าง เนื่องจาก $\delta(q_1, y, z)$ คือเป็นกรณีที่ผิดปกติ (เนื่องจากไม่ได้นิยามไว้ในเซตของกราฟ) และจะไม่ถูกยอมรับโดยภาษา L ดังนั้น

$$\delta(q_1, y, z) = \emptyset$$

อนึ่ง เนื่องจากภาษา $\{x^n y^n : n \geq 0\}$ มี DPDA รองรับ เราจึงสามารถเรียกภาษานี้ว่า ภาษาคอมเพกซ์ฟรีที่คาดเดาได้ (Deterministic Context-free Language)

ตัวอย่างที่ 6.2 จงสร้าง DPDA สำหรับภาษา

$$L = \{wxw^R : w \in \{a, b\}^*\}$$

วิธีทำ :

แนวคิด : รับครึ่งแรกของสตริงเข้ามา ถ้าหากเป็น a ให้ใส่ a ลงบนสแต็ก หรือถ้าเป็น b ก็ใส่ b ลงบนสแต็กจนพบกึ่งกลางของสตริง (ทราบได้โดยตัวอักษร x ที่รับเข้ามา) จากนั้นรับสตริงครึ่งหลังเข้ามาทีละตัว หากตัวอักษรที่รับเข้ามาเหมือนกันกับตัวอักษรที่อยู่ในส่วนบนสุดของสแต็ก ให้นำตัวอักษรที่อยู่บนสุดของสแต็กออก ทำซ้ำไปเรื่อยๆ จนอินพุตหมด และสแต็กเหลือเพียงสัญลักษณ์เริ่มต้น z จึงเข้าสู่สถานะสุดท้าย

$$M = (\{q_0, q_1, q_2, q_f\}, \{a, b, x\}, \{a, b, z\}, \delta, q_0, z, \{q_f\})$$

$$\delta : \delta(q_0, a, z) = (q_1, az),$$

$$\delta(q_0, b, z) = (q_1, bz),$$

$$\delta(q_1, a, a) = (q_1, aa),$$

$$\delta(q_1, b, a) = (q_1, ba),$$

$$\delta(q_1, a, b) = (q_1, ab),$$

$$\delta(q_1, b, b) = (q_1, bb),$$

$$\delta(q_1, x, a) = (q_2, a),$$

$$\delta(q_1, x, b) = (q_2, b),$$

$$\delta(q_2, a, a) = (q_2, \varepsilon),$$

$$\delta(q_2, b, b) = (q_2, \varepsilon),$$

$$\delta(q_2, \varepsilon, z) = (q_f, \varepsilon)$$

จากตัวอย่างนี้ นับเป็นโฉดคือที่ทราบจุดกึ่งกลางของสตริงจากตัวอักษร x แต่ถ้าภาษาไม่มี x เป็นจุดกึ่งกลางสตริง จะไม่สามารถสร้าง DPDA ที่สามารถรับได้เลย ดังนั้นความสามารถของ DPDA จึงลูกจำกด้วยขอบเขตหนึ่ง ในหัวข้อถัดไปจะเป็นการกล่าวถึงพุชดาวน์อ็อโตมาตาอีกประเภทหนึ่งที่มีความสามารถเหนือกว่า DPDA ซึ่งสามารถรองรับภาษา $\{ww^R : w \in \{a, b\}^*\}$ ได้

6.2 พุชดาวน์อ็อโตมาตาที่คาดเดาไม่ได้

พุชดาวน์อ็อโตมาตาที่คาดเดาไม่ได้ (Nondeterministic Pushdown Automata) หรือเรียกย่อ ๆ ว่า NPDA เป็นอ็อโตมาตาที่มีส่วนประกอบคล้ายกับอ็อโตมาตาแบบ DPDA แต่ต่างกันตรงที่ NPDA มีทางเลือกในการเปลี่ยนสถานะการทำงานที่คาดเดาได้ค่อนข้างลำบาก เนื่องจากมีทางเลือกที่จะทำงานมากกว่า 1 ทาง สำหรับนิยามของ NPDA มีดังนี้

นิยาม 6.2.1 : พุชดาวน์อ็อโตมาตาที่คาดเดาไม่ได้ถูกนิยามดังนี้

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

โดยที่

Q คือ เซตของสถานะภายในหน่วยประมวลผล

Σ คือ เซตของตัวอักษรอนุพุต

Γ คือ เซตของสัญลักษณ์ที่ถูกบรรจุลงสแต็ก

δ คือ เซตของทรานซิชันฟังก์ชัน (Transition Function) ซึ่งเป็นความล้มเหลว ดังนี้

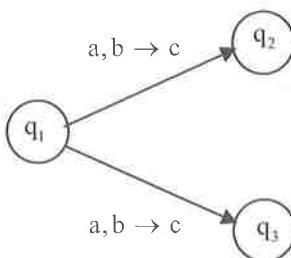
$$Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \longrightarrow Q \times \Gamma^* \quad (\text{โดยไม่มีข้อบังคับใด ๆ})$$

q_0 คือ สถานะเริ่มต้นของหน่วยประมวลผล โดยที่ $q_0 \in Q$

z คือ สัญลักษณ์กำหนดจุดเริ่มต้นของสแต็ก เช่น $z, \$$

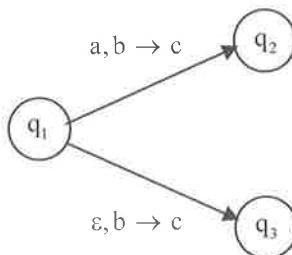
F คือ เซตของสถานะสุดท้าย โดยที่ $F \subseteq Q$

ตัวอย่างของทรานซิชันที่อนุญาตให้เกิดขึ้นใน NPDA (แต่ไม่สามารถเกิดใน DPDA) เช่น



รูปที่ 6.8 ทรานซิชันที่อนุญาตให้เกิดขึ้นใน NPDA

จากรูปที่สถานะ q_1 เมื่ออินพุตเป็น a และส่วนบนของสแต็กเป็น b օอโตมาตาสามารถเลือกเปลี่ยนสถานะได้ 2 ทาง คือ สถานะ q_2 และ q_3 โดยดึงเกตว่าทั้งสองทรานซิชันบังเอิญได้สัญลักษณ์ c ลงไปบนสแต็กซึ่งที่จริงไม่มีความจำเป็นที่จะต้องใส่สัญลักษณ์เหมือนกัน ความคาดเดาไม่ได้ของ NPDA พิจารณาจาก 3 สิ่งเท่านั้นคือ สถานะปัจจุบัน ตัวอักษรอินพุต และสัญลักษณ์ที่ส่วนบนของสแต็ก อย่างไรก็ตาม หากมีทรานซิชันที่รับตัวอักษร อินพุตเป็นสตริงว่าง (ϵ) ความคาดเดาไม่ได้ก็สามารถเกิดขึ้นได้ เช่นกัน ดังอีกด้วยทั้งหนึ่งของทรานซิชันที่อนุญาตให้เกิดขึ้นกับ NPDA ดังนี้



รูปที่ 6.9 ทรานซิชันที่อนุญาตให้เกิดขึ้นใน NPDA

จากตัวอย่างทรานซิชันนี้ NPDA สามารถเลือกได้ว่าจะอ่านอินพุต a และเปลี่ยนสถานะไป q_2 หรือเลือกที่จะไม่อ่านอินพุตอะไรมาก่อนแล้วเปลี่ยนสถานะไป q_3

นอกเหนือจากการแสดงการเปลี่ยนแปลงภายในพื้นที่ของตัวถูกคำนวณแล้ว ยังสามารถแสดงการเปลี่ยนแปลงที่เกิดขึ้นด้วยรูปแบบง่าย ๆ ดังต่อไปนี้

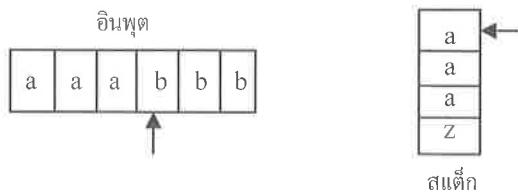
 (q, u, s)

โดยที่ q คือ สถานะปัจจุบันของ PDA

u คือ อินพุตสตริงที่ยังไม่ได้อ่าน

s คือ สัญลักษณ์ทั้งหมดที่บรรจุอยู่ในสแต็ก

เช่น สมมุติว่าพื้นที่ของตัวถูกคำนวณมีสถานะปัจจุบันคือ q_1 และมีอินพุตและสแต็กซึ่งมีตำแหน่งของหัวอ่าน และตัวซึ่งสแต็กดังรูป



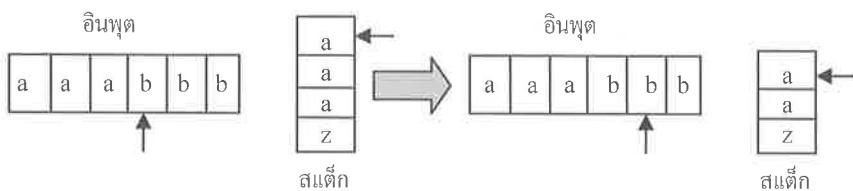
รูปที่ 6.10 สถานะปัจจุบันภายในพุชดาวน์อ็อตโนมาดา

พุชดาวน์อ็อตโนมาดาในรูปนี้สามารถแทนด้วยสัญลักษณ์ $(q_1, bbb, aaaz)$ นั่นคือ ออโตมาตามีสถานะปัจจุบันคือ q_1 ยังเหลืออินพุตสตริง bbb ที่ยังไม่ได้อ่าน และมีสตริงของสัญลักษณ์ $aaaz$ บรรจุอยู่ในสแต็ก เรียกวิธีการอธิบายพุชดาวน์อ็อตโนมาดาแบบนี้ว่า การอธิบาย ณ ช่วงขณะเวลาหนึ่ง (Instantaneous Description)

นอกจากนี้ เราสามารถใช้สัญลักษณ์ \vdash แทนการเกิดทราบซิชันได้อีกด้วย เช่น

$$(q_1, bbb, aaaz) \vdash (q_2, bb, aaz)$$

หมายถึง การเปลี่ยนทราบซิชันของพุชดาวน์อ็อตโนมาดาดังนี้



รูปที่ 6.11 การเปลี่ยนทราบซิชันของพุชดาวน์อ็อตโนมาดา

ตัวอย่างที่ 6.3 จงสร้าง NPDA สำหรับภาษา

$L = \{w \in \{a, b\}^*: \text{จำนวนตัวอักษร } a \text{ ภายในสตริง } w \text{ มีจำนวนเท่ากับจำนวนตัวอักษร } b \text{ ภายในสตริง } w\}$

วิธีทำ :

แนวคิด : เริ่มต้น หากพบอินพุต a และส่วนบนของสแต็กที่ไม่ใช่ b ให้ใส่ a ลงในสแต็ก แต่ถ้าส่วนบนของสแต็กเป็น b ก็หักล้างกันระหว่าง a และ b โดยการนำ b ออกจากส่วนบนของสแต็ก (ในการกรณีของอินพุต b ก็ให้ทำการล้าง a กัน แต่ให้นำตัวอักษร a ออกจากส่วนบนของสแต็กแทน) หากอินพุตหมดและสแต็กเหลือเพียงสัญลักษณ์ z แสดงว่าจำนวนของ a และ b ภายในสตริง w มีจำนวนเท่ากันซึ่งหักล้างกันหมดพอต่อหน่วยประมาณผลก็จะเปลี่ยนสถานะเข้าสู่สถานะสุดท้าย ดังนั้น NPDA สำหรับภาษา L จึงแทนได้ดังนี้

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z\}, \delta, q_0, z, \{q_0\})$$

$$\delta : \delta(q_0, a, z) = (q_1, az),$$

$$\delta(q_0, b, z) = (q_1, bz),$$

$$\delta(q_1, a, a) = (q_1, aa),$$

$$\delta(q_1, b, b) = (q_1, bb),$$

$$\begin{aligned}
 \delta(q_1, a, z) &= (q_1, az), \\
 \delta(q_1, b, z) &= (q_1, bz), \\
 \delta(q_1, a, b) &= (q_1, \varepsilon), \\
 \delta(q_1, b, a) &= (q_1, \varepsilon), \\
 \delta(q_1, \varepsilon, z) &= (q_0, \varepsilon)
 \end{aligned}$$

ผู้ที่เริ่มต้นศึกษาพื้นฐานออโตมาตาจะพบความยากลำบากในการตัดสินใจว่า สถานะของออโตมาตา จะเปลี่ยนแปลงเมื่อใด และบางสถานการณ์ทำไม่ยังสามารถใช้สถานะเดิมได้ แนวทางในการเปลี่ยนสถานะนั้นมีอยู่ว่า จะต้องทราบว่าสถานะที่ถูกสร้างขึ้นนี้นิใช้แทนสถานการณ์อะไรและมีวัตถุประลงค์เพื่ออะไร เช่น จากโจทย์ข้อนี้ เราสร้าง q_0 เพื่อทำหน้าที่คอยรับอินพุตที่ป้อนเข้ามา โดยที่สถานะ q_0 นี้จำนวนของ a และ b จะเท่ากัน แต่ถ้ามีอินพุต a หรือ b เข้ามา ก็จะทำให้จำนวนของ a และ b ไม่เท่ากัน ซึ่งทำให้เราจำเป็นจะต้องเปลี่ยนสถานะไปเป็น q_1 ดังนั้น การเปลี่ยนหรือไม่เปลี่ยนสถานะนั้น ขึ้นอยู่กับสถานการณ์ ณ ตอนนั้นว่ามีการเปลี่ยนแปลงไปหรือมีวัตถุประลงค์ที่เปลี่ยนไปหรือไม่ หากจะทั้งหมดนี้จะเกิดขึ้นเมื่อผ่านการฝึกฝนการแก้ปัญหาพื้นฐานออโตมาตาลักษณะนั้น ขอให้ผู้อ่านศึกษาจากตัวอย่างของหนังสือไปเรื่อยๆ และจะรู้ว่าทักษะในการเปลี่ยนสถานะนั้นไม่ใช่เรื่องยากเลย

ตัวอย่างที่ 6.4 จงสร้าง NPDA สำหรับภาษา

$$L = \{ww^R : w \in \{a, b\}^*\}$$

วิธีทำ :

แนวคิด : วิธีการจัดการกับปัญหานี้จะคล้ายกันกับปัญหา $L = \{wxw^R : w \in \{a, b\}^*\}$ ซึ่งเคยพูดมา ก่อนหน้านี้ แต่แตกต่างกันตรงที่ไม่ทราบจุดกึ่งกลางของสตริงที่แนชัด เนื่องจากไม่มีตัวอักษร x ที่ใช้แสดงตำแหน่ง กึ่งกลางของสตริง เราจะใช้ข้อได้เปรียบของ NPDA ในการจัดการกับกึ่งกลางของสตริง โดยเขียนทรัพย์สัมภพที่ชี้ไปยังจุดที่ต้องการ โดยอินพุตปัจจุบันอาจเป็นตัวอักษรที่อยู่ ณ ตำแหน่งกึ่งกลางของสตริง หรืออาจจะเป็นอินพุตในส่วน ครึ่งแรกของสตริงก็ได้ NPDA สำหรับภาษา L สามารถเขียนได้ดังนี้

$$\begin{aligned}
 M &= (\{q_0, q_1, q_f\}, \{a, b\}, \{a, b, z\}, \delta, q_0, z, \{q_f\}) \\
 \delta : \quad \delta(q_0, \varepsilon, z) &= (q_f, \varepsilon), \quad /* \text{ หากเป็นสตริงว่างให้เข้าสู่สถานะสุดท้ายได้ทันที } * / \\
 \delta(q_0, a, z) &= (q_0, az), \\
 \delta(q_0, b, z) &= (q_0, bz), \\
 \delta(q_0, a, a) &= (q_0, aa), \\
 \delta(q_0, a, b) &= (q_0, ab), \\
 \delta(q_0, b, a) &= (q_0, ba), \\
 \delta(q_0, b, b) &= (q_0, bb), \\
 \delta(q_0, \varepsilon, a) &= (q_1, a), \\
 \delta(q_0, \varepsilon, b) &= (q_1, b), \\
 \delta(q_1, a, a) &= (q_1, \varepsilon), \\
 \delta(q_1, b, b) &= (q_1, \varepsilon), \\
 \delta(q_1, \varepsilon, z) &= (q_f, \varepsilon)
 \end{aligned}$$

} /* จัดการกับครึ่งแรกของสตริง โดยหากเจอ a ให้ใส่ a ลงบนสแต็คหรือหากเจอ b ให้ใส่ b ลงบนสแต็ค */ \\
 } /* เปลี่ยนสถานะเป็น q_1 เมื่อถึงจุดกึ่งกลางสตริง */ \\
 } /* จัดการกับครึ่งหลังของสตริง โดยหากอินพุต กับล่วงบนสุดของสแต็คเมื่อตัวอักษรเหมือนกัน ก็ให้นำตัวอักษรส่วนบนสุดของสแต็คออก */ \\
 } /* เงื่อนไขในการยอมรับสตริง */

หากอินพุตสตริง คือ $abbbbba$ NPDA จะมีขั้นตอนการทำงานดังต่อไปนี้

$$\begin{array}{ll}
 (q_0, abbbbba, z) \xrightarrow{} (q_0, bbbbba, az) & \xrightarrow{} (q_0, bbbba, baz) \\
 \xrightarrow{} (q_0, bba, bbaz) & \xrightarrow{} (q_1, bba, bbaz) \\
 \xrightarrow{} (q_1, ba, baz) & \xrightarrow{} (q_1, a, az) \\
 \xrightarrow{} (q_1, \epsilon, z) & \xrightarrow{} (q_0, \epsilon, \epsilon)
 \end{array}$$

โปรดสังเกตว่า NPDA นี้มีทางเลือกในการเปลี่ยนสถานะ เช่น

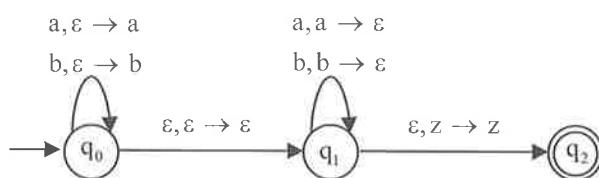
$$\delta(q_0, \epsilon, a) = (q_1, a)$$

$$\text{และ } \delta(q_0, a, a) = (q_0, aa)$$

จากทราบชิ้นฟังก์ชันทั้งสอง NPDA สามารถเลือกได้ว่าจะไม่รับอินพุตใด ๆ จากนั้นเปลี่ยนสถานะเป็น q_1 เพื่อแทนสถานะก่อนคลายของสตริง หรือรับอินพุต a เข้ามาแล้วไฝ่ a ลงในสแต็กเพื่อแทนสถานะก่อนถึงก่อนคลายของสตริง ซึ่งทราบชิ้นฟังก์ชันลักษณะนี้จะเกิดขึ้นกับอโตมาตาแบบ DPDA ไม่ได้

สิ่งหนึ่งที่น่าสนใจจากขั้นตอนการรับสตริง $abbbbba$ ก็คือ หาก NPDA เลือกเปลี่ยนสถานะจาก $(q_0, abbbbba, z)$ ไปเป็น $(q_f, abbbbba, z)$ แม้ว่า NPDA จะเข้าสู่สถานะสุดท้าย q_f แต่อินพุตสตริง $abbbbba$ ยังคงค้างอยู่ในอินพุตเทปซึ่งจะทำให้ NPDA หยุดทำงานไม่ได้ จะต้องย้อนกลับมาเปลี่ยนสถานะจาก $(q_0, abbbbba, z)$ ไปเป็น $(q_0, bbbba, az)$ และทำงานต่อไปจนกว่าอินพุตสตริงจะหมด และเปลี่ยนสถานะเข้าสู่ q_f ออโตมาตาจึงจะหยุดทำงานและยอมรับสตริงนั้น

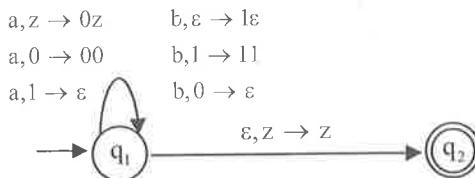
นอกจาก NPDA นี้ที่สร้างขึ้นเพื่อรับภาษา $\{ww^R : w \in \{a, b\}^*\}$ แล้ว เรา yang สามารถสร้าง NPDA ใหม่ซึ่งมีขนาดเล็กกว่า NPDA เดิมได้ โดยใช้คุณสมบัติของสตริงว่าเข้ามาช่วยวิเคราะห์สัญลักษณ์บนสุดของสแต็ก เนื่องจาก NPDA เดิมจะต้องพิจารณาสัญลักษณ์บนสุดของสแต็กว่าเป็นสัญลักษณ์อะไรได้บ้างในทุก ๆ กรณีที่เป็นไปได้ ซึ่งหากพิจารณาสัญลักษณ์บนสุดเป็นสตริงว่างแทนก็จะช่วยลดทราบชิ้นลงไปได้มาก ดังเช่น NPDA ใหม่ซึ่งแสดงโดยทราบชิ้นกราฟดังนี้



รูปที่ 6.12 NPDA ที่ถูกแสดงโดยทราบชิ้นกราฟ

จากรูป NPDA ใส่ตัวอักษรทุกตัวในส่วนครึ่งแรกของอินพุตสตริงลงไปในสแต็ก เมื่อถึงก่อนคลายของอินพุตสตริง (เรารู้ก่อนคลายสตริงโดยอาศัยคุณสมบัติการคาดเดาไม่ได้ของ NPDA โดยใช้ทราบชิ้น $\epsilon, \epsilon \rightarrow \epsilon$) NPDA เปรียบเทียบตัวอักษรของสตริงครึ่งหลังกับสัญลักษณ์ส่วนบนของสแต็ก หากเหมือนกันก็จะนำสัญลักษณ์ส่วนบนสแต็กออก จนกระทั่งอินพุตสตริงครึ่งหลังถูกย่านจนหมดและสแต็กเหลือเพียงสัญลักษณ์เริ่มต้น z NPDA ก็จะยอมรับอินพุตสตริง ww^R โดยเปลี่ยนสถานะเข้าสู่ q_2

ตัวอย่างที่ 6.5 จงจำลองการทำงานของ NPDA ดังต่อไปนี้ ด้วยสตริง abbaab

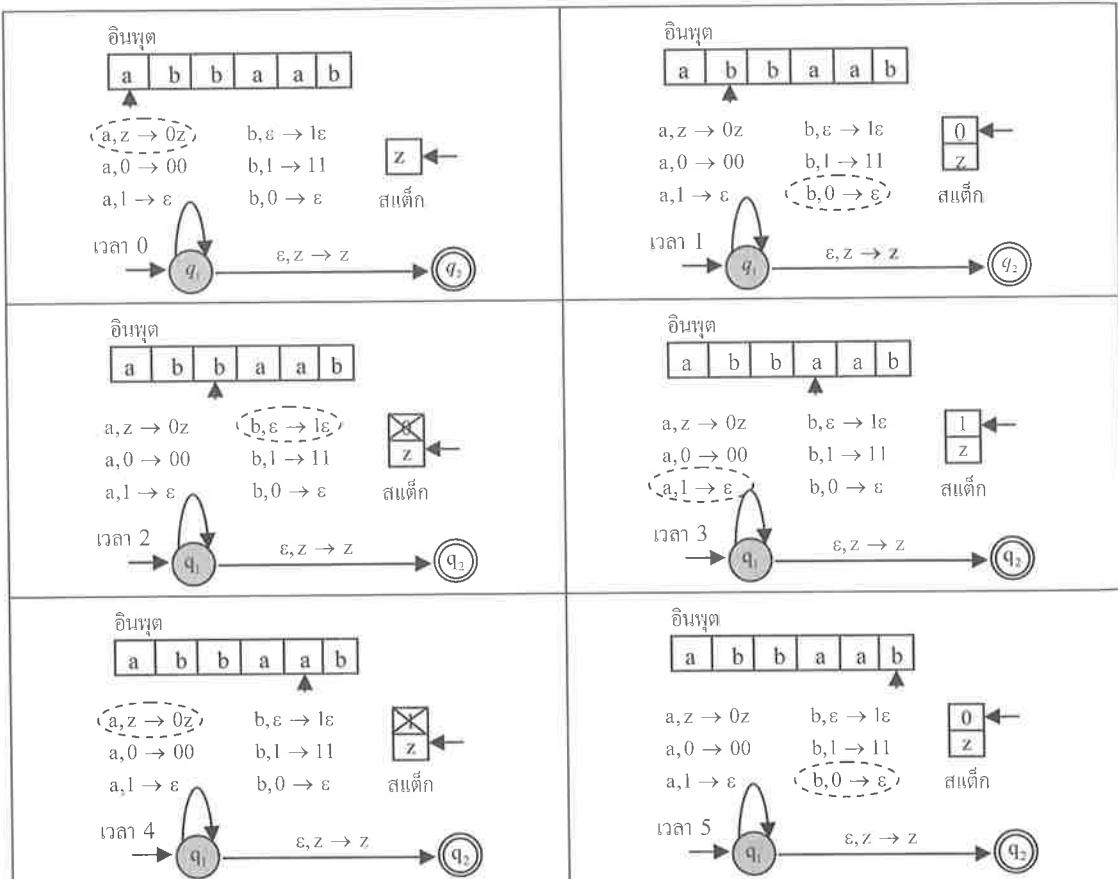


รูปที่ 6.13 NPDA สำหรับภาษา L

วิธีทำ : NPDA นี้รองรับภาษาที่โดยศึกษาแล้วจากตัวอย่างก่อนหน้านี้ นั่นคือ ภาษา

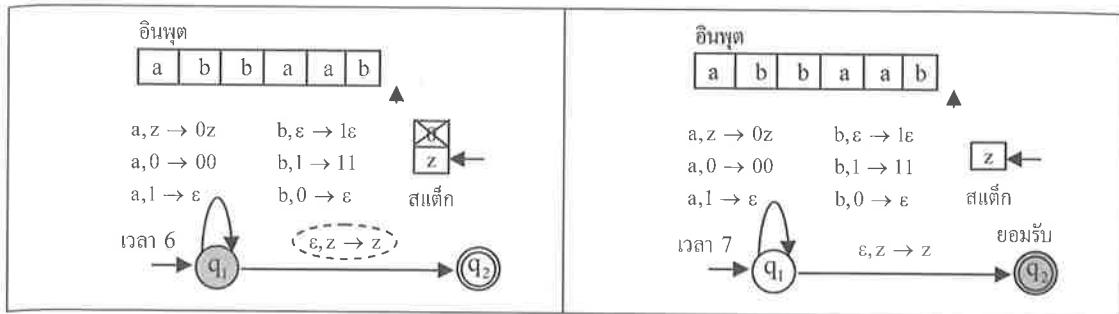
$$L = \{w \in \{a, b\}^*: \text{จำนวนตัวอักษร } a \text{ ภายในสตริง } w \text{ มีจำนวนเท่ากับจำนวนตัวอักษร } b \text{ ภายในสตริง } w\}$$

สำหรับ NPDA นี้จะแตกต่างจากตัวอย่างก่อนหน้านี้เล็กน้อย คือ มีจำนวนทรานซิชันฟังก์ชันน้อยกว่า แต่ มีหลักการทำงานคล้ายเดิม โดยในที่นี้จะแทนสัญลักษณ์ในสแต็กสำหรับตัวอักษร a ด้วย 0 และแทนตัวอักษร b ด้วย 1 เราสามารถจำลองการทำงานของ NDPA นี้โดยใช้สตริง abbaab ได้ดังนี้



รูปที่ 6.14 การทำงานของ NPDA ที่เวลาต่าง ๆ

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)



รูปที่ 6.14 (ต่อ)

ตัวอย่างที่ 6.6 จงสร้าง NPDAs สำหรับภาษาดังต่อไปนี้

$L = \{w \in \{a, b\}^*: \text{จำนวนตัวอักษร } a \text{ ภายในสตริง } w \text{ มีจำนวนเป็นสองเท่าของจำนวนตัวอักษร } b \text{ ภายในสตริง } w\}$

วิธีทำ :

แนวคิด : ภาษาที่ประกอบด้วยสตริงที่มีจำนวนของ a เป็นสองเท่าของจำนวนของ b ดังนั้นจะต้องสร้าง NPDAs ที่สามารถหักล้างระหว่าง a จำนวน 2 ตัวกับ b จำนวน 1 ตัว โดยเราจะใช้ 2 สถานะแทนสถานะที่แตกต่างกันคือ q_0 ใช้แทนสถานะที่ NPDAs ได้รับตัวอักษร a เข้ามาเป็นจำนวนคู่ และ q_1 แทนสถานะที่ NPDAs ได้รับตัวอักษร a เข้ามาเป็นจำนวนคี่ NPDAs สำหรับภาษา L สามารถเขียนได้ดังนี้

$$M = (\{q_0, q_1, q_f\}, \{a, b\}, \{a, b, z\}, \delta, q_0, z, \{q_f\})$$

$$\delta : \delta(q_0, \epsilon, z) = (q_f, z),$$

$\delta(q_0, a, z) = (q_1, z), \quad /* \text{ เมื่อ } a \text{ ตัวแรกให้เปลี่ยนสถานะโดยไม่ต้องกระทำการใด ๆ กับสแต็ค */$

$$\delta(q_0, a, a) = (q_1, a),$$

$$\delta(q_0, a, b) = (q_1, b),$$

$$\delta(q_0, b, b) = (q_0, bb),$$

$$\delta(q_0, b, a) = (q_0, \epsilon),$$

$$\delta(q_0, b, z) = (q_0, bz), \quad /* \text{ หากเจอ } b \text{ ให้ใส่ } b \text{ ลงบนสแต็ค */}$$

$$\delta(q_1, a, z) = (q_0, az), \quad /* \text{ ที่ } q_1 \text{ หากเจอตัวอักษร } a \text{ อีก ให้ใส่ } a \text{ ลงบนสแต็ค } \\ \text{พร้อมทั้งกลับสู่สถานะ } q_0 */$$

$$\delta(q_1, b, z) = (q_1, bz),$$

$$\delta(q_1, a, b) = (q_0, \epsilon),$$

$$\delta(q_1, a, a) = (q_0, aa),$$

$$\delta(q_1, b, b) = (q_1, bb),$$

$$\delta(q_1, b, a) = (q_1, \epsilon)$$

หากป้อนอินพุตสตริง คือ aaabab NPDA จะมีขั้นตอนการทำงานดังต่อไปนี้

$$\begin{aligned}
 (q_0, aaabab, z) &\xrightarrow{} (q_1, aabab, z) \xrightarrow{} (q_0, abab, az) \\
 &\xrightarrow{} (q_1, bab, az) \xrightarrow{} (q_1, ab, z) \\
 &\xrightarrow{} (q_0, b, az) \xrightarrow{} (q_0, \epsilon, z) \\
 &\xrightarrow{} (q_f, \epsilon, z)
 \end{aligned}$$

ตัวอย่างที่ 6.7 จงสร้าง NPDA สำหรับภาษาดังต่อไปนี้

$L = \{w \in \{a,b\}^*: N_a(w) \geq N_b(w) - 1 \text{ และ } N_a(v) \geq N_b(v)\} \cup \{b\}$ โดยที่ $N_a(s)$ และ $N_b(s)$ คือจำนวนตัวอักษร a และ b ภายในสตริง s ตามลำดับ และ v คือ สตริงย่อยส่วนหน้า (Prefix) ใด ๆ ของสตริง w ซึ่ง $|v| < |w|$

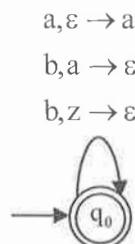
วิธีทำ : ก่อนที่จะเริ่มสร้าง NPDA สำหรับภาษานี้ เราควรทดลองสร้างสตริงซึ่งเป็นสมาชิกในภาษา L ออกมารูปแบบตัวอย่างทีละสตริง เพื่อให้เกิดความเข้าใจในภาษามากขึ้น

$$\begin{aligned}
 L = \{ &\epsilon, a, b, aa, aaa, \dots, ab, abb, aab, aabb, abab, aabb, aaab, aaabb, aaabbb, \\
 &aaabbbb, abaaaabbbbbbab, ababaababbaabbbb, \dots \}
 \end{aligned}$$

ภาษาสตริงที่ทดลองสร้างขึ้น เราจะสังเกตได้ว่าจำนวนของตัวอักษร b จะต้องมีจำนวนไม่เกินจำนวนของตัวอักษร a มากด้วย 1 ซึ่งนำมารูปแบบนี้ในการสร้าง NPDA ดังนี้

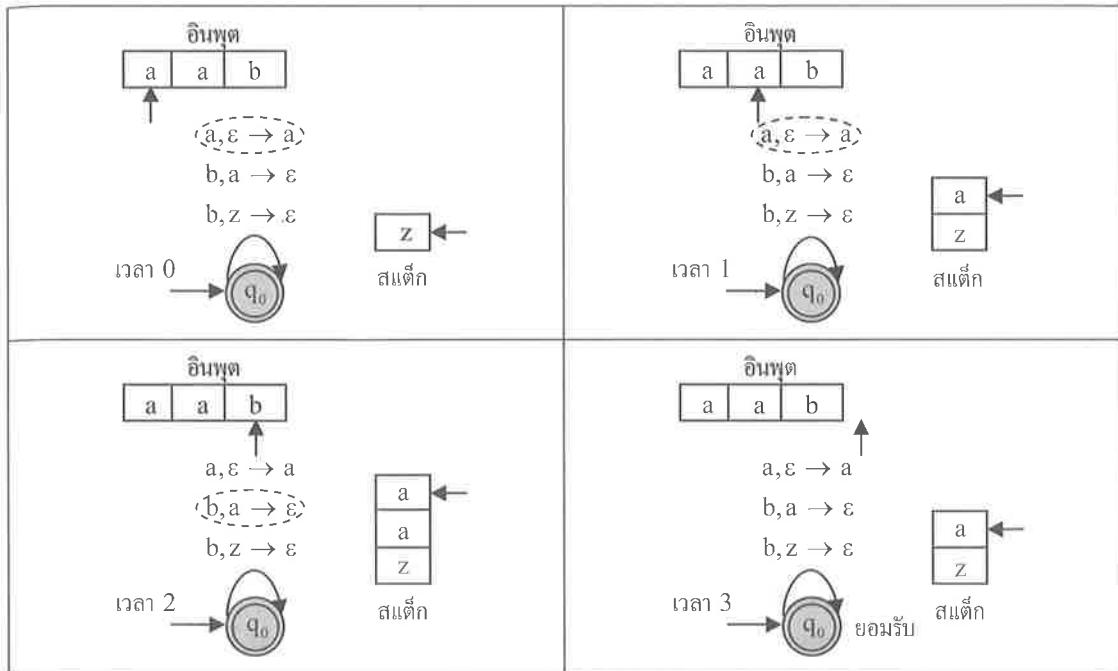
แนวคิด : เมื่อพอนอินพุต a ให้ใส่ a ลงไปบนสแต็ก จนกระทั่งพอบอินพุต b ให้ตรวจสอบส่วนบนของสแต็กว่าเป็น a หรือไม่ เพื่อจะหักล้างระหว่างอินพุต b และสัญลักษณ์บนสแต็กซึ่งเป็น a หลังจากที่หักล้างกันจนกระทั่งสัญลักษณ์ a หมดไปจากสแต็ก ทำให้สแต็กเหลือสัญลักษณ์เริ่มต้น z อยู่ตัวเดียว ให้อ่านอินพุต b ตัวต่อไปและนำสัญลักษณ์เริ่มต้นสแต็ก z ออก ซึ่งจะทำให้สแต็กกว่างและส่งผลให้ NPDA หยุดทำงาน (Halt) หากไม่มีอินพุตเหลืออยู่บนอินพุตเทป สตริงที่อ่านเข้ามาจะถูกยอมรับโดย NPDA แต่ถ้ายังเหลืออินพุตสตริงอยู่บนเทป NPDA ก็จะปฏิเสธอินพุตสตริงนั้น

NPDA สำหรับภาษา L สามารถแทนได้ด้วยกราฟดังนี้



รูปที่ 6.15 NPDA สำหรับภาษา L

สามารถจำลองการทำงานของ NDPA นี้โดยใช้สตริง aab ดังนี้



รูปที่ 6.16 การทำงานของ NPDA ที่เวลาต่าง ๆ

ตัวอย่างที่ 6.8 จงสร้าง PDA สำหรับภาษาดังต่อไปนี้

$$L = \{a^n b^{n+m} c^m : n \geq 0, m \geq 1\}$$

วิธีทำ :

- แนวคิด :
- (1) รับอินพุต a เข้ามาทีละตัว จากนั้นไป่ a ลงบนสแต็ก
 - (2) รับอินพุต b เข้ามาทีละตัว ถ้าข้างบนของสแต็กเป็น a ให้นำ a ออกจากสแต็ก จากนั้น ทำซ้ำจนกระทั่งสแต็กว่างแล้วให้ใส่ b ที่ยังเหลืออยู่ในอินพุตสตริงลงสแต็กทีละตัว
 - (3) รับอินพุต c เข้ามาทีละตัว ถ้าข้างบนของสแต็กเป็น b ให้นำ b ออกจากสแต็ก จนกระทั่ง อินพุตหมดและสแต็กเหลือเพียงสัญลักษณ์ริมต้น z พร้อมกันจึงเข้าสู่สถานะสุดท้าย

PDA สำหรับภาษา L สามารถเขียนได้ดังนี้

$$M = (\{q_0, q_1, q_2, q_f\}, \{a, b, c\}, \{a, b, c, z\}, \delta, q_0, z, \{q_f\})$$

$$\delta : \delta(q_0, a, z) = (q_0, az),$$

$$\delta(q_0, a, a) = (q_0, aa),$$

$$\delta(q_0, b, z) = (q_1, bz),$$

$$\delta(q_0, b, a) = (q_1, \epsilon),$$

$$\delta(q_1, b, a) = (q_1, \epsilon),$$

$$\delta(q_1, b, z) = (q_1, bz),$$

$$\delta(q_1, b, b) = (q_1, bb),$$

$$\delta(q_1, c, b) = (q_2, \epsilon),$$

$$\delta(q_2, c, b) = (q_2, \epsilon),$$

$$\delta(q_2, \epsilon, z) = (q_f, \epsilon)$$

หากป้อนอินพุตสตริง คือ abbc PDA จะมีขั้นตอนการทำงานดังต่อไปนี้

$$(q_0, abbc, z) \xrightarrow{} (q_0, bbc, az) \xrightarrow{} (q_1, bc, z)$$

$$\xrightarrow{} (q_1, c, bz) \xrightarrow{} (q_2, \epsilon, z)$$

$$\xrightarrow{} (q_f, \epsilon, \epsilon)$$

6.3 พุทธานน์อโตมาตาที่คาดเดาไม่ได้ยอมรับภาษาคอนเทกซ์ฟรี

ทฤษฎี 6.3.1 : ภาษาที่ถูกยอมรับโดยพุทธานน์อโตมาตาที่คาดเดาไม่ได้ (NPDA) เป็นภาษาคอนเทกซ์ฟรี

พิสูจน์ : จากทฤษฎีนี้ กล่าวได้อีกนัยหนึ่งคือ เราต้องการพิสูจน์ว่า

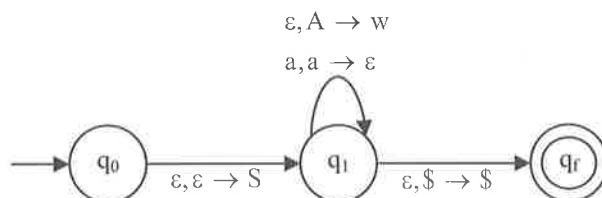
ภาษาคอนเทกซ์ฟรี = ภาษาที่ถูกยอมรับโดย NPDA

วิธีการพิสูจน์ประกอบด้วย 2 ขั้นตอน คือ

- พิสูจน์ว่าภาษาคอนเทกซ์ฟรี \subseteq ภาษาที่ถูกยอมรับโดย NPDA
นั่นคือ จะต้องแปลงไวยากรณ์ของภาษาคอนเทกซ์ฟรี G ไปเป็น NPDA M โดยที่ $L(G) = L(M)$
- พิสูจน์ว่าภาษาคอนเทกซ์ฟรี \supseteq ภาษาที่ถูกยอมรับโดย NPDA
นั่นคือ จะต้องแปลง NPDA M ให้เป็นไวยากรณ์ของภาษาคอนเทกซ์ฟรี G โดยที่ $L(G) = L(M)$

1. พิสูจน์ว่าภาษาคอนเทกซ์ฟรี \subseteq ภาษาที่ถูกยอมรับโดย NPDA พิสูจน์โดยการสร้าง NPDA M จากไวยากรณ์คอนเทกซ์ฟรี G ได้ดังนี้

- (1) สร้างสถานะเริ่มต้น q_0 และเชื่อมสถานะ q_0 Majority สถานะกลาง q_1 โดยใช้tranชิ้น $\epsilon, \epsilon \rightarrow S$
- (2) สำหรับโปรดักชันใด ๆ ที่อยู่ในรูปแบบ $A \rightarrow w$ และสำหรับทุก ๆ สัญลักษณ์เทอร์มินอล a ให้สร้างtranชิ้น $\epsilon, A \rightarrow w$ และ $a, a \rightarrow \epsilon$
- (3) จากสถานะ q_1 สร้างtranชิ้น Majority สถานะสุดท้าย q_f โดยใช้tranชิ้น $\epsilon, \$ \rightarrow \$$ เมื่อ $\$$ คือ สัญลักษณ์แสดงจุดเริ่มต้นของสแตก



รูปที่ 6.17 รูปแสดง NPDA เพื่อพิสูจน์ขั้นตอนที่ 1

เช่น สมมุติว่าภาษาค่อนเท็กซ์ฟรีมีไวยากรณ์ค่อนเท็กซ์ฟรี G :

$$S \rightarrow aSTb$$

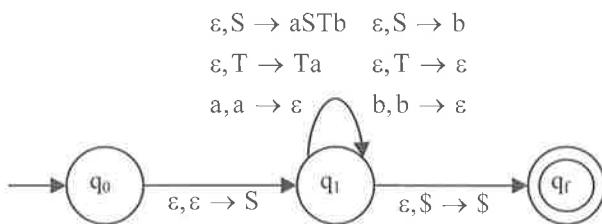
$$S \rightarrow b$$

$$T \rightarrow Ta$$

$$T \rightarrow \epsilon$$

เนื่องจากสัญลักษณ์เทอร์มินอลของไวยากรณ์นี้เป็นตัวอักษร a และ b ดังนั้นจึงใช้สัญลักษณ์ $\$$ แทนจุดเริ่มต้นของสแต็กเพื่อหลีกเลี่ยงความสับสนที่เกิดกับสัญลักษณ์ z

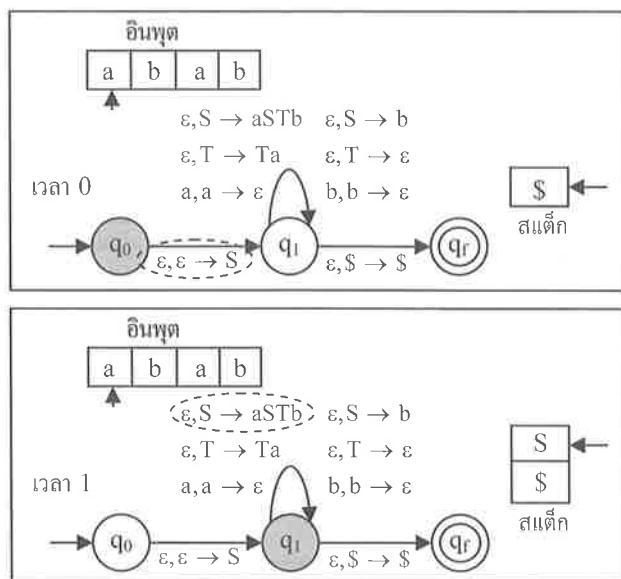
จากขั้นตอนทั้ง 3 ข้อ เราสามารถสร้าง NPDA M ได้ดังนี้



รูปที่ 6.18 NPDA M ที่ได้จากไวยากรณ์ค่อนเท็กซ์ฟรี G

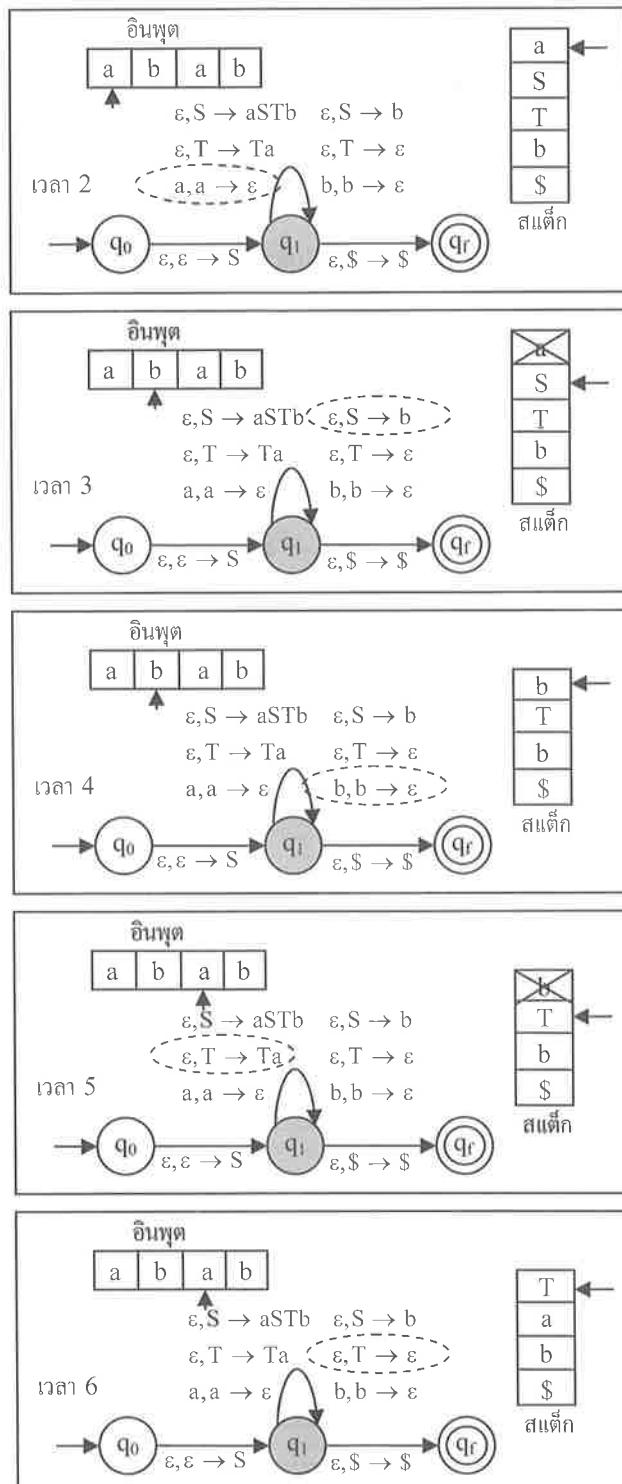
เพื่อทำความเข้าใจการทำงานของ NPDA M จะยกตัวอย่างอินพุต $abab$ ซึ่งสตั๊กนี้สามารถสร้างได้จากไวยากรณ์ค่อนเท็กซ์ฟรี G โดยจะแสดงการได้มาจากการซ้ายก่อนดังนี้

$$S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab \Rightarrow abab$$

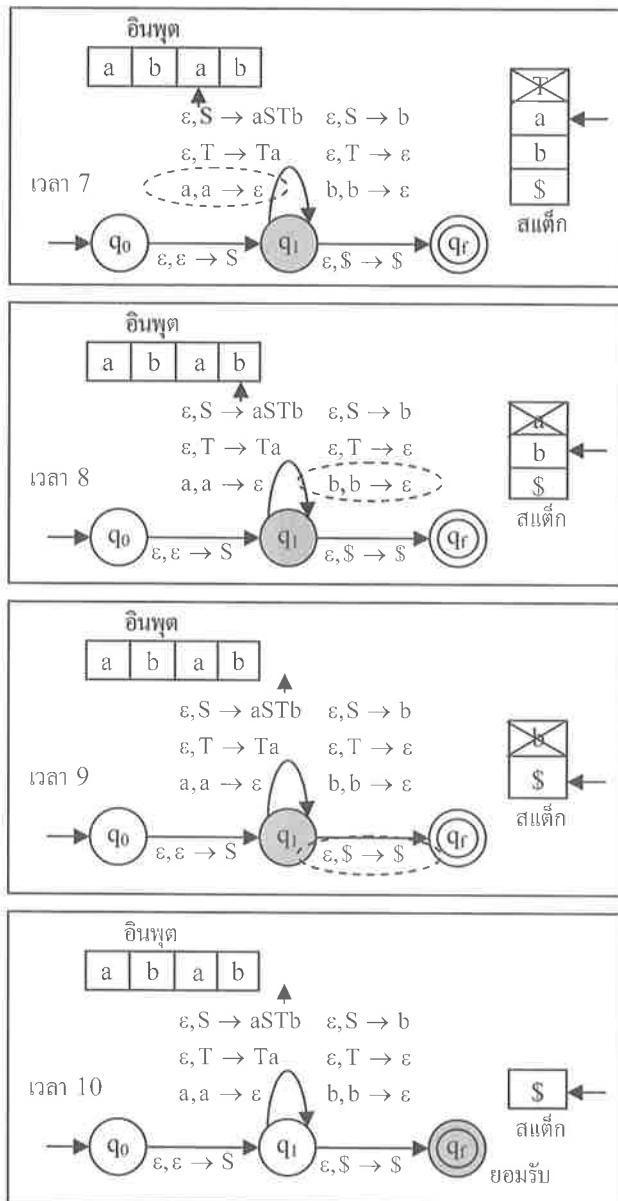


รูปที่ 6.19 การทำงานของ NPDA M ที่เวลาต่าง ๆ

(ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)



រូបថត 6.19 (ពេល)

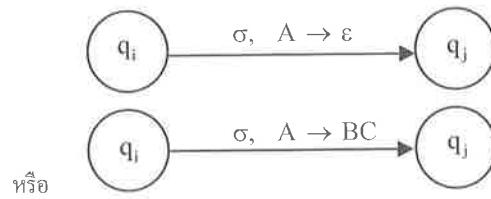


รูปที่ 6.19 (ต่อ)

จากการที่สามารถสร้าง NPDA M จากไวยากรณ์คอนเท็กซ์ฟรี G ได้ จึงสรุปได้ว่าสำหรับทุก ๆ ไวยากรณ์ คอนเท็กซ์ฟรี G เราสามารถสร้าง NPDA M ขึ้นมารองรับ โดยที่ $L(G) = L(M)$ ได้เสมอ

2. พิสูจน์ว่าภาษาคอนเท็กซ์ฟรี \subseteq ที่ถูกยอมรับโดย NPDA

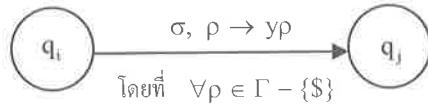
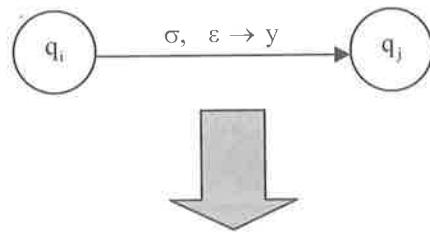
พิสูจน์โดยการสร้างไวยากรณ์คอนเท็กซ์ฟรี G จาก NPDA M โดยที่ $L(M) = L(G)$ โดยก่อนอื่นจะต้องแปลงทรานซิชันที่อยู่ใน NPDA ให้อยู่ในรูปมาตรฐานนี่ดังต่อไปนี้



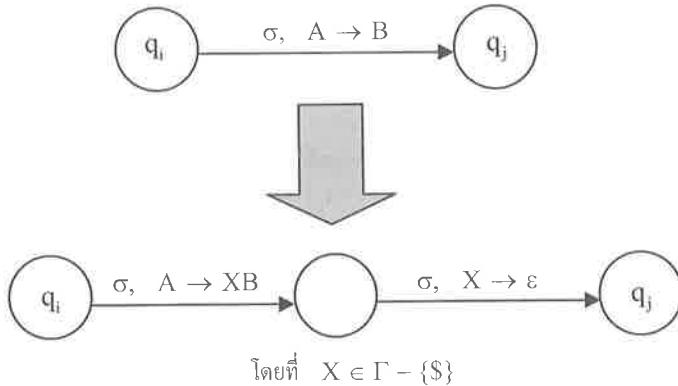
โดยที่ A, B, C คือ สัญลักษณ์ภายในสแต็ก

หากทราบซิชันมีรูปแบบอื่นนอกเหนือจากทั้ง 2 รูปแบบนี้ เราสามารถแปลงได้ดังนี้

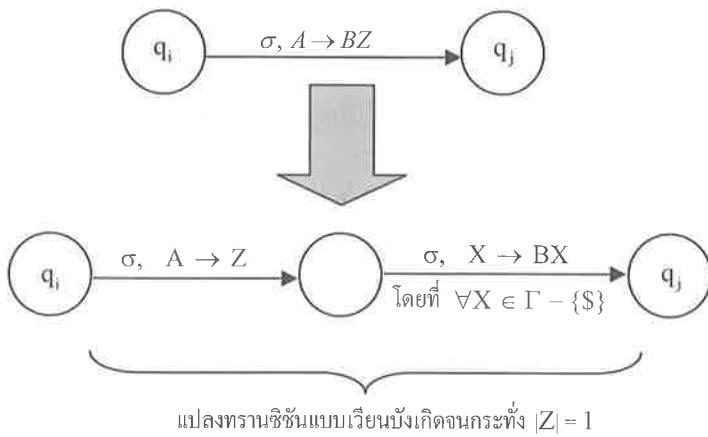
- $\delta(q_i, \sigma, \varepsilon) = (q_j, y)$



- $\delta(q_i, \sigma, A) = (q_j, B)$

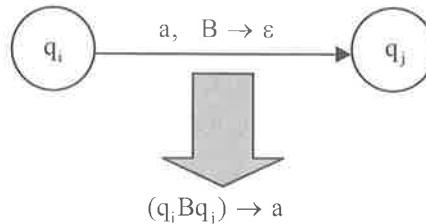


- $\delta(q_i, \sigma, A) = (q_j, BZ)$ โดยที่ $|Z| \geq 2$

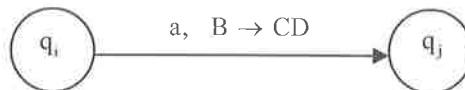


หลังจากแปลงทรานซิชันของ NPDA ให้อยู่ในรูปแบบที่กำหนดแล้ว นำทรานซิชันที่ได้มาสร้างไวยากรณ์ គอนเทกซ์ฟรี ตามขั้นตอนดังต่อไปนี้

- สำหรับทรานซิชัน $\delta(q_i, a, B) = (q_j, \varepsilon)$ ให้สร้างโปรดักชัน $(q_i B q_j) \rightarrow a$ โดยสัญลักษณ์ (q_iBq_j) หมายถึง สัญลักษณ์ตัวแปร (Variable) หรือ Nonterminal Symbol) 1 ตัว

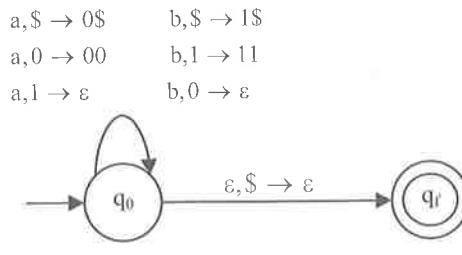


- สำหรับทรานซิชัน $\delta(q_i, a, B) = (q_j, CD)$ ให้สร้างโปรดักชัน $(q_i B q_k) \rightarrow a(q_j C q_l)(q_l D q_k)$ สำหรับทุก ๆ สถานะ q_i และ q_k ที่เป็นไปได้ทั้งหมดใน NPDA



- ให้สัญลักษณ์ตัวแปรเริ่มต้น (Start Symbol) คือ $(q_0 \$ q_f)$ โดยที่ q₀ และ q_f คือสถานะเริ่มต้นและสถานะสุดท้ายของ NPDA ตามลำดับ

เช่น สมมุติว่ามี NPDA M ดังรูป



รูปที่ 6.20 NPDA M

สามารถแปลง NPDA M ให้กลายเป็นไวยากรณ์ค่อนเทิร์กช์ฟรีได้ดังนี้

- $\delta(q_0, a, 1) = (q_0, \epsilon)$ แปลงเป็น $(q_0 | q_0) \rightarrow a$
- $\delta(q_0, b, \$) = (q_0, 1\$)$ แปลงเป็น
 $(q_0 \$ q_0) \rightarrow b(q_0 | q_0)(q_0 \$ q_0) \mid b(q_0 | q_f)(q_f \$ q_0)$
 $(q_0 \$ q_f) \rightarrow b(q_0 | q_0)(q_0 \$ q_f) \mid b(q_0 | q_f)(q_f \$ q_f)$
- $\delta(q_0, \epsilon, \$) = (q_f, \epsilon)$ แปลงเป็น $(q_0 \$ q_f) \rightarrow \epsilon$
- ทำเช่นเดียวกันกับทรานซิชันที่เหลือ จนได้ไวยากรณ์สุดท้ายดังนี้
 $(q_0 \$ q_0) \rightarrow b(q_0 | q_0)(q_0 \$ q_0) \mid b(q_0 | q_f)(q_f \$ q_0)$
 $(q_0 \$ q_f) \rightarrow b(q_0 | q_0)(q_0 \$ q_f) \mid b(q_0 | q_f)(q_f \$ q_f)$
 $(q_0 | q_0) \rightarrow b(q_0 | q_0)(q_0 | q_0) \mid b(q_0 | q_f)(q_f | q_0)$
 $(q_0 | q_f) \rightarrow b(q_0 | q_0)(q_0 | q_f) \mid b(q_0 | q_f)(q_f | q_f)$
 $(q_0 \$ q_0) \rightarrow a(q_0 0 q_0)(q_0 \$ q_0) \mid a(q_0 0 q_f)(q_f \$ q_0)$
 $(q_0 \$ q_f) \rightarrow a(q_0 0 q_0)(q_0 \$ q_f) \mid a(q_0 0 q_f)(q_f \$ q_f)$
 $(q_0 0 q_0) \rightarrow a(q_0 0 q_0)(q_0 0 q_0) \mid a(q_0 0 q_f)(q_f 0 q_0)$
 $(q_0 0 q_f) \rightarrow a(q_0 0 q_0)(q_0 0 q_f) \mid a(q_0 0 q_f)(q_f 0 q_f)$
 $(q_0 | q_0) \rightarrow a$
 $(q_0 0 q_0) \rightarrow b$
 $(q_0 \$ q_f) \rightarrow \epsilon$

ตัวอย่างการได้มาสำหรับอินพุตสตริง abba กับไวยากรณ์ค่อนเทิร์กช์ฟรีที่ได้ สามารถแสดงได้ดังนี้

$$(q_0 \$ q_f) \Rightarrow a(q_0 0 q_0)(q_0 \$ q_f) \Rightarrow ab(q_0 | q_f)(q_0 \$ q_f) \Rightarrow abb(q_0 | q_0)(q_0 \$ q_f) \Rightarrow abba(q_0 \$ q_f) \Rightarrow abba$$

จากวิธีการแปลง NPDA ไปเป็นไวยากรณ์ค่อนเทิร์กช์ฟรีที่ได้เสนอมา สรุปได้ว่าสำหรับ NPDA M ใด ๆ สามารถแปลงให้กลายเป็นไวยากรณ์ค่อนเทิร์กช์ฟรี G โดยที่ $L(M) = L(G)$ ได้เสมอ

6.4 ความสัมพันธ์ระหว่าง NPDA และ DPDA

ทฤษฎี 6.4.1 : พุชดาวน์อัตโนมัติค่าเดาไม่ได้ (NPDA) สามารถรองรับภาษาค่อนเท็กซ์พรีได้มากกว่า พุชดาวน์อัตโนมัติค่าเดาได้ (DPDA)

พิสูจน์ : จากทฤษฎีนี้ กล่าวได้อีกนัยหนึ่งว่าเราต้องการพิสูจน์

$$\text{DPDA} \subseteq \text{NPDA}$$

จะพิสูจน์โดยยกตัวอย่างให้เห็นว่า มีภาษาค่อนเท็กซ์พรีบางภาษาซึ่งสามารถสร้าง NPDA รองรับได้ แต่ไม่สามารถสร้าง DPDA รองรับภาษานั้นได้ โดยใช้วิธีการพิสูจน์แบบขัดแย้ง (Contradiction Proof) ด้วยข้อต่อไปนี้

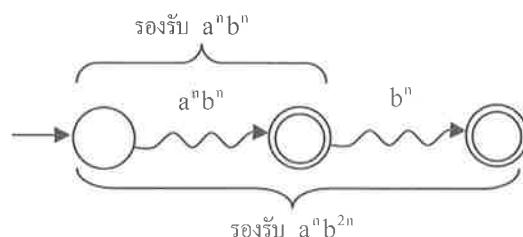
ก่อนอื่น ควรทดสอบดูว่ามี NPDA ซึ่งสามารถรองรับภาษา L ได้หรือไม่ เนื่องจากภาษา L เกิดจาก 2 ภาษาeasy เนียนกัน ดังนั้นจะต้องสร้าง NPDA 2 ส่วน ซึ่งแต่ละส่วนรองรับแต่ละภาษา ส่วนแรกรองรับ $\{a^n b^n\}$ สามารถสร้าง NPDA ขึ้นมารองรับได้ โดยเมื่อพบอินพุต a ให้ใส่ a ลงบนสแต็กไปเรื่อย ๆ จนกระทั่งพบอินพุต b ก็ให้หักล้างกับสัญลักษณ์ a ซึ่งอยู่บนสแต็กโดยการนำ a ออกจากสแต็กไปเรื่อย ๆ จนอินพุตถูกอ่านหมดและสแต็กเหลือเพียงสัญลักษณ์เริ่มต้น $\$$ ให้เปลี่ยนสถานะเข้าสู่สถานะสุดท้าย และยอมรับอินพุตสตริงนั้น

สำหรับส่วนที่ 2 รองรับ $\{a^n b^{2n}\}$ สามารถสร้าง NPDA ขึ้นมารองรับได้ โดยใช้วิธีที่คล้ายกับส่วนแรก แต่แตกต่างกันตรงที่ เมื่อพบอินพุต a ให้เราใส่ aa ลงบนสแต็ก จากนั้นก็หักล้าง b กับ a ไปเรื่อย ๆ จนกว่าอินพุตจะหมดและสแต็กเหลือเพียงสัญลักษณ์เริ่มต้น $\$$ ก็ให้เปลี่ยนสถานะเข้าสู่สถานะสุดท้าย และยอมรับอินพุตสตริงนั้น จากนั้นเชื่อม NPDA ทั้งสองส่วนด้วย ϵ -transition เพื่อให้สามารถเลือกได้ว่าจะเข้าสู่ NPDA ส่วนแรกหรือส่วนที่ 2 โดยไม่ต้องอ่านอินพุตและไม่ต้องเปลี่ยนแปลงค่าในสแต็ก ดังนั้นจะเห็นว่าสามารถสร้าง NPDA เพื่อรองรับภาษา L ได้จริง

ขั้นตอนต่อไปเป็นการพิสูจน์แบบขัดแย้ง โดยตั้งสมมุติฐานเพื่อให้เกิดการขัดแย้งว่า

“สมมุติให้มี DPDA M ซึ่งสามารถรองรับภาษา $L = \{a^n b^n\} \cup \{a^n b^{2n}\}$ ได้”

ดังนั้นจากสมมุติฐานนี้ ไม่สนใจรายละเอียดภายใน DPDA M นี้ โดยจะแทนทวนซิชันภายใน M ด้วยเส้นหมาย ดังรูป



รูปที่ 6.21 DPDA M

เพื่อให้เกิดประโยชน์ต่อการพิสูจน์ จะขอนำความรู้ในบทดังไปมาใช้ในการพิสูจน์ดังนี้

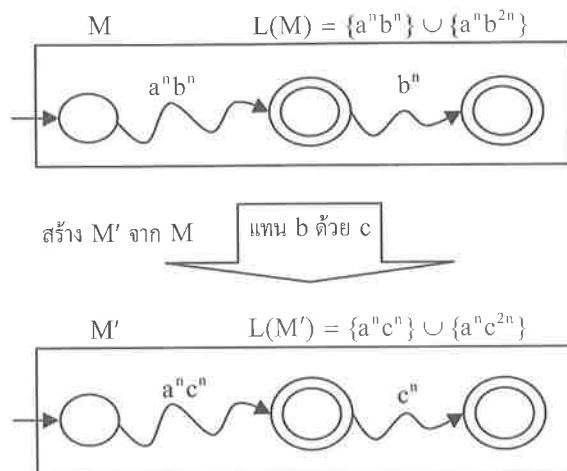
1. ภาษา $a^n b^n c^n$ ไม่ใช่ภาษาค่อนเท็กซ์ฟรี (เราจะพิสูจน์ในบทต่อไป)

2. เนื่องจากภาษา $L = \{a^n b^n\} \cup \{a^n b^{2n}\}$ เป็นภาษาค่อนเท็กซ์ฟรี หากเราคำนวณ L หมายเนี่ยนกับภาษา $a^n b^n c^n$ ผลที่ได้จะไม่เป็นภาษาค่อนเท็กซ์ฟรีด้วย นั่นคือ

$L \cup a^n b^n c^n$ ไม่ใช่ภาษาค่อนเท็กซ์ฟรี

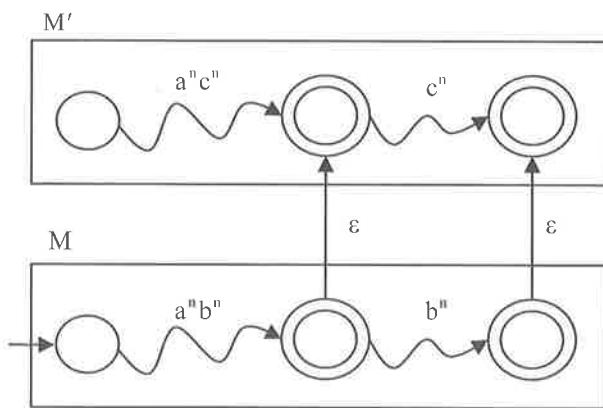
ดังนั้นจะต้องไม่มี NPDA ใด ๆ ที่รับภาษา $L \cup a^n b^n c^n$ ได้อย่างเด็ดขาด เนื่องจากภาษาดังกล่าวไม่ใช่ภาษาค่อนเท็กซ์ฟรี หากเราสามารถสร้าง NPDA ที่รับภาษา $L \cup a^n b^n c^n$ ได้ ก็แสดงว่าสมมุติฐานที่เราตั้งไว้นั้นผิด

ขั้นตอนต่อไป จะสร้าง DPDA M' โดยแทนตัวอักษร b ด้วยตัวอักษร c เพื่อให้ DPDA M' รับภาษา $\{a^n c^n\} \cup \{a^n c^{2n}\}$



รูปที่ 6.22 การแปลงจาก M ไปเป็น M'

จากนั้นเชื่อม DPDA M และ DPDA M' เข้าด้วยกันโดยใช้tranซิชันสติงว่าง (ϵ -transition) กลายเป็น NPDA และให้สถานะเริ่มต้นของ M เป็นสถานะเริ่มต้นของ NPDA ดังนี้

รูปที่ 6.23 NPDA ใหม่ที่เกิดจากการประกอบกันของ M และ M'

จะเห็นได้ชัดเจนว่า NPDA รูปนี้สามารถรองรับภาษา $a^n b^n c^n$ ได้ ซึ่งทราบมาก่อนหน้านี้แล้วว่าไม่มี NPDA ใด ๆ สามารถรองรับภาษา $a^n b^n c^n$ ได้ ดังนั้นสมมุติฐานที่ตั้งไว้ตอนต้นจึงผิด ทำให้สรุปได้ว่าไม่มี DPDA M ซึ่งสามารถรองรับภาษา $L = \{a^n b^n\} \cup \{a^n b^{2n}\}$ ได้ นั่นคือ $DPDA \subseteq NPDA$ นั่นเอง

แบบฝึกหัด

1. DPDA และ NPDA มีความแตกต่างกันอย่างไร
สำหรับข้อ 2 - 8 จงสร้างพุชดาวน์อ็อโตมาตาที่รับภาษาดังต่อไปนี้
 2. $L = \{a^n b^{2n} \mid n \geq 0\}$
 3. $L = \{w \in \{a, b\}^* \mid n_a(w) > n_b(w)\}$
 4. $L = \{w \in \{a, b\}^* \mid n_a(w) \neq n_b(w)\}$
 5. $L = \{a^n b^m a^{n+m} \mid n, m \geq 1\}$
 6. $L = \{010(010)^m 1(10)^m \mid m \geq 0\}$
 7. $L = \{w \in \{a, b\}^* \mid n_a(w) + 1 < n_b(w)\}$
 8. $L = \{a^n b^m \mid n \leq m \leq 2n\}$
9. จงอธิบายว่าเหตุใดเราจึงไม่สามารถสร้างพุชดาวน์อ็อโตมาตาสำหรับภาษา L ดังต่อไปนี้ได้
 $L = \{ww^R w \mid w \in \{a, b\}^*\}$
10. จงหาภาษาที่ถูกกรองรับโดย NPDA M ดังต่อไปนี้

$$M = (\{q_0, q_1, q_f\}, \{0, 1\}, \{0, 1, z\}, \delta, q_0, z, \{q_f\})$$

$$\delta : \begin{aligned} \delta(q_0, 0, z) &= \{(q_1, 0), (q_f, \epsilon)\}, \\ \delta(q_1, 1, 1) &= (q_1, 1), \\ \delta(q_1, 1, 0) &= (q_1, 1), \\ \delta(q_1, 0, 1) &= (q_f, 1) \end{aligned}$$

การพิสูจน์ความไม่เป็นคุณสมบัติปิดของภาษาเรกูลาร์และวิธีการพิสูจน์ความไม่เป็นภาษาเรกูลาร์

และคุณสมบัติของภาษาคุณสมบัติปิดของภาษาเรกูลาร์

ในบทที่ 4 ได้ศึกษาเกี่ยวกับคุณสมบัติปิดของภาษาเรกูลาร์และวิธีการพิสูจน์ความไม่เป็นภาษาเรกูลาร์ แม้แล้ว เนื้อหาในบทนี้จะมีความคล้ายคลึงกับเนื้อหาในบทที่ 4 แต่ต่างกันที่ระดับของภาษาที่มีความซับซ้อนมากขึ้น ภาษาที่จะกล่าวถึงในบทนี้ คือ ภาษาคุณสมบัติปิดของภาษาเรกูลาร์ (Context-free Language)

ภาษาคุณสมบัติปิดของภาษาเรกูลาร์ เป็นภาษาที่เป็นชูปอร์เซต (Super Set) ของภาษาเรกูลาร์ แต่ก็มีได้หมายความว่าภาษาคุณสมบัติปิดของภาษาเรกูลาร์จะเป็นภาษาที่ใหญ่ที่สุด เมื่อจากยังมีบางภาษาที่ภาษาคุณสมบัติปิดของภาษาเรกูลาร์ไม่สามารถครอบคลุมไปถึงได้ ดังนั้น จึงเป็นเรื่องที่จำเป็นที่จะต้องทราบคุณสมบัติของภาษาคุณสมบัติปิดของภาษาเรกูลาร์ และวิธีการพิสูจน์ความไม่เป็นภาษาคุณสมบัติปิดของภาษาเรกูลาร์

7.1 คุณสมบัติปิดของภาษาคุณสมบัติปิดของภาษาเรกูลาร์

ภาษาคุณสมบัติปิดของภาษาเรกูลาร์ มีคุณสมบัติปิดภายใต้การกระทำดังต่อไปนี้

- ยูเนียน (Union)
- การต่อ กัน (Concatenation)
- การทำซ้ำ (Star-closure)

พิสูจน์ :

1. ยูเนียน : สมมุติให้ L_1 และ L_2 เป็นภาษาคุณสมบัติปิดโดย

L_1 มีไวยากรณ์ $G_1 = (V_1, T_1, S_1, P_1)$ และ

L_2 มีไวยากรณ์ $G_2 = (V_2, T_2, S_2, P_2)$

หากเรานำ L_1 มา yuniean กับ L_2 เพื่อสร้างภาษาใหม่ L_3 เราจะได้ไวยากรณ์ของภาษา L_3 ดังนี้

$$G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, S_3, P_1 \cup P_2 \cup \{S_3 \rightarrow S_1|S_2\})$$

เนื่องจาก G_3 อยู่ในรูปแบบไวยากรณ์คุณสมบัติปิดของภาษาเรกูลาร์ ดังนั้นภาษา L_3 จึงเป็นภาษาคุณสมบัติปิดของภาษาเรกูลาร์ เช่นเดียวกับภาษา L_1 และ L_2

2. การต่อ กัน : สมมุติให้ L_1 และ L_2 เป็นภาษาคุณสมบัติปิดที่แนบด้วยไวยากรณ์ G_1 และ G_2 ตามลำดับ

$$G_1 = (V_1, T_1, S_1, P_1)$$

$$G_2 = (V_2, T_2, S_2, P_2)$$

หาก L_3 เป็นภาษาที่เกิดจากการนำ L_1 และ L_2 มาเชื่อมต่อกัน ไวยากรณ์ของ L_3 จะถูกแทนด้วย G_3 ดังนี้

$$G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, S_3, P_1 \cup P_2 \cup \{S_3 \rightarrow S_1S_2\})$$

ซึ่งจะเห็นได้ว่า G_3 เป็นไวยากรณ์แบบค่อนเท็กซ์ฟรี ดังนั้นจึงทำให้ L_3 เป็นภาษาค่อนเท็กซ์ฟรี

3. การทำซ้ำ : สมมุติให้ L_1 และ G_1 เป็นดังเช่นเดียวกับการพิสูจน์ใน 2 ข้อแรก สามารถทำการทำซ้ำของภาษา L_1 ได้ด้วยไวยากรณ์ดังต่อไปนี้

$$G_s = (V_1 \cup \{S_s\}, T_1, S_s, P_1 \cup \{S_s \rightarrow S_1S_s | \epsilon\})$$

นั่นคือ $L(G_1)^* = L(G_s)$

เนื่องจาก G_s เป็นไวยากรณ์แบบค่อนเท็กซ์ฟรี ดังนั้น $L(G_1)^*$ จึงเป็นภาษาค่อนเท็กซ์ฟรี

ตัวอย่างที่ 7.1 กำหนดให้

$$L = \{a^n b^n\} \cup \{ww^R\}$$

จะพิสูจน์ว่าภาษา L เป็นภาษาค่อนเท็กซ์ฟรี

พิสูจน์ : กำหนดให้ $L_1 = \{a^n b^n\}$

$$L_2 = \{ww^R\}$$

พื้นภาษา L_1 และ L_2 ต่างก็เป็นภาษาค่อนเท็กซ์ฟรี เนื่องจากสามารถสร้างไวยากรณ์ค่อนเท็กซ์ฟรี G_1 (แทนภาษา L_1) และไวยากรณ์ค่อนเท็กซ์ฟรี G_2 (แทนภาษา L_2) ได้ดังนี้

$$G_1 : S_1 \rightarrow aS_1b | \epsilon$$

$$G_2 : S_2 \rightarrow aS_2a | bS_2b | \epsilon$$

เมื่อนำภาษา L_1 และ L_2 มาบูนียนกัน ผลที่ได้จะเป็นภาษา L

$$L = \{a^n b^n\} \cup \{ww^R\} = L_1 \cup L_2$$

สามารถสร้างไวยากรณ์สำหรับภาษา L ได้โดยการนำไวยากรณ์ G_1 และ G_2 มาบูนียนกันดังนี้

$$G : S \rightarrow S_1 | S_2$$

$$S_1 \rightarrow aS_1b | \epsilon$$

$$S_2 \rightarrow aS_2a | bS_2b | \epsilon$$

เนื่องจาก $L(G) = L$ ดังนั้นภาษา L จึงเป็นภาษาค่อนเท็กซ์ฟรี

นอกจากนี้ โดยอาศัยวิธีการพิสูจน์ความเป็นค่อนเท็กซ์ฟรีที่คล้ายกัน เราสามารถพิสูจน์ว่าภาษา $\{a^n b^n\}. \{ww^R\}$ เป็นภาษาค่อนเท็กซ์ฟรีโดยสร้างไวยากรณ์สำหรับภาษาดังกล่าวจากการนำไวยากรณ์ของภาษา L_1 และ L_2 มาต่อกัน (Concatenation) ดังนี้

$$S \rightarrow S_1S_2$$

$$S_1 \rightarrow aS_1b | \epsilon$$

$$S_2 \rightarrow aS_2a | bS_2b | \epsilon$$

ตัวอย่างที่ 7.2 กำหนดให้

$$L = \{a^n b^n\}^*$$

งพิสูจน์ว่าภาษา L เป็นภาษาคุณเท็กซ์ฟรี

พิสูจน์ : ภาษา L เกิดจากการทำซ้ำภาษา $\{a^n b^n\}$ ตั้งแต่ 0 ครั้งขึ้นไป สำหรับไวยากรณ์ของภาษา $\{a^n b^n\}$ แสดงไว้ในตัวอย่างก่อนหน้านี้ซึ่งเป็นไวยากรณ์คุณเท็กซ์ฟรี สามารถนำไวยากรณ์นี้มาสร้างไวยากรณ์สำหรับภาษา L ได้ดังนี้

$$S \rightarrow S_1 S \mid \epsilon$$

$$S_1 \rightarrow a S_1 b \mid \epsilon$$

เนื่องจากไวยากรณ์ของภาษา L เป็นไวยากรณ์คุณเท็กซ์ฟรี จึงทำให้ภาษา L เป็นภาษาคุณเท็กซ์ฟรี

ตัวอย่างที่ 7.3 กำหนดให้

$$L = \{a^n b^n c^m d^m : n \geq 0, m \geq 0\}$$

งพิสูจน์ว่าภาษา L เป็นภาษาคุณเท็กซ์ฟรี

พิสูจน์ : กำหนดให้ $L_1 = \{a^n b^n : n \geq 0\}$

$$L_2 = \{c^m d^m : m \geq 0\}$$

จากความรู้ในบทที่ผ่านมา เราทราบกันแล้วว่าทั้ง L_1 และ L_2 ต่างก็เป็นภาษาคุณเท็กซ์ฟรี ดังนั้น $L_1 L_2$ จะต้องเป็นภาษาคุณเท็กซ์ฟรีด้วย

$$L_1 L_2 = \{a^n b^n c^m d^m : n, m \geq 0\}$$

เนื่องจาก $L = L_1 L_2$ ดังนั้นภาษา L จึงเป็นภาษาคุณเท็กซ์ฟรี

ตัวอย่างที่ 7.4 งพิสูจน์ว่าภาษาคุณเท็กซ์ฟรีไม่มีคุณสมบัติปิดภายใต้การอินเตอร์เซกชัน (Intersection)

พิสูจน์ : เมื่อจากการพิสูจน์ในตัวอย่างนี้เป็นการพิสูจน์เชิงลง วิธีหนึ่งที่นิยมใช้ในการพิสูจน์คือการยกตัวอย่างภาษาเพื่อแสดงให้เห็นว่าคุณสมบัติปิดภายใต้การอินเตอร์เซกชันของภาษาคุณเท็กซ์ฟรีไม่มีอยู่จริง ตัวอย่างภาษาที่จะถูกนำมาพิสูจน์คือ

$$L_1 = \{a^n b^n c^m\}$$

$$L_2 = \{a^n b^m c^n\}$$

ทั้งภาษา L_1 และ L_2 ต่างก็เป็นภาษาคุณเท็กซ์ฟรี เนื่องจากเราสามารถสร้างไวยากรณ์คุณเท็กซ์ฟรีขึ้นมา รองรับได้ ดัง

$$S \rightarrow AC$$

$$S \rightarrow AB$$

$$G_1 : A \rightarrow aAb \mid \epsilon$$

$$G_2 : A \rightarrow aA \mid \epsilon$$

$$C \rightarrow cC \mid \epsilon$$

$$B \rightarrow bBc \mid \epsilon$$

โดยที่ $L(G_1) = L_1$ และ $L(G_2) = L_2$

.

ภาษา L_1 บังคับว่าจำนวนตัวอักษร a และ b จะต้องมีจำนวนเท่ากัน ส่วนภาษา L_2 นั้นบังคับว่าจำนวนตัวอักษร b และ c นั้นจะต้องมีจำนวนเท่ากัน เมื่อนำภาษาทั้งสองมาทำการอินเตอร์เซกชันจะได้ภาษาที่บังคับว่าจำนวนตัวอักษร a จะต้องเท่ากับจำนวนตัวอักษร b และจำนวนตัวอักษร b จะต้องเท่ากับจำนวนตัวอักษร c นั่นคือ

$$\begin{aligned} L_1 \cap L_2 &= \{a^n b^n c^m\} \cap \{a^n b^m c^m\} \\ &= \{a^n b^n c^n\} \end{aligned}$$

เนื่องจากภาษา $\{a^n b^n c^n\}$ ไม่ใช่ภาษาค่อนเทิกซ์ฟรี (ผู้อ่านจะได้ศึกษาการพิสูจน์ว่าภาษา $\{a^n b^n c^n\}$ ไม่ใช่ภาษาค่อนเทิกซ์ฟรีในหัวข้อที่ 7.4 ปัจจุบันมาสำหรับภาษาค่อนเทิกซ์ฟรี ของบทนี้) ดังนั้นภาษาค่อนเทิกซ์ฟรีจึงไม่มีคุณสมบัติปิดภายใต้การอินเตอร์เซกชัน

ตัวอย่างที่ 7.5 จงพิสูจน์ว่าภาษาค่อนเทิกซ์ฟรีไม่มีคุณสมบัติปิดภายใต้การคอมพลีเม้นต์ (Complement)

พิสูจน์ : การพิสูจน์สำหรับตัวอย่างนี้สามารถทำได้ง่าย ๆ โดยการใช้กฎของเดอมอร์แกน (Demorgan's Laws) กำหนดให้ภาษา L_1 และ L_2 เป็นภาษาค่อนเทิกซ์ฟรี

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

เนื่องจากภาษาค่อนเทิกซ์ฟรีไม่มีคุณสมบัติปิดภายใต้การอินเตอร์เซกชัน ดังนั้นด้านขวาของสมการก็ต้องไม่มีคุณสมบัตินี้เช่นกัน แต่เนื่องจากทราบมาก่อนหน้านี้แล้วว่าภาษาค่อนเทิกซ์ฟรีไม่มีคุณสมบัติปิดภายใต้การยูเนียน ดังนั้นໂອเปอร์เรเตอร์ตัวเดียวที่ทำให้ข้ามมือของสมการไม่มีคุณสมบัติปิดก็คือ คอมพลีเม้นต์ ดังนั้นเราจึงสรุปได้ว่า ภาษาค่อนเทิกซ์ฟรีไม่มีคุณสมบัติปิดภายใต้การคอมพลีเม้นต์

ตัวอย่างที่ 7.6 จงพิสูจน์ว่าภาษาค่อนเทิกซ์ฟรีไม่มีคุณสมบัติปิดภายใต้การทำผลต่าง (Difference)

พิสูจน์ : กำหนดให้

L_1 และ L_2 เป็นภาษาค่อนเทิกซ์ฟรีได ๆ

เนื่องจาก

$$L_1 - L_2 = L_1 \cap \overline{L_2}$$

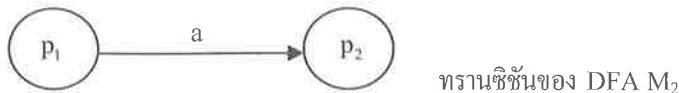
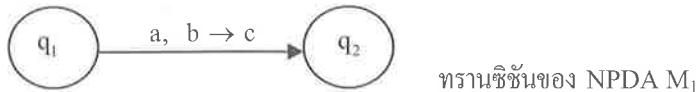
เราทราบว่าภาษาค่อนเทิกซ์ฟรีไม่มีคุณสมบัติปิดภายใต้การอินเตอร์เซกชันและคอมพลีเม้นต์ ดังนั้นจึงสรุปได้ว่า $L_1 - L_2$ ไม่ใช่ภาษาค่อนเทิกซ์ฟรี

7.2 การอินเตอร์เซกชันระหว่างภาษาค่อนเทิกซ์ฟรีและภาษาเรกูลาร์

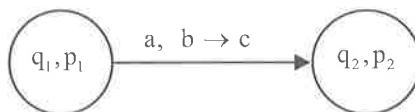
ทฤษฎี 7.2.1 : กำหนดให้ L_1 เป็นภาษาเรกูลาร์ และ L_2 เป็นภาษาค่อนเทิกซ์ฟรี ผลที่ได้จาก $L_1 \cap L_2$ จะเป็นภาษาค่อนเทิกซ์ฟรี

พิสูจน์ : กำหนดให้ M_1 เป็น NPDA ซึ่งรองรับภาษา L_1 และ M_2 เป็น DFA ซึ่งรองรับภาษา L_2 สามารถพิสูจน์ทฤษฎีนี้โดยการสร้าง NPDA M ซึ่งสามารถรองรับได้ทั้งภาษา L_1 และภาษา L_2 ได้ วิธีการสร้าง NPDA M คือการยุบรวม NPDA M_1 และ DFA M_2 เข้าด้วยกันโดยแยกออกเป็นกรณีได้ดังนี้ .

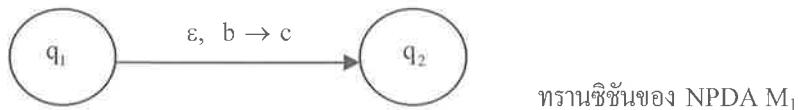
กรณีที่ 1 : สำหรับtranซึ้นของ NPDA M_1 และ DFA M_2 ซึ่งมีลักษณะเป็น



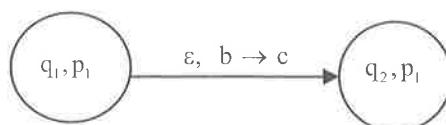
สามารถสร้างtranซึ้นใหม่ของ NPDA M ซึ่งจะรองรับทั้งtranซึ้นของ M_1 และ M_2 ได้ดังนี้



กรณีที่ 2 : สำหรับtranซึ้นของ NPDA M_1 และ DFA M_2 ซึ่งมีลักษณะเป็น



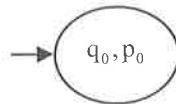
สามารถสร้างtranซึ้นใหม่ของ NPDA M ซึ่งรองรับทั้งtranซึ้นของทั้ง M_1 และ M_2 ได้ดังนี้



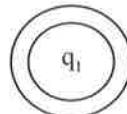
กรณีที่ 3 : สำหรับสถานะเริ่มต้นของ NPDA M_1 และ DFA M_2



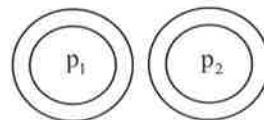
สร้างสถานะเริ่มต้นใหม่ของ NPDA M ซึ่งใช้เป็นสถานะเริ่มต้นร่วมกันทั้ง M_1 และ M_2 ดังนี้



กรณีที่ 4 : สำหรับสถานะสุดท้ายของ NPDA M_1 และ DFA M_2

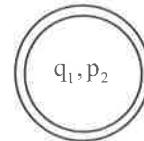
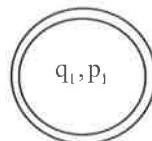


สถานะสุดท้ายของ NPDA M_1



สถานะสุดท้ายของ DFA M_2

สร้างสถานะสุดท้ายของ NPDA M ซึ่งสามารถรองรับทั้ง M_1 และ M_2 ได้ดังนี้



จากกรณีที่เป็นไปได้ทั้งหมด 4 กรณี จะเห็นได้ว่าเราสามารถสร้าง NPDA M ซึ่งสามารถรองรับการทำงานของทั้ง NPDA M_1 และ DFA M_2 ดังนั้นจึงเป็นการพิสูจน์ได้ว่าการอนุเคราะห์ชันระหว่างภาษาค่อนเทิกซ์ฟรีและภาษาเรกูลาร์ ผลที่ได้จะเป็นภาษาค่อนเทิกซ์ฟรีอย่างแน่นอน

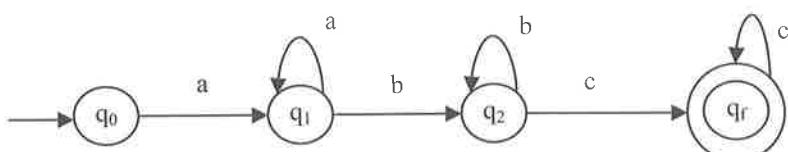
ตัวอย่างที่ 7.7 จงแสดงว่าภาษา L ดังต่อไปนี้ไม่ใช่ภาษาค่อนเทิกซ์ฟรี

$$L = \{w \in \{a, b, c\}^* : n_a(w) = n_b(w) = n_c(w)\}$$

พิสูจน์ : กำหนดให้

$$L_1 = \{a^+ b^+ c^+\}$$

เราทราบว่า L_1 เป็นภาษาเรกูลาร์ เพราะสามารถเขียนໄไฟน์ตอโอมາตาได้ดังนี้



รูปที่ 7.1 ໄไฟน์ตอโอมາตาสำหรับภาษา L_1

หาก L เป็นภาษาค่อนเทิกซ์ฟรี $L \cap L_1$ จะต้องเป็นภาษาค่อนเทิกซ์ฟรีด้วย

แต่ $L \cap L_1 = \{a^n b^n c^n : n \geq 1\}$ ซึ่งเราทราบมาก่อนหน้านี้แล้วว่าภาษา $a^n b^n c^n$ ไม่ใช่ภาษาค่อนเทิกซ์ฟรี ดังนั้นจึงสรุปได้ว่าภาษา L ไม่ใช่ภาษาค่อนเทิกซ์ฟรี

ตัวอย่างที่ 7.8 จงพิสูจน์ว่าภาษา L ดังต่อไปนี้เป็นภาษาค่อนเทิกซ์ฟรี

$$L = \{a^n b^n : n \neq 100, n \geq 0\}$$

พิสูจน์ : กำหนดให้

$$L_1 = \{a^n b^n : n \geq 0\}$$

$$\text{และ } L_2 = \{a^{100} b^{100}\}$$

ภาษา L_1 เป็นภาษาค่อนเทิกซ์ฟรี เนื่องจากมีพืชดาวน์อโตมาตราองรับดังที่ผ่านมาแล้ว ส่วน L_2 เป็นภาษาเรกูลาร์ เนื่องจากสามารถเขียนไฟในต่ออโตมาตราองรับได้ แม้จะมีขนาดใหญ่มากก็ตาม นอกจากนี้ $\overline{L_2}$ ยังเป็นภาษาเรกูลาร์อันเนื่องมาจากคุณสมบัติปิดของภาษาเรกูลาร์ภายใต้การคอมพลีเมนต์

$$\overline{L_2} = \{(a + b)^*\} - \{a^{100} b^{100}\}$$

เนื่องจาก $\overline{L_2}$ ประกอบด้วยสตริงที่เกิดจากการทำซ้ำของ a และ b ยกเว้นสตริงเดียวคือ $a^{100} b^{100}$ เมื่อเรานำ $\overline{L_2}$ มาอินเตอร์เชกชันกับภาษา L_1 ผลที่ได้คือภาษา L

$$\begin{aligned} L_1 \cap \overline{L_2} &= \{a^n b^n\} \cap \{(a + b)^*\} - \{a^{100} b^{100}\} \\ &= \{a^n b^n : n \neq 100, n \geq 0\} \\ &= L \end{aligned}$$

เนื่องจากภาษาค่อนเทิกซ์ฟรีมีคุณสมบัติปิดภายใต้การอินเตอร์เชกชันกับภาษาเรกูลาร์ ดังนั้นภาษา L จึงเป็นภาษาค่อนเทิกซ์ฟรี

ตัวอย่างที่ 7.9 จงพิสูจน์ว่าภาษา L ดังต่อไปนี้เป็นภาษาค่อนเทิกซ์ฟรี

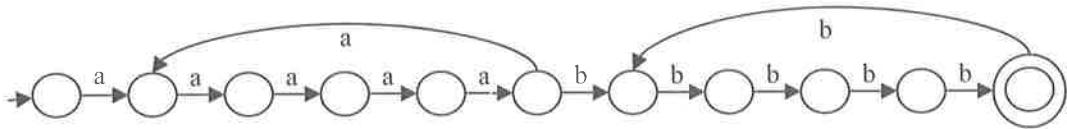
$$L = \{a^n b^n : n > 0, n \bmod 5 \neq 0\}$$

พิสูจน์ : ตัวอย่างนี้สามารถพิสูจน์ได้โดยการสร้างพืชดาวน์อโตมาตราหรือสร้างไวยากรณ์แบบค่อนเทิกซ์ฟรีที่รองรับภาษา L แต่วิธีพิสูจน์ที่ง่ายกว่านั้นคือการใช้คุณสมบัติปิดของการอินเตอร์เชกชันกับภาษาเรกูลาร์มาช่วย กำหนดให้

$$L_1 = \{a^n b^n : n > 0\}$$

$$\text{และ } L_2 = \{a^{5m} b^{5n} : m \text{ และ } n \text{ มีค่าตั้งแต่ } 1, 2, 3, \dots\}$$

เราทราบจากบทที่ผ่านมาว่า L_1 เป็นภาษาค่อนเทิกซ์ฟรี เนื่องจากมีพืชดาวน์อโตมาตราและไวยากรณ์แบบค่อนเทิกซ์ฟรีรองรับ ส่วน L_2 เป็นภาษาเรกูลาร์ เนื่องจากสามารถเขียนไฟในต่ออโตมาตราองรับได้ดังนี้

รูปที่ 7.2 ไฟน์ต่อโถมตามาสำหรับภาษา L_2

เนื่องจากภาษาเรกุลาร์มีคุณสมบัติปิดภายใต้การคอมพลีเมนต์ ดังนั้น $\overline{L_2}$ จะต้องเป็นภาษาเรกุลาร์ เราจะได้ว่า

$$\begin{aligned} L_1 \cap \overline{L_2} &= \{a^n b^n : n > 0\} \cap \overline{L_2} \\ &= \{a^n b^n : n > 0, n \bmod 5 \neq 0\} \\ &= L \end{aligned}$$

ดังนั้นภาษา L จึงเป็นภาษาคอนเทกซ์ฟรี

ตัวอย่างที่ 7.10 จงพิสูจน์ว่าภาษา L ดังต่อไปนี้เป็นภาษาคอนเทกซ์ฟรี

$$L = \{w_1 w_2 : |w_1| = |w_2|, w_1 \in \{a, b\}^*, w_2 \in \{c, d\}^*\}$$

พิสูจน์ : กำหนดให้ภาษา

$$L_1 = \{(a + c)^*\}$$

สตริงของภาษา L ประกอบด้วยสตริงอยู่ 2 ส่วนที่มีความยาวเท่ากัน สตริงย่อยส่วนแรกประกอบด้วยการทำซ้ำอักษร a และ b ส่วนสตริงย่อยส่วนหลังประกอบด้วยการทำซ้ำอักษร c และ d หากต้องการแสดงตัวอย่างของสตริงซึ่งเป็นสมาชิกภายในภาษาสามารถแสดงได้ดังนี้

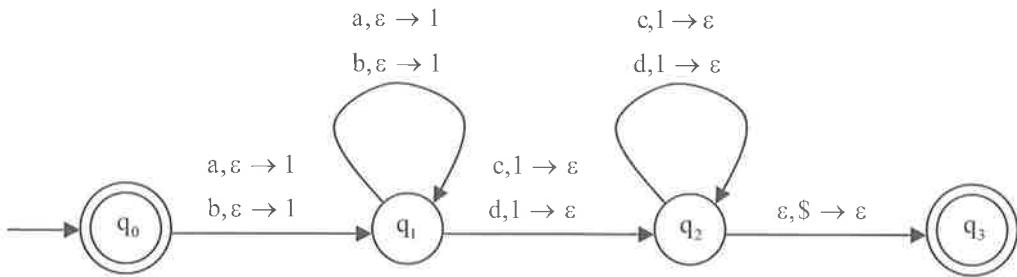
$$\begin{aligned} L = \{&\epsilon, ac, ad, bc, bd, aacc, aaed, aadc, aadd, abcc, abcd, abdc, abdd, \\ &bacc, bacd, badc, badd, bbcc, bbcd, bbdc, bbdd, \dots\} \end{aligned}$$

จะเห็นได้ว่าสตริงภายในภาษา L มีความซับซ้อนค่อนข้างมาก แต่ถ้านำภาษา L มาอินเตอร์เซกชันกับภาษา L_1 ซึ่งบังคับว่าสตริงจะต้องประกอบด้วยตัวอักษร a และ/หรือ c เท่านั้น (อาจเป็นสตริงว่างได้) ผลที่ได้คือภาษาที่เราคุ้นเคยกันอยู่แล้ว นั่นคือ $a^n c^n$ ซึ่งเป็นภาษาคอนเทกซ์ฟรี

$$L \cap L_1 = \{a^n c^n : n \geq 0\}$$

ดังนั้นจึงสรุปได้ว่าภาษา L เป็นภาษาคอนเทกซ์ฟรี เนื่องจากเมื่อนำมาอินเตอร์เซกชันกับภาษา L_1 ซึ่งเป็นภาษาเรกุลาร์ ผลลัพธ์ที่ได้คือภาษา $a^n c^n$ ซึ่งเป็นภาษาคอนเทกซ์ฟรี

อีก การพิสูจน์ความเป็นคอนเทกซ์ฟรีสำหรับภาษาในตัวอย่างนี้อาจทำได้อีกวิธีหนึ่งโดยการสร้าง DPDA สำหรับภาษา L โดยเมื่อ DPDA พบรดับอักษร a หรือ b จะใส่ 1 ลงในบัน叠ตึก จนอินพุตที่อ่านเข้ามายังเป็นตัวอักษร c หรือ d ก็จะตรวจสอบลัญลักษณ์ส่วนบนของสแตกกว่าเป็นเลข 1 หรือไม่ หากเป็นเลข 1 ก็จะนำ 1 ออกจากสแตกทำซ้ำแบบนี้ไปเรื่อยๆ จนกระทั่งอินพุตหมดและสแตกเหลือเพียงลัญลักษณ์เริ่มต้น ที่ให้เปลี่ยนสถานะเข้าสู่สถานะสุดท้ายเพื่อยอมรับอินพุตสตริง ดังแสดงในรูปที่ 7.3

รูปที่ 7.3 DPDA สำหรับภาษา L

7.3 คำาณพื้นฐานเกี่ยวกับภาษาตอนเทกซ์ฟรี

หลังจากที่ศึกษาคุณสมบัติของภาษาตอนเทกซ์ฟรีมาแล้ว ต่อไปจะศึกษาคำาณพื้นฐานที่เกี่ยวข้องกับภาษาตอนเทกซ์ฟรี ซึ่งเนื้อหาในหัวข้อนี้มีความคล้ายคลึงกันหัวข้อที่ 4.2 คำาณพื้นฐานเกี่ยวกับภาษาเรกูลาร์ ในบทที่ 4 แต่ในหัวข้อนี้จะเน้นคำาณพื้นฐานของภาษาตอนเทกซ์ฟรี เพื่อเป็นพื้นฐานในการพิสูจน์ปั๊มปิงлемมาสำหรับภาษาตอนเทกซ์ฟรีต่อไป

คำาณที่ 1 กำหนดให้ L เป็นภาษาตอนเทกซ์ฟรี จะทราบได้อย่างไรว่าสตริง w เป็นสมาชิกอยู่ภายในภาษา L ?

คำาณ : จะรู้ได้ว่าสตริง w เป็นสมาชิกอยู่ภายในภาษาตอนเทกซ์ฟรี L ด้วยการทำวีวิภาค (Parsing) โดยการค้นหาแบบເອົກສໍາສົມບັດ (Exhaustive Search) เพื่อค้นหาการทำวีวิภาคในทุก ๆ กรณีที่เป็นไปได้ อย่างไรก็ตาม ดังที่ได้กล่าวไว้ในบทที่ 5 ว่าวິທີກົດນິ້ນມີເໜາະສົມກັບການປະຢຸກຕີໃຈງານຈົງ ເນື່ອຈາກໃຊ້ເວລານານໃນການທຳງານອີກວິທີໜຶ່ງທີ່ເຄຍນຳເສັນອີປແລ້ວໃນบทที่ 5 ເຫັນກັນຄື່ອງ การค้นหาໂດຍໃຊ້ອັກອວິທີ່ມ CYK ວິທີກົດນິ້ນມີເໜາະສົມກັບການທຳວິວິວິກາມມາກວ່າວິທີກົດນິ້ນມີເໜາະສົມກັບການເອົກສໍາສົມບັດ ເນື່ອຈາກໃຊ້ເວລາໃນການທຳວິວິວິການນ້ອຍກວ່າ

คำาณที่ 2 กำหนดให้ G เป็นໄວຍາกรົນຄອນเทกซ์ฟรี จะทราบได้อย่างไรว่าภาษาของໄວຍາกรົນ G เป็นเซตของภาษาที่ว่างเปล่า นໍ້າດີ $L(G) = \emptyset$?

คำาณ : สามารถตรวจสอบว่าภาษาของໄວຍາกรົນເປັນภาษาที่ว่างเปล่าได้โดยการໃຊ້ອັກອວິທີ່ມดังนີ້

1. กำจัดตัวแปรที่ไม่มีประโยชน์ (Useless Variable)
2. ตรวจสอบดูว่าตัวแปรเริ่มต้น (Start Variable) เป็นตัวแปรที่มีประโยชน์หรือไม่
3. ถ้าสามารถทำวีວິວິກາດໂດຍເຮັດຈຳກັດຕັ້ງຕົ້ນ ແລ້ວໄດ້ພຼອອອກมาເປັນສตรິງຂອງສັນລັກຜົນເທິງມີນອດແສດງວ່າภาษาນີ້ມີວ່າງเปล่า ໃນທາງຕຽບຕັ້ງ ถ้าໄມ່ສາມາດສ້າງສตรິງຂອງສັນລັກຜົນເທິງມີນອດຈາກສັນລັກຜົນ ເຮັດຈຳກັດຕັ້ງໄດ້ ແສດງວ່າภาษาຂອງໄວຍາกรົນ G ວ່າງเปล่า

คำถามที่ 3 กำหนดให้ G เป็นไวยากรณ์ค่อนเทิกซ์ฟรี จะทราบได้อย่างไรว่าภาษาของไวยากรณ์ G เป็นเซตของภาษาที่ไม่จำกัด ?

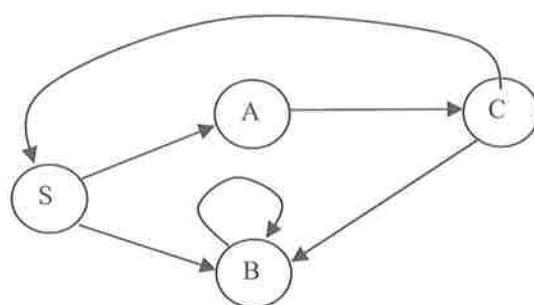
คำตอบ : สามารถตรวจสอบว่าภาษาของไวยากรณ์เป็นเซตของภาษาที่ไม่จำกัดได้โดยการใช้อัลกอริทึมดังต่อไปนี้

1. กำจัดตัวแปรที่ไม่มีประโยชน์
2. กำจัดโปรดักชัน ϵ และโปรดักชันที่สร้างสัญลักษณ์นอนเทอร์มินอลเพียง 1 ตัว (Unit Production)
3. สร้างกราฟแสดงความเกี่ยวข้อง (Dependency Graph) สำหรับตัวแปรทั้งหมดที่เหลืออยู่ในไวยากรณ์ เพื่อแสดงความเกี่ยวข้องและการขึ้นต่อกันของตัวแปร
4. ถ้ามีลูปปรากฏอยู่ภายในกราฟ แสดงว่าภาษาที่นี้เป็นเซตของภาษาที่ไม่จำกัด

ด้วยเช่น เราต้องการตรวจสอบว่าภาษาของไวยากรณ์ G เป็นเซตที่ไม่จำกัด

$$\begin{aligned} S &\rightarrow AB \\ G : A &\rightarrow aCb \mid a \\ B &\rightarrow bB \mid bb \\ C &\rightarrow cBS \end{aligned}$$

หลังจากการกำจัดโปรดักชัน ϵ และโปรดักชันที่สร้างสัญลักษณ์นอนเทอร์มินอลเพียง 1 ตัว สามารถสร้างกราฟแสดงความเกี่ยวข้องได้เป็น



รูปที่ 7.4 กราฟแสดงการเกิดลูป

จากราฟนี้ เราจะเห็นได้ว่ามีลูปเกิดขึ้นภายในกราฟ ดังนั้นจึงสรุปได้ว่าภาษาของไวยากรณ์ G เป็นเซตของภาษาที่ไม่จำกัด

7.4 ปั๊มปิงเลมมาสำหรับภาษาค่อนเทิกซ์ฟรี

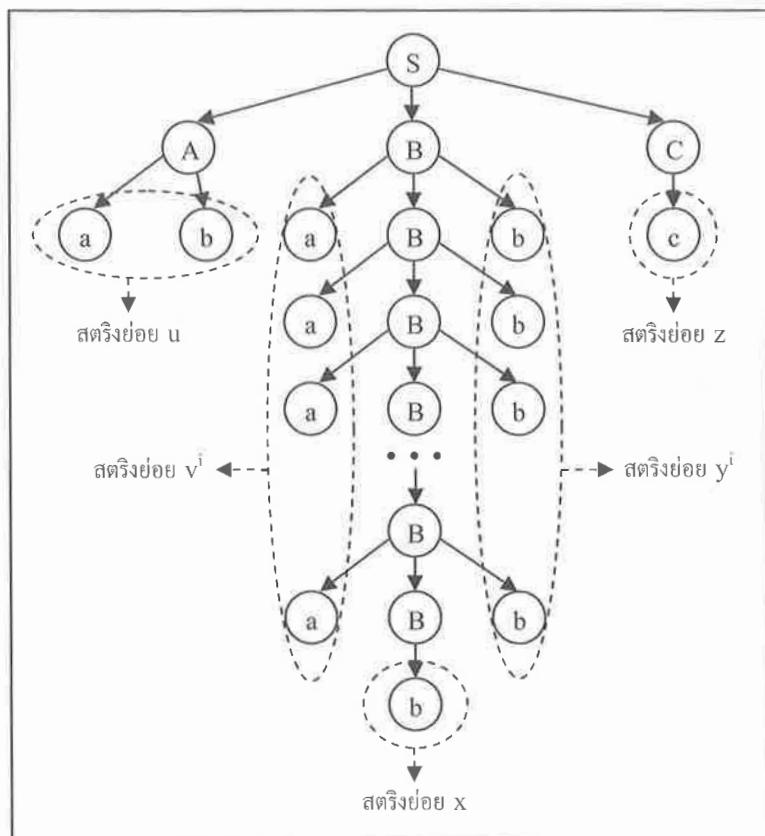
จากที่ได้นำเสนอไปในบทที่ 4 เกี่ยวกับการพิสูจน์ปั๊มปิงเลมมาสำหรับภาษาที่ไม่เป็นเรกูลาร์ ในหัวข้อนี้ จะศึกษาวิธีการพิสูจน์ความไม่เป็นค่อนเทิกซ์ฟรีของภาษาโดยใช้ปั๊มปิงเลมมาสำหรับภาษาค่อนเทิกซ์ฟรี

เราทราบมาแล้วว่า หากภาษาใดมีจำนวนสมาชิกจำกัด ภาษานั้นถือว่าเป็นภาษาเรกูลาร์โดยไม่ต้องลงสัญเนื่องจากสามารถสร้าง DFA หรือ NFA ที่มีจำนวนสถานะจำกัดแทนสมาชิกแต่ละตัวภายในเซตของภาษานั้นได้เสมอ แต่ออย่างไรก็ตาม ถ้าภาษานี้มีจำนวนสมาชิกไม่จำกัด ภาษาดังกล่าวอาจจะเป็นภาษาเรกูลาร์หรือไม่ก็ได้ หลักการนี้สามารถนำมาประยุกต์ใช้ได้กับภาษาของกอนเทเกอร์ฟรี เมื่อจากภาษาที่มีจำนวนสมาชิกไม่จำกัดเท่านั้นจึงจะมีโอกาสเป็นภาษาที่ไม่ใช่กอนเทเกอร์ฟรี สำหรับภาษาเรกูลาร์ ส่วนของสตริงที่ถูกทำซ้ำจะมีอยู่เพียงส่วนเดียวเท่านั้น นั่นคือทำซ้ำสตริงย่อย y ใน xy^iz สำหรับ $i = 0, 1, 2, \dots$ สำหรับภาษาของกอนเทเกอร์ฟรี ส่วนของสตริงที่ถูกทำซ้ำสามารถมีได้ 2 ส่วน เช่น ภาษา L เกิดจากไวยากรณ์กอนเทเกอร์ฟรี G ดังนี้

$$S \rightarrow ABC$$

$$G : \begin{array}{l} A \rightarrow ab \\ B \rightarrow aBb \mid b \\ C \rightarrow c \end{array}$$

เมื่อทดลองสร้างสตริงจากไวยากรณ์ G สตริงที่ได้จะประกอบด้วยสตริงย่อย 5 ส่วน คือ uv^ixy^iz เมื่อ $i = 0, 1, 2, 3, \dots$



รูปที่ 7.5 การทำซ้ำสตริงย่อยภายใน uv^ixy^iz

โดยที่ $u = ab$

$$v^i = a^i \quad ; i = 0, 1, 2, 3, \dots$$

$$x = b$$

$$y^i = b^i \quad ; i = 0, 1, 2, 3, \dots$$

$$z = c$$

โปรดสังเกตว่าสตริงย่อยที่ถูกทำซ้ำนั้นมี 2 สตริง คือ v และ y โดยจำนวนครั้งที่ทำซ้ำสตริงย่อยทั้งสองจะต้องมีจำนวนเท่ากัน ดังนั้นสตริงที่ได้จากไวยากรณ์ G จะต้องมีรูปแบบดังนี้

$$uv^i xy^i z : i = 0, 1, 2, 3, \dots$$

สิ่งที่สังเกตได้ทั้งหมดจากตัวอย่างการทำซ้ำสตริงของไวยากรณ์ตอนที่ก๊อชฟรี G นี้ จะเป็นพื้นฐานเพื่อความเข้าใจหลักการทำงานของปั๊มปิงเลมมาสำหรับภาษาตอนที่ก๊อชฟรี ซึ่งจะอธิบายในทฤษฎีต่อไป

ทฤษฎี 7.4.1 : กำหนดให้ L เป็นภาษาตอนที่ก๊อชฟรีที่สามารถสร้างพุทธดาวน์อ่อมตามาที่มีจำนวนสถานะเท่ากับ m สำหรับทุก ๆ สตริง $s \in L$ โดยที่ $|s| \geq m$ สามารถแบ่งสตริง s ออกเป็นสตริงย่อย ๆ ได้ดังนี้

$$s = uvxyz \quad \text{โดยที่ } |vxy| \leq m \text{ และ } |vy| = k, k \geq 1$$

หากทำซ้ำสตริงย่อย v และ y พร้อม ๆ กันดังแต่ 0 ครั้งขึ้นไป ผลที่ได้จะเป็นสตริงซึ่งยังคงเป็นสมาชิกอยู่ภายใต้ภาษา L

$$uv^i xy^i z \in L \quad ; i = 0, 1, 2, 3, \dots$$

ปั๊มปิงเลมมาสำหรับภาษาตอนที่ก๊อชฟรีมีความคล้ายคลึงกับปั๊มปิงเลมมาของภาษาเรกูลาร์ แต่แตกต่างกันที่ความซับซ้อนของการแบ่งสตริงออกเป็น 5 ส่วน $uvxyz$ แล้วทำซ้ำสตริงย่อยส่วน v และ y พร้อม ๆ กัน ซึ่งไม่ว่าจะแบ่งสตริงเป็นอย่างไร ผลของการทำซ้ำสตริงย่อยจะต้องได้เป็นสตริงที่เป็นสมาชิกของภาษาที่เป็นตอนที่ก๊อชฟรีนั้นเสมอ

ขั้นตอนการพิสูจน์ความไม่เป็นตอนที่ก๊อชฟรีโดยใช้ปั๊มปิงเลมมา จะคล้ายกับการพิสูจน์ความไม่เป็นเรกูลาร์ โดยเป็นการเล่นเกมระหว่าง 2 ฝ่าย ฝ่ายเรายืนยันว่าภาษา L ที่กำลังพิจารณาไม่เป็นภาษาตอนที่ก๊อชฟรี แต่ฝ่ายตรงข้ามกลับโต้แย้งว่าภาษาดังกล่าวนั้นเป็นภาษาตอนที่ก๊อชฟรี ดังนั้นจึงต้องมีการพิสูจน์แบบขัดแย้ง (Contradiction Proof) เริ่มต้นจะต้องสมมุติว่าภาษานั้นเป็นภาษาตอนที่ก๊อชฟรี (สมมุติเพื่อให้เกิดการขัดแย้งภายหลัง) จากนั้นเลือกสตริง s ซึ่ง $s \in L$ และ $|s| \geq m$ แล้วให้ฝ่ายตรงข้ามแบ่งสตริง s ออกเป็น 5 ส่วน คือ $uvxyz$ เมื่อฝ่ายตรงข้ามแบ่งสตริงเส็งแล้ว ให้นำสตริงมาทำซ้ำส่วน v และ y ในจำนวนครั้งที่เท่ากัน หากผลจากการทำซ้ำทำให้สตริง $uv^i xy^i z \notin L$ และแสดงว่าสมมุติฐานที่ตั้งไว้ว่าภาษา L เป็นภาษาตอนที่ก๊อชฟรีนั้นผิด เพราะหาก L เป็นภาษาตอนที่ก๊อชฟรีจริง สตริงที่ถูกทำซ้ำจะต้องเป็นสมาชิกอยู่ภายใต้ภาษา L ดังนั้นจึงสรุปได้ว่าภาษา L ไม่ใช่ภาษาตอนที่ก๊อชฟรี

ข้อควรระวัง ใน การใช้ปั๊มปิงเลมมาสำหรับภาษาตอนที่ก๊อชฟรีคือ อย่าใช้ปั๊มปิงเลมมากับภาษาที่เป็นภาษาตอนที่ก๊อชฟรี เนื่องจากผลที่ได้จะไม่เป็นไปตามทฤษฎีของปั๊มปิงเลมมา ดังนั้นก่อนจะใช้ปั๊มปิงเลมมาสำหรับภาษาตอนที่ก๊อชฟรี เราจะต้องมั่นใจก่อนว่าภาษานั้นไม่ใช่ภาษาตอนที่ก๊อชฟรี ปั๊มปิงเลมมาเป็นเพียงเหตุผลที่ใช้สนับสนุนการตัดสินใจว่าภาษาที่พิจารณาตนไม่ใช่ภาษาตอนที่ก๊อชฟรี

ตัวอย่างที่ 7.11 กำหนดให้ภาษา L ลูกนิยามดังนี้

$$L = \{a^n b^n c^n : n \geq 0\}$$

งพิสูจน์ว่าภาษา L ไม่ใช่ภาษาค่อนเท็กซ์พรี

พิสูจน์ : โดยใช้ปั๊มปิงเลนมาสำหรับภาษาค่อนเท็กซ์พรี

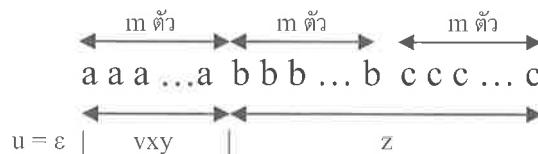
ขั้นตอนที่ 1 : สมมุติให้ L เป็นภาษาค่อนเท็กซ์พรีด้วยค่าคงที่ m

ขั้นตอนที่ 2 : เลือกสตริง $s \in L$ โดยที่ $|s| \geq m$

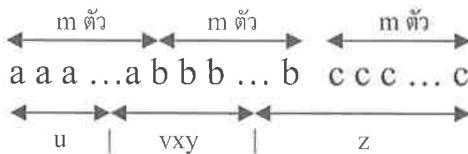
$$s = a^m b^m c^m$$

ขั้นตอนที่ 3 : ฝ่ายตรงข้ามแบ่งสตริง s ออกเป็น 5 ส่วน $uvxyz$, $|vxy| \leq m$ และ $|vy| = k$, $k \geq 1$ โดยการแบ่งสตริงจะต้องพิจารณาทุกกรณีที่สตริง vxy จะเป็นไปได้

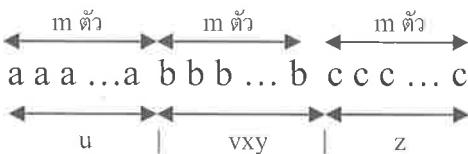
(3.1)



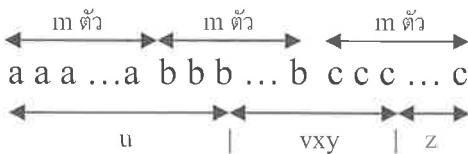
(3.2)



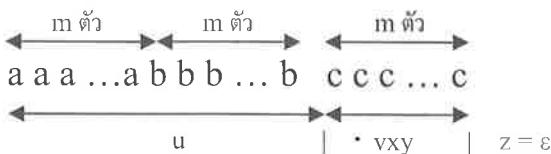
(3.3)



(3.4)



(3.5)



ขั้นตอนที่ 4 : เลือก i เพื่อทำซ้ำ uv^ixy^iz ให้ออกจากภาษา L

- สำหรับกรณีที่ (3.1), (3.3) และ (3.5) เลือก $i = 0$ ก็สามารถทำให้ uv^0xy^0z มีจำนวนตัวอักษร a, b และ c ไม่เท่ากัน เช่น กรณี (3.1) uv^0xy^0z จะมีจำนวนตัวอักษร a น้อยกว่าจำนวนตัวอักษร b และ c
- สำหรับกรณีที่ (3.2) และ (3.4) เลือก $i = 0$ ก็สามารถทำให้จำนวนตัวอักษร c หรือ a มากกว่า จำนวนตัวอักษรตัวอื่น ๆ เช่น กรณี (3.2) uv^0xy^0z จะมีจำนวนตัวอักษร c มากกว่าจำนวนตัวอักษร a และ b

ดังนั้นไม่ว่าจะเป็นกรณีใด ๆ ถ้า $i = 0$, $uv^0xy^0z \notin L$ ดังนั้นจึงสรุปได้ว่า

$$L = \{a^n b^n c^n : n \geq 0\} \text{ ไม่ใช่ภาษาค่อนเทิกซ์ฟรี}$$

ตัวอย่างที่ 7.12 งพิสูจน์ว่าภาษา

$$L = \{w \in \{a, b, c\}^*: n_a(w) > n_b(w), n_a(w) > n_c(w)\} \text{ ไม่ใช่ภาษาค่อนเทิกซ์ฟรี}$$

โดยที่ $n_x(w)$ หมายถึง จำนวนตัวอักษร x ภายในสตริง w

พิสูจน์ : โดยใช้ปั๊มปิงเลนมาสำหรับภาษาค่อนเทิกซ์ฟรี

ขั้นตอนที่ 1 : สมมุติให้ L เป็นภาษาค่อนเทิกซ์ฟรีด้วยค่าคงที่ m

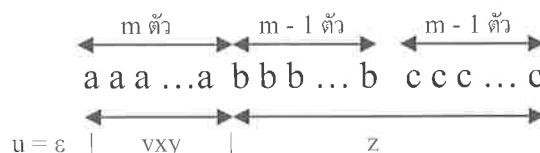
ขั้นตอนที่ 2 : เลือกสตริง $s \in L$ โดยที่ $|s| \geq m$

$$s = a^m b^{m-1} c^{m-1}$$

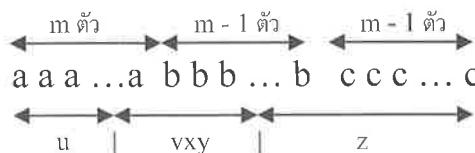
ขั้นตอนที่ 3 : ฝ่ายตรงข้ามแบ่งสตริง s ออกเป็น 5 ส่วน $uvxyz$ โดยที่ $|vxy| \leq m$ และ $|vy| = k$, $k \geq 1$

ดังนี้

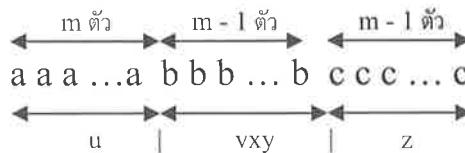
(3.1)



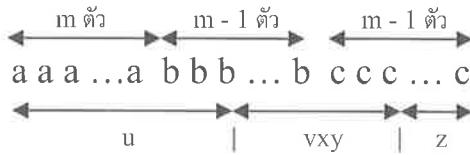
(3.2)



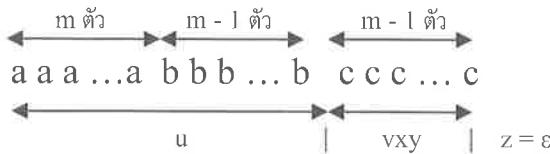
(3.3)



(3.4)



(3.5)



ขั้นตอนที่ 4 : เลือก i เพื่อ Pump uv^ixy^iz ออกจาก L

- สำหรับกรณีที่ (3.1) เลือก $i = 0$ ก็สามารถทำให้ uv^0xy^0z มีจำนวนตัวอักษร a น้อยกว่าหรือเท่ากับ จำนวนตัวอักษร b และ c
- สำหรับกรณีที่ (3.2) เราเลือก $i = 0$ ก็สามารถทำให้ uv^0xy^0z มีจำนวนตัวอักษร a น้อยกว่าหรือ เท่ากับจำนวนตัวอักษร c
- สำหรับกรณีที่ (3.3) เราเลือก $i = 2$ ก็สามารถทำให้ uv^2xy^2z มีจำนวนตัวอักษร b มากกว่าหรือ เท่ากับจำนวนตัวอักษร a
- สำหรับกรณีที่ (3.4) เราเลือก $i = 2$ ก็สามารถทำให้ uv^2xy^2z มีจำนวนตัวอักษร b และ c มากกว่า หรือเท่ากับจำนวนตัวอักษร a
- สำหรับกรณีที่ (3.5) เราเลือก $i = 2$ ก็สามารถทำให้ uv^2xy^2z มีจำนวนตัวอักษร c มากกว่าหรือ เท่ากับจำนวนตัวอักษร a

ไม่ว่ากรณีใด ๆ ของ $uvxyz$ สามารถทำให้ $uv^ixy^iz \notin L$ ดังนี้จะส្មปได้ว่า

$$L = \{w \in \{a, b, c\}^* : n_a(w) > n_b(w), n_a(w) > n_c(w)\}$$
 ไม่ใช่ภาษาค่อนเทิกซ์ฟรี

ตัวอย่างที่ 7.13 จงพิสูจน์ว่าภาษา

$$L = \{w \in \{a, b\}^* : n_b(w) = n_a(w)^2\}$$
 ไม่ใช่ภาษาค่อนเทิกซ์ฟรี

พิสูจน์ : โดยใช้ปิงเลมมาสำหรับภาษาค่อนเทิกซ์ฟรี

ขั้นตอนที่ 1 : สมมุติให้ L เป็นภาษาค่อนเทิกซ์ฟรีด้วยค่าคงที่ m

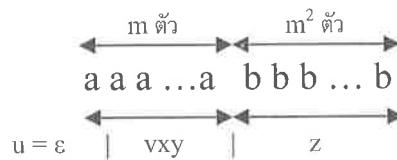
ขั้นตอนที่ 2 : เลือกสตริง $s \in L$ โดยที่ $|s| \geq m$

$$s = a^m b^{m^2}$$

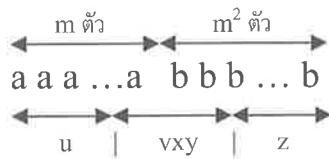
ขั้นตอนที่ 3 : ฝ่ายตรงข้ามแบ่งสตริง s ออกเป็น 5 ส่วน $uvxyz$ โดยที่ $|vxy| \leq m$ และ $|vy| = k$, $k \geq 1$

ดังนี้

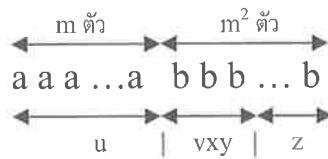
(3.1)



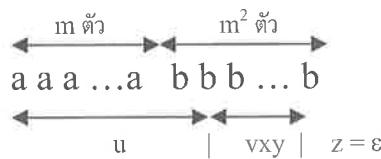
(3.2)



(3.3)



(3.4)



โปรดสังเกตว่ากรณีที่ (3.3) และ (3.4) ถือว่าเป็นกรณีเดียวกันเนื่องจาก vxy ประกอบด้วย b เท่านั้นกับขั้นตอนที่ 4 : เลือก i เพื่อทำซ้ำ uv^ixy^iz ให้ออกจากภาษา L

- สำหรับกรณีที่ (3.1) เลือก $i = 0$ ก็สามารถทำให้ uv^0xy^0z มีจำนวนตัวอักษร a ลดลงไป k ตัว ซึ่งจะทำให้ $(m - k)^2 \neq m^2$
 - สำหรับกรณีที่ (3.2) สามารถพิจารณาได้อีก 3 กรณีย่อยคือ
 - กรณีที่ v ประกอบด้วยทั้ง a และ b ส่วน y ประกอบด้วย b อย่างเดียว
 - กรณีที่ v ประกอบด้วย a อย่างเดียว ส่วน y ประกอบด้วยทั้ง a และ b
 - กรณีที่ v ประกอบด้วย a อย่างเดียว ส่วน y ประกอบด้วย b อย่างเดียว
- ใน 2 กรณีย่อยแรก สามารถเลือก $i = 2$ เพื่อทำให้เกิดการสลับตำแหน่งของ a และ b ซึ่งจะทำให้สตริงที่ได้จากการทำซ้ำไม่เป็นสมาชิกของภาษา L

สำหรับในกรณีย่อยสุดท้าย สมมุติให้ $|v| = k_1$ และ $|y| = k_2$ เราเลือก $i = 0$ ก็สามารถทำให้ uv^0xy^0z มีจำนวนตัวอักษร a ลดลงไป k_1 ตัว และจำนวนตัวอักษร b ลดลงไป k_2 ตัว ดังนั้นจำนวนตัวอักษร a และ b หลังจากการทำซ้ำด้วย $i = 0$ จะมีค่าเป็น

$$\text{จำนวนตัวอักษร } a = m - k_1 \text{ ตัว}$$

$$\text{จำนวนตัวอักษร } b = m^2 - k_2 \text{ ตัว}$$

หากต้องการตรวจสอบว่า สตริงที่ได้จากการทำซ้ำเป็นสามชิกของภาษา L หรือไม่ สามารถทำได้โดยนำจำนวนตัวอักษร a ที่ได้จากการทำซ้ำ many กำลังสอง หากเท่ากับจำนวนตัวอักษร b หลังจากการทำซ้ำก็แสดงว่า สตริงนั้นบังคงอยู่ในภาษา L นั่นคือ $(m - k_1)^2 = (m^2 - k_2)$

แต่เนื่องจากข้อเท็จจริงที่ว่า

$$\text{จำนวนตัวอักษร } a \text{ ยกกำลัง } 2$$

$$(m - k_1)^2$$



$$m^2 - 2k_1m + k_1^2$$

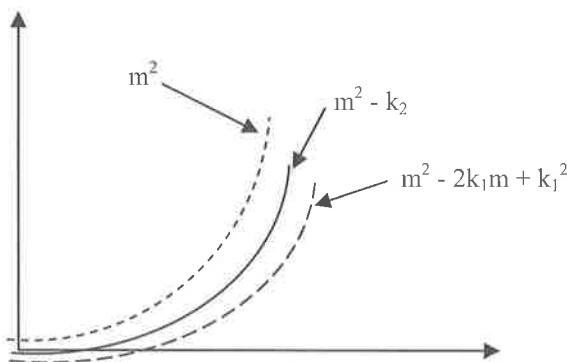
$$\text{จำนวนตัวอักษร } b$$

$$(m^2 - k_2)$$



$$m^2 - k_2$$

ถ้านำนิพจน์ทั้งสองไปเขียนเป็นกราฟ จะได้กราฟดังรูป



รูปที่ 7.6 กราฟแสดงความสัมพันธ์ของ $m^2 - k_2$ และ $m^2 - 2k_1m + k_1^2$

จากราฟ จะเห็นได้อย่างชัดเจนว่า $m^2 - k_2$ มีค่ามากกว่า $m^2 - 2k_1m + k_1^2$ เสมอ เมื่อจาก k_2 เป็นค่าคงที่ไม่เกิน m ส่วน $2k_1m$ เป็นกราฟเส้นตรง ซึ่งทำให้ $(m - k_1)^2 \neq (m^2 - k_2)$

ดังนั้นจึงทำให้สตริงที่ได้จากการทำซ้ำด้วย $i = 0$ ไม่เป็นสามชิกภาษาในภาษา L

- สำหรับกรณีที่ (3.3) และ (3.4) เลือก $i = 0$ จะทำให้จำนวนตัวอักษร b ลดลง k ตัว ซึ่งทำให้ $m^2 \neq m^2 - k$

ดังนั้นไม่ว่ากรณีใด ๆ ของ $uvxyz$ เราสามารถทำให้ $uv^ixy^jz \notin L$ ได้เสมอ จึงสรุปได้ว่า

$$L = \{w \in \{a, b\}^* : n_b(w) = n_a(w)^2\} \text{ ไม่ใช่ภาษาค่อนเท็กซ์ฟรี}$$

ตัวอย่างที่ 7.14 จงพิสูจน์ว่าภาษา

$$L = \{ww : w \in \{a, b\}^*\} \text{ ไม่ใช่ภาษาค่อนเท็กซ์ฟรี}$$

พิสูจน์ : โดยใช้ปั๊มปิงเลมมาสำหรับภาษาคอนเทกซ์ฟรี

ขั้นตอนที่ 1 : สมมุติให้ L เป็นภาษาคอนเทกซ์ฟรีด้วยค่าคงที่ m

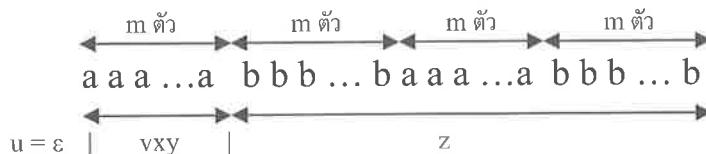
ขั้นตอนที่ 2 : เลือกสตริง $s \in L$ โดยที่ $|s| \geq m$

$$s = a^m b^m a^m b^m$$

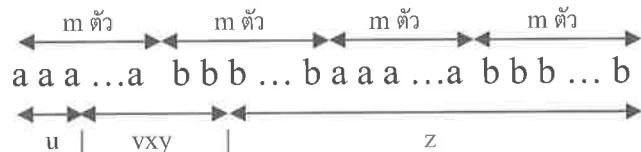
ขั้นตอนที่ 3 : ฝ่ายตรงข้ามแบ่งสตริง s ออกเป็น 5 ส่วน $uvxyz$ โดยที่ $|vxy| \leq m$ และ $|vy| \geq 1$

ดังนี้

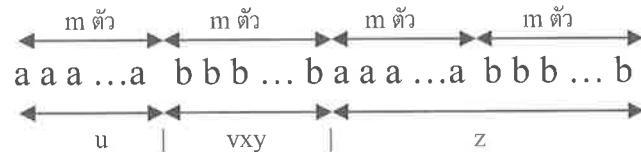
(3.1)



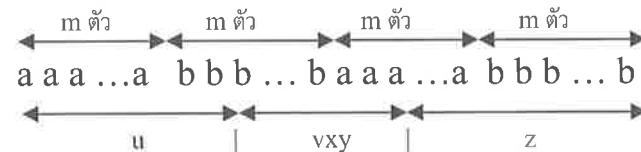
(3.2)



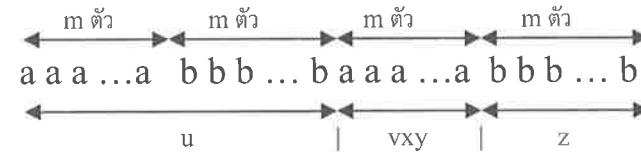
(3.3)



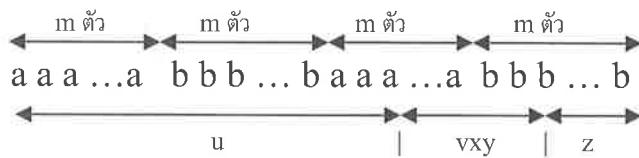
(3.4)



(3.5)



(3.6)



(3.7)



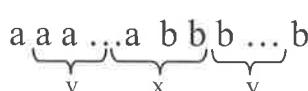
โปรดสังเกตว่า

กรณีที่ (3.1) และ (3.5) ถือว่าคล้ายกันนี้ของจาก vxy ประกอบด้วย a เหมือนกัน

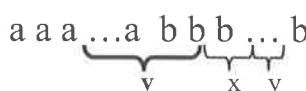
กรณีที่ (3.3) และ (3.7) ถือว่าคล้ายกันนี้ของจาก vxy ประกอบด้วย b เหมือนกัน

กรณีที่ (3.2) และ (3.6) สามารถเกิดกรณีอยู่ได้อีก 2 กรณี คือ

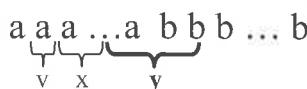
(a) กรณีที่ v ประกอบด้วยตัวอักษร a และ y ประกอบด้วยตัวอักษร b



(b) กรณีที่ v หรือ y ประกอบด้วยตัวอักษรทั้ง a และ b สมมูลกัน

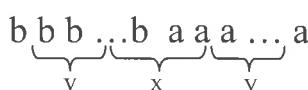


หรือ



กรณีที่ 3.4 สามารถเกิดกรณีอยู่ได้อีก 2 กรณี คือ

(a) กรณีที่ v ประกอบด้วยตัวอักษร b และ y ประกอบด้วยตัวอักษร a



(b) กรณีที่ v หรือ y ประกอบด้วยตัวอักษรทั้ง b และ a ผสมกัน

$b\ b\ b\ \dots\ b\ \underbrace{a\ a\ a}_{v}\ \dots\ a$

หรือ

$b\ b\ b\ \dots\ b\ \underbrace{a\ a\ a\ \dots\ a}_{y}$

ขั้นตอนที่ 4 : เลือก i เพื่อทำซ้ำ $uv^i xy^j z$ ให้ออกจากภาษา L

- สำหรับกรณีที่ (3.1) และ (3.5) เลือก $i = 0$ ก็สามารถทำให้ $uv^0 xy^0 z$ มีจำนวนตัวอักษร a ลดลงไป k ตัว ซึ่งจะทำให้ $(m - k) \neq m$
- สำหรับกรณีที่ (3.3) และ (3.7) เลือก $i = 0$ ก็สามารถทำให้ $uv^0 xy^0 z$ มีจำนวนตัวอักษร b ลดลงไป k ตัว ซึ่งจะทำให้ $(m - k) \neq m$
- สำหรับกรณีที่ (3.2a) และ (3.6a) หากสมมุติให้ $|v| = k_1$ และ $|y| = k_2$ เลือก $i = 0$ ก็สามารถทำให้ $uv^0 xy^0 z$ มีจำนวนตัวอักษร a ลดลงไป k_1 ตัว และจำนวนตัวอักษร b ลดลงไป k_2 ตัว ดังนี้จำนวนตัวอักษร a และ b หลังจากการทำซ้ำ ด้วย $i = 0$ จะหายไป k_1 และ k_2 ตัว ตามลำดับ
- สำหรับกรณีที่ (3.2b) และ (3.6b) เราเลือก $i = 2$ ก็สามารถทำให้ $uv^2 xy^2 z$ นอกจากจะมีตัวอักษร a และ b เกินมาแล้ว ยังทำให้เกิดการสลับที่กันของตัวอักษร a และ b ระหว่างภาษาในของสตริง s ทำให้ $uv^2 xy^2 z \notin L$
- สำหรับกรณีที่ 3.4 มีลักษณะการพิสูจน์ที่คล้ายคลึงกับกรณี 3.2 และ 3.6

ดังนั้นไม่ว่ากรณีใด ๆ ของ $uvxyz$ เราสามารถทำให้ $uv^i xy^j z \notin L$ ได้เสมอ จึงสรุปได้ว่า

$L = \{ww : w \in \{a, b\}^*\}$ ไม่ใช่ภาษาคอนเทกซ์ฟรี

ตัวอย่างที่ 7.15 จงพิสูจน์ว่าภาษา

$L = \{a^{n!} : n \geq 0\}$ ไม่ใช่ภาษาคอนเทกซ์ฟรี

พิสูจน์ : โดยใช้ปั๊มปิ่งлемมาสำหรับภาษาคอนเทกซ์ฟรี

ขั้นตอนที่ 1 : สมมุติให้ L เป็นภาษาคอนเทกซ์ฟรีด้วยค่าคงที่ m

ขั้นตอนที่ 2 : เลือกสตริง $s \in L$ โดยที่ $|s| \geq m$

$$s = a^{n!}$$

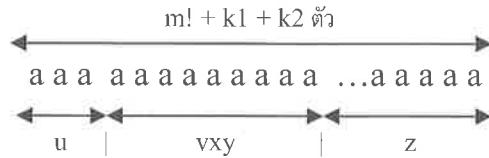
ขั้นตอนที่ 3 : ฝ่ายตรงข้ามแบ่งสตริง s ออกเป็น 5 ส่วน $uvxyz$ โดยที่ $|vxy| \leq m$ และ $|vy| = k$, $k \geq 1$

ซึ่งมีเพียงกรณีเดียวเท่านั้นที่ต้องพิจารณา คือ



กำหนดให้ $|v| = k_1$ และ $|y| = k_2$ โดยที่ $1 \leq k_1 + k_2 \leq m$ และ $k_1 + k_2 = k$

ขั้นตอนที่ 4 : เลือก i เพื่อทำซ้ำ $uv^i xy^i z$ ให้ออกจากภาษา L โดยเลือก $i = 2$ จะทำให้ $uv^2 xy^2 z$ มีลักษณะดังนี้



เนื่องจาก $k_1 + k_2 = k$ ดังนั้น

$$m! + k_1 + k_2 = m! + k$$

การพิสูจน์ว่า $uv^2 xy^2 z \notin L$ ได้นั้น จะต้องนั่นใจได้ว่า

$$a^{m!+k} \neq a^{p!} \text{ สำหรับ } 1 \leq k \leq m \text{ และค่า } p \text{ ใด ๆ}$$

นั่นคือ

$$(m! + k) \neq p! \text{ สำหรับ } 1 \leq k \leq m \text{ และค่า } p \text{ ใด ๆ}$$

สามารถพิสูจน์ความไม่เท่ากันดังกล่าวได้ดังนี้

$$\begin{aligned} (m! + k) &\leq (m! + m) \quad \text{สำหรับ } 1 \leq k \leq m \\ &\leq (m! + m!) \\ &< (m!.m + m!) \\ &< m!(m + 1) \\ &< (m + 1)! \end{aligned}$$

โปรดสังเกตว่า $(m! + k)$ จะมีค่าเท่ากับ $p!$ ได้มีเมื่อค่า p มีค่าเป็น $m + 1$ หรือ $m + 2$ หรือ $m + n$ แต่ผลจากการที่ได้พิสูจน์มา สรุปได้ว่าสำหรับ $1 \leq k \leq m$ ค่าของ $(m! + k)$ จะมีค่าน้อยกว่า $(m + 1)!$ เสมอ ดังนั้นจึงไม่มีทางที่จะหาค่า p ที่ทำให้ $(m! + k) = p!$ เป็นจริงได้เลย ด้วยเหตุผลนี้จึงทำให้ $uv^2 xy^2 z \notin L$ และเป็นผลให้สมมุติฐานที่ตั้งไว้ตอนต้นนั้นเป็นเท็จ ดังนั้นภาษา L จึงไม่ใช่ภาษาคุณเท็จช์ฟรี

แบบฝึกหัด

1. การพิสูจน์ความไม่เป็นภาษาค่อนเท็กซ์พรีสามารถทำได้กี่วิธี อะไรบ้าง

สำหรับข้อ 2 - 5 จะใช้มีปิงเล่มมาสำหรับภาษาค่อนเท็กซ์พรีพิสูจน์ว่าภาษาดังต่อไปนี้ไม่เป็นภาษาค่อนเท็กซ์พรี

$$2. L = \{a^n b^{3n} a^n \mid n \geq 1\}$$

$$3. L = \{w \in \{a, b, c\}^* \mid n_a(w) = \min(n_b(w), n_c(w))\}$$

$$4. L = \{a^i b^j c^k \mid i > j > k\}$$

$$5. L = \{a^n b^m c^{n+m} \mid m, n \geq 1\}$$

สำหรับข้อ 6 - 9 จะเลือกว่าภาษาใดเป็นภาษาค่อนเท็กซ์พรี พร้อมทั้งพิสูจน์คำตอบนั้น

$$6. L = \{a^n b^m a^n b^m \mid m, n \geq 1\}$$

$$7. L = \{wcw \mid w \in \{a\}^*\}$$

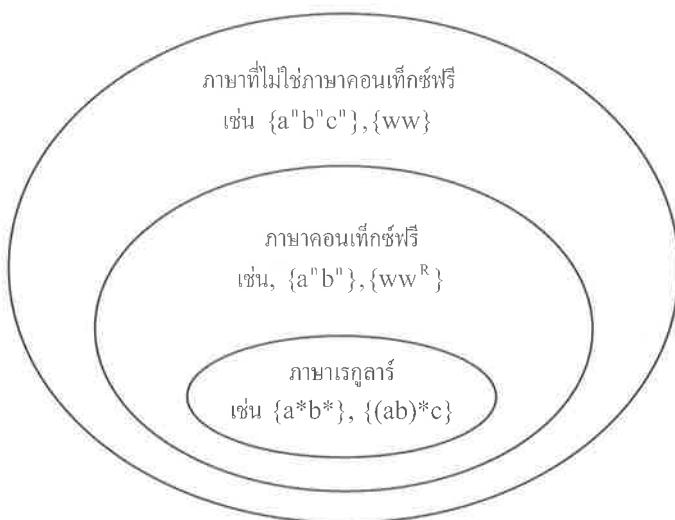
$$8. L = \{w \in \{a, b\}^* \mid n_a(w) = 3 n_b(w)\}$$

$$9. \text{ คอมพลีเมนต์ของภาษา } L \text{ โดยที่ } L = \{a^i b^j c^k \mid i \geq j \text{ or } j \geq k\}$$

10. ปั๊มปิงเล่มมาสำหรับภาษาค่อนเท็กซ์พรีมีความแตกต่างจากปั๊มปิงเล่มมาสำหรับภาษาเรกุляร์อย่างไรบ้าง

ทัวริงแมชีน

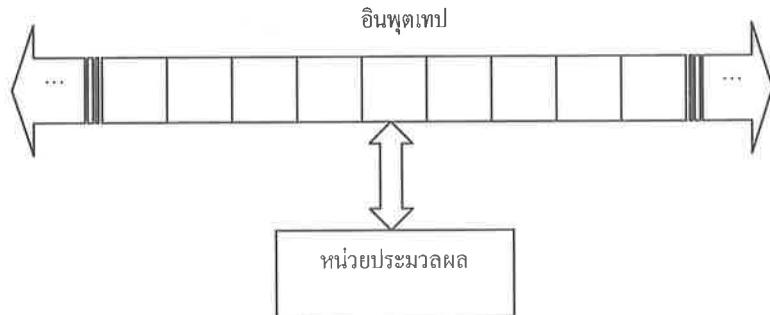
จากบทที่ผ่านมา ไฟโนต์อโตมาตาเป็นเครื่องมือที่ใช้ในการรองรับภาษาเรกูลาร์ และพุชดาวน์อโตมาตา เป็นเครื่องมือที่ใช้รองรับภาษาค่อนเทิกซ์ฟรี แต่ยังมีอีกหลายภาษาที่ทั้งไฟโนต์อโตมาตาและพุชดาวน์อโตมาตา ไม่สามารถรองรับได้ เช่น $\{a^n b^n c^n : n \geq 0\}$ ดังนั้นในบทนี้ เราจะได้ศึกษาถึงจักรกลชนิดใหม่ที่มีขีดความสามารถสูง ในการรองรับภาษาที่ไม่ใช่ภาษาเรกูลาร์และภาษาค่อนเทิกซ์ฟรี นั่นคือทัวริงแมชีน (Turing machine)



รูปที่ 8.1 ความสัมพันธ์ของภาษาทั้งสามประเภท

ทัวริงแมชีนถูกคิดค้นโดยนักวิทยาศาสตร์คอมพิวเตอร์ชื่อ อเลน ทัวริง (Alan M. Turing) และใช้งานกันอย่างแพร่หลายในยุคนี้ จนทัวริงแมชีนได้กลายมาเป็นต้นแบบของการทำงานภายในของคอมพิวเตอร์ในยุคแรก ๆ รวมถึงเป็นรากฐานที่สำคัญของการทำงานของคอมพิวเตอร์ในยุคปัจจุบัน

8.1 ส่วนประกอบของทัวริงแมชีน



รูปที่ 8.2 ส่วนประกอบภายในทัวริงแมชีน

ทัวริงแมชีนมีมาตราฐานประกอบด้วยหน่วยประมวลผล ซึ่งทำหน้าที่ควบคุมการทำงานของทัวริงแมชีน และ อินพุตเทปซึ่งทำหน้าที่เก็บสตอริจที่จะป้อนเข้าสู่ทัวริงแมชีน

หน่วยประมวลผล เป็นหน่วยที่มีสถานะภายในซึ่งสามารถเปลี่ยนแปลงสถานะได้ตามตัวอักษรที่อ่านเข้ามา ทางอินพุตเทป

อินพุตเทป เป็นเทปเก็บอินพุตที่จะป้อนเข้าสู่ทัวริงแมชีน โดยหน่วยประมวลผลสามารถอ่านและเขียน ตัวอักษรลงบนอินพุตเทปได้ผ่านทางหัวอ่าน-เขียน เพื่อให้ง่ายในการอธิบาย ผู้เขียนจะใช้คำว่า “หัวอ่าน” แทนคำว่า “หัวอ่าน-เขียน” หัวอ่านสามารถเคลื่อนที่ไปทางซ้ายหรือขวาทีละ 1 ตำแหน่ง เนื่องจากคุณสมบัติของหัวอ่าน ที่สามารถเคลื่อนที่ย้อนกลับไปยังอินพุตที่เคยอ่านมาแล้วหรือเคลื่อนที่ไปยังอินพุตที่ยังไม่เคยอ่านได้ ทำให้สามารถ การเข้าถึงข้อมูลไม่ถูกจำกัดดังเช่นที่เกิดขึ้นกับสแตก์ที่มีการทำงานแบบเข้าที่หลัง-ออกก่อน (Last-In First-Out) ทำให้ทัวริงแมชีนมีความสามารถเหนือกว่าพุชดาวน์อัลมาต้า

นิยาม 8.1.1 : ทัวริงแมชีนที่คาดเดาได้ (Deterministic Turing machine) ถูกนิยามโดย

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

โดยที่

Q คือ เซตของสถานะภายในหน่วยประมวลผล

Σ คือ เซตของตัวอักษรอินพุต

Γ คือ เซตของตัวอักษรที่ถูกบรรจุลงในอินพุตเทป

δ คือ เซตของทรานซิชันฟังก์ชัน (Transition Function) ซึ่งเป็นความสัมพันธ์ดังนี้

$$Q \times \Gamma \longrightarrow Q \times \Gamma \times \{R, L\}$$

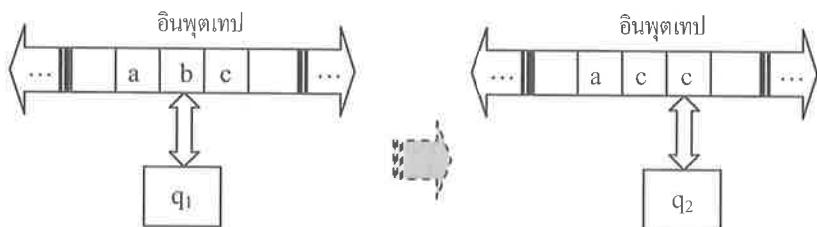
ความสัมพันธ์นี้เป็นการเปลี่ยนแปลงภายในทัวริงแมชีน ซึ่งเกิดจากสถานะภายในหน่วยประมวลผลและ ตัวอักษรปัจจุบันบนอินพุตเทป โดยการเปลี่ยนแปลงที่เกิดขึ้นอาจทำให้สถานะภายในและตัวอักษรบนอินพุตเทป เกิดการเปลี่ยนแปลงหรือไม่ก็ได้ นอกจากนี้หัวอ่านภายในทัวริงแมชีนยังสามารถเคลื่อนที่ไปทางขวาหรือซ้ายทีละ 1 ตำแหน่งได้อีกด้วย (แต่หัวอ่านจะอยู่ที่ตำแหน่งเดิมไม่ได้)

- q₀ คือ สถานะเริ่มต้นของหน่วยประมวลผล โดยที่ q₀ ∈ Q
- คือ ซ่องว่าง โดยอินพุตที่ถูกป้อนเข้าสู่ทัวริงแมชีนจะมีซ่องว่างเพื่อระบุตำแหน่งเริ่มต้นและตำแหน่งสิ้นสุดของอินพุตเสมอ โดยเมื่อเริ่มต้นทำงาน ทัวริงแมชีนจะทิ้งหัวอ่านไปที่ตำแหน่งตัวอักษรซ้ายสุดที่อยู่ติดกับซ่องว่าง
- F คือ เซตของสถานะสุดท้ายโดยที่ F ⊆ Q

ตัวอย่างที่ 8.1 กำหนดให้ทรานซิชันฟังก์ชันของทัวริงแมชีนคือ

$$\delta(q_1, b) = (q_2, c, R)$$

เราสามารถแสดงการเปลี่ยนแปลงที่เกิดขึ้นภายในทัวริงแมชีนเมื่อทรานซิชันฟังก์ชันดังกล่าวถูกเรียกใช้งานได้ดังนี้



รูปที่ 8.3 การเกิดทรานซิชันภายในทัวริงแมชีน

เมื่อสถานะภายใน คือ q₁ และตัวอักษรอินพุตบนเทปคือ b ทัวริงแมชีนจะเกิดการเปลี่ยนแปลงโดยเปลี่ยนสถานะไปเป็น q₂ และเปลี่ยนอินพุต b บนเทปให้เป็น c จากนั้นเคลื่อนหัวอ่านไปทางขวา 1 ตำแหน่ง

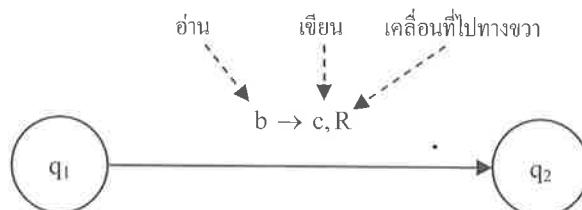
นอกเหนือจากการแสดงการเปลี่ยนแปลงภายในทัวริงแมชีนด้วยรูปภาพแล้ว ยังสามารถแสดงการเปลี่ยนแปลงที่เกิดขึ้นด้วยรูปแบบง่าย ๆ ดังต่อไปนี้

$$aq_1bc \vdash acq_2c$$

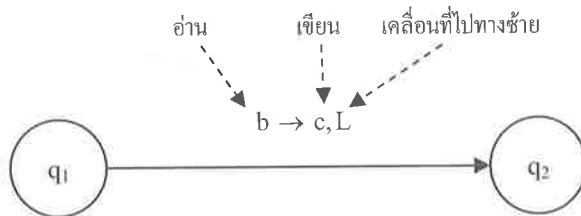
หากสถานะอยู่ข้างหน้าตัวอักษรใดก็แสดงว่าทัวริงแมชีนกำลังอ่านตัวอักษรตัวนั้นอยู่ และจะใช้สัญลักษณ์ \vdash แทนการเกิดทรานซิชัน วิธีการอธิบายเช่นนี้เหมือนกับวิธีการอธิบายที่พูมามาในเรื่องพุชดาวน์อโตมาตา และเรียกการอธิบายในลักษณะนี้ว่า การอธิบาย ณ ช่วงขณะเวลาหนึ่ง (Instantaneous Description)

เพื่อเป็นการเสริมความเข้าใจในการทำงานของทัวริงแมชีน สามารถแทนทัวริงแมชีนและเซตของทรานซิชันทั้งหมดด้วยทรานซิชันกราฟดังต่อไปนี้

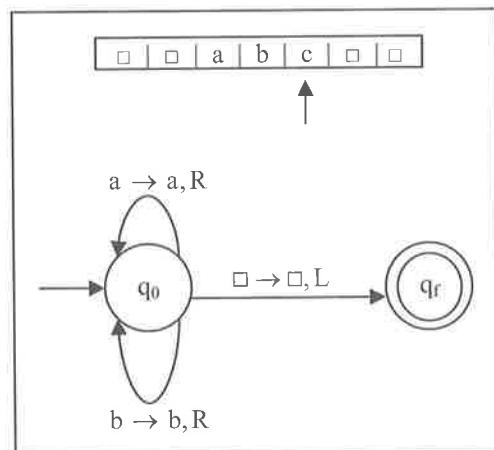
- ทรานซิชัน $\delta(q_1, b) = (q_2, c, R)$ ถูกแทนด้วย



- ทรานซิชัน $\delta(q_1, b) = (q_2, c, L)$ ถูกแทนด้วย



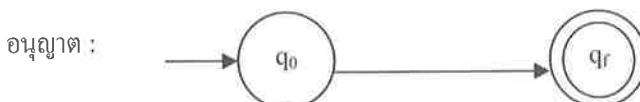
อนั้น ทัวริงแมชีนจะหยุดทำงาน (Halt) เมื่อไม่มีทรานซิชันใด ๆ สามารถเกิดขึ้น ณ ขณะนั้นได้อีก เช่น



รูปที่ 8.4 การหยุดการทำงานของทัวริงแมชีน

จากรูป จะเห็นได้ว่าที่สถานะ q_0 ไม่มีการนิยามทรานซิชันสำหรับอินพุต c ไว้เลย ดังนั้นทัวริงแมชีนจะหยุดทำงานที่สถานะ q_0 และปฏิเสธสตริง abc ทันที การหยุดการทำงานของทัวริงแมชีนแบ่งออกเป็น 2 แบบ คือ แบบปฏิเสธอินพุตสตริง และแบบยอมรับอินพุตสตริง การหยุดการทำงานแบบแรกจะเกิดขึ้นเมื่อทัวริงแมชีนหยุดการทำงานที่สถานะที่ไม่ใช่สถานะสุดท้าย หรือทัวริงแมชีนทำงานเต็มอยู่ในลูปที่ไม่สิ้นสุด ส่วนการหยุดการทำงานแบบที่ 2 จะเกิดขึ้นเมื่อทัวริงแมชีนหยุดการทำงานที่สถานะสุดท้ายพอดี ซึ่งถ้าสนใจว่าทัวริงแมชีนไม่คำนึงว่าอินพุตสตริงที่อ่านเข้ามาจะถูกอ่านหมดหรือไม่ ขอเพียงให้การทำงานไปหยุดอยู่ที่สถานะสุดท้ายก็จะถือว่าทัวริงแมชีนยอมรับอินพุตสตริงที่อยู่บนเทปนั้นทันที ทั้งนี้เนื่องจากหัวอ่านสามารถเดล่อนที่ไปทางซ้ายหรือขวาได้ ดังนั้นจึงเป็นเรื่องยากที่จะบอกว่าอินพุตถูกอ่านหมดหรือยัง

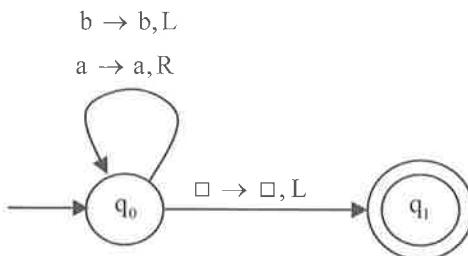
นอกจากนี้สถานะสุดท้ายของทัวริงแมชีนจะมีทรานซิชันที่ว่างเข้าได้อ่าย่างเดียวเท่านั้น ไม่มีทรานซิชันที่ว่างออกจากการสถานะสุดท้าย เช่น



ไม่อนุญาต :

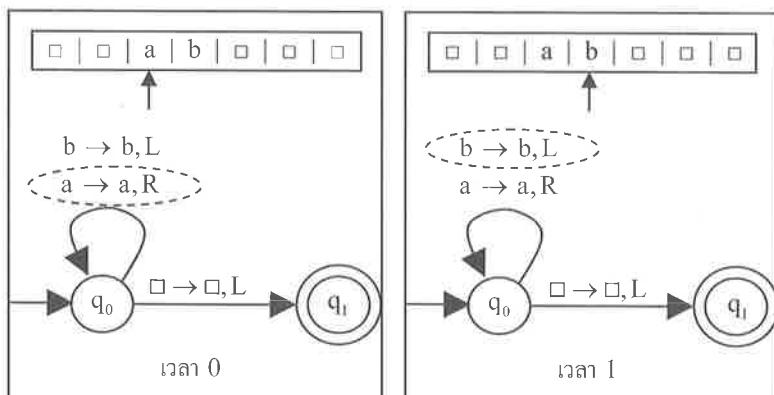


ตัวอย่างที่ 8.2 จงยกตัวอย่างทัวริ่งแมชีนที่ทำงานด้วยในลูปที่ไม่ลื้นสุด

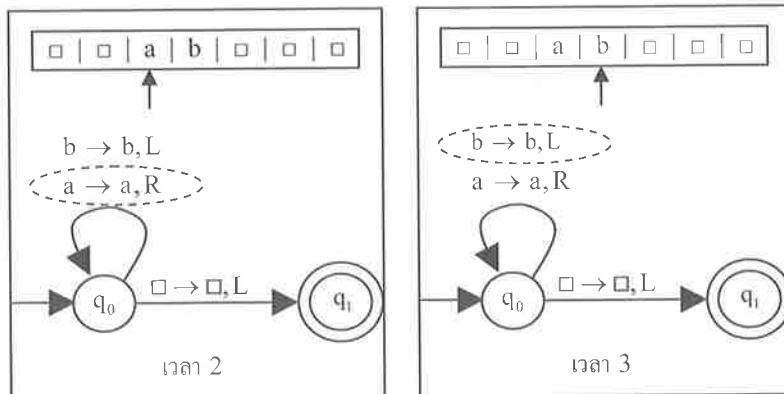


รูปที่ 8.5 ทัวริ่งแมชีนที่ทำงานด้วยในลูปไม่ลื้นสุด

ทัวริ่งแมชีนนี้เมื่อพับอินพุต b จะเขียน b ทั้ง (ซึ่งเท่ากับไม่ได้เปลี่ยนแปลงอินพุต b) จากนั้นเคลื่อนหัวอ่านไปทางซ้าย หากพับอินพุต a ก็จะเขียน a ทั้ง แล้วเคลื่อนหัวอ่านไปทางขวา จนพับซ่องว่างก็จะเข้าสู่สถานะสุดท้าย จะเห็นได้ว่าหากอินพุตที่เข้ามาเป็น ab จะทำให้ทัวริ่งแมชีนทำงานอยู่ในลูปที่ไม่ลื้นสุด โดยจะเคลื่อนหัวอ่านไปทางขวาและซ้ายกลับไปกลับมาดังรูป



รูปที่ 8.6 การทำงานของทัวริ่งแมชีนที่ขยับเวลาต่อ ๆ



รูปที่ 8.6 (ต่อ)

ตัวอย่างที่ 8.3 จงอธิบายการทำงานของทัวริงแมชีนดังต่อไปนี้

$$M = (\{q_0, q_f\}, \{a, b\}, \{A, B, \square\}, \delta, q_0, \square, \{q_f\})$$

$$\delta: \delta(q_0, a) = (q_0, A, R),$$

$$\delta(q_0, b) = (q_0, B, R),$$

$$\delta(q_0, \square) = (q_f, \square, L)$$

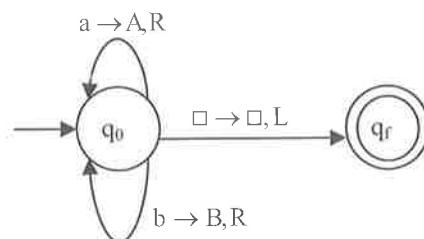
ในการศึกษาการทำงานของทัวริงแมชีน เราจะต้องศึกษาการทำงานของทรานซิชันฟังก์ชันทั้งหมดก่อน

$\delta(q_0, a) = (q_0, A, R)$ ที่สถานะเริ่มต้น q_0 หากอินพุตเป็น a ให้เปลี่ยนเป็นตัวอักษรพิมพ์ใหญ่ A และเคลื่อนหัวอ่านไปทางขวา 1 ตำแหน่ง ส่วนสถานะยังคงเดิม

$\delta(q_0, b) = (q_0, B, R)$ ที่สถานะเริ่มต้น q_0 หากอินพุตเป็น b ให้เปลี่ยนเป็นตัวอักษรพิมพ์ใหญ่ B และเคลื่อนหัวอ่านไปทางขวา 1 ตำแหน่ง ส่วนสถานะยังคงเดิม

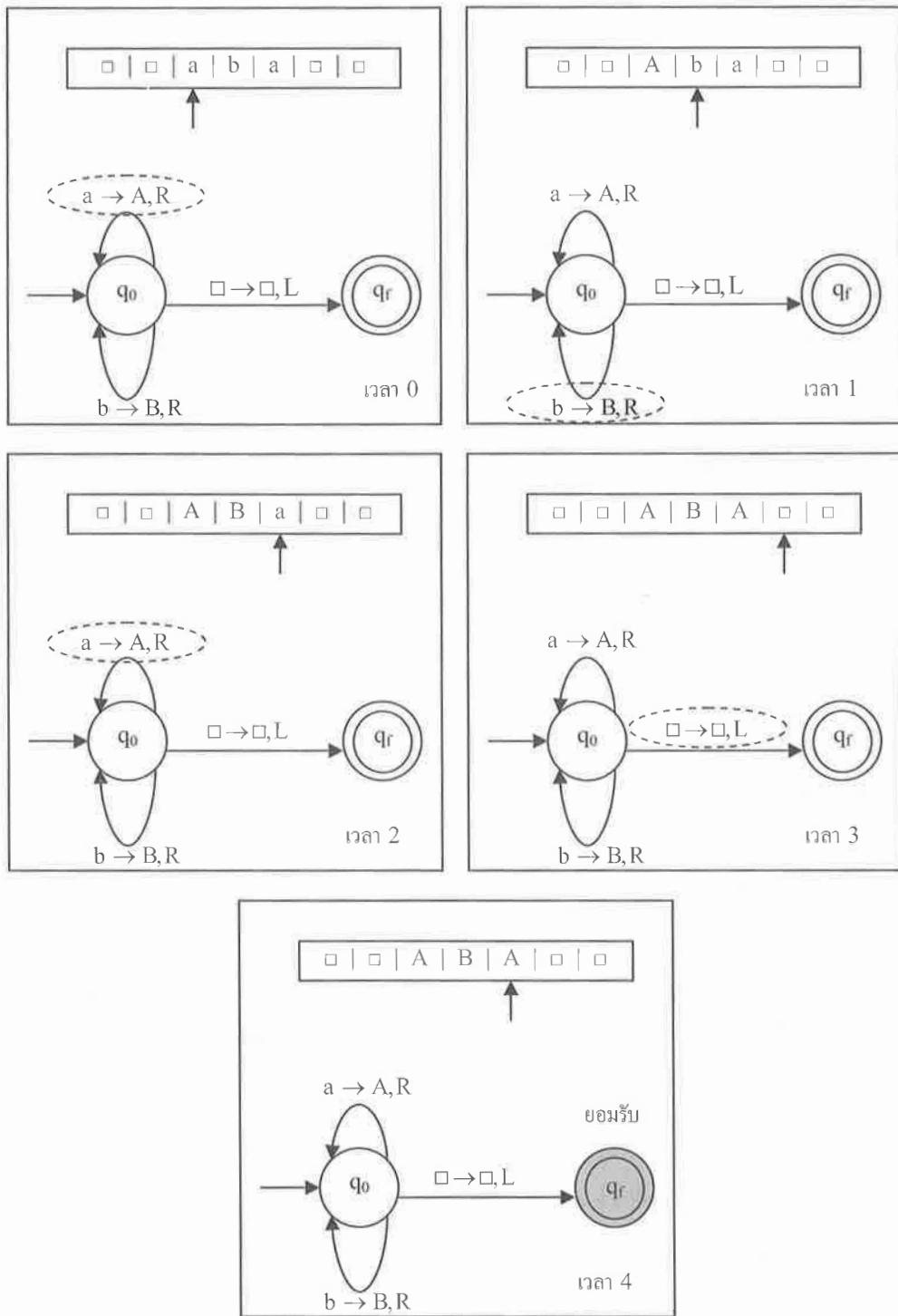
$\delta(q_0, \square) = (q_f, \square, L)$ ที่สถานะเริ่มต้น q_0 หากอินพุตเป็นช่องว่าง (แสดงจุดสิ้นสุดของอินพุต) ให้เปลี่ยนสถานะเข้าสู่สถานะสุดท้าย q_f และเคลื่อนหัวอ่านมาทางซ้าย 1 ตำแหน่ง โดยปล่อยให้ช่องว่างอยู่เหมือนเดิม

สามารถสร้างทรานซิชันกราฟแสดงการทำงานของทัวริงแมชีนได้ดังนี้



รูปที่ 8.7 ทรานซิชันกราฟแสดงการทำงานของทัวริงแมชีน

จากการทำงานของทรานซิชันฟังก์ชันทั้งสามข้อ พолжะสรุปได้ว่าทัวริงแมชีนนี้เปลี่ยนตัวอักษรพิมพ์เล็ก (a, b) บนอินพุตเทป ให้กลายเป็นตัวอักษรพิมพ์ใหญ่ (A, B) ดังตัวอย่างต่อไปนี้



รูปที่ 8.8 การทำงานของทัวริงแมชีนที่เวลาต่าง ๆ

หากต้องการอธิบายด้วยการอธิบาย ณ ชั่วขณะะเวลาหนึ่ง (Instantaneous Description) สามารถอธิบายได้ดังนี้

$$\begin{array}{l} q_0aba \vdash Aq_0ba \vdash ABq_0a \\ \quad \vdash ABAq_0 \vdash ABq_1A \end{array}$$

หรือแสดงการอธิบาย ณ ชั่วขณะะเวลาหนึ่งแบบย่อได้เป็น

$$q_0aba \vdash^* ABq_1A$$

8.2 ภาษาที่ทัวริงแมปเป็นรองรับ

นิยาม 8.2.1 : กำหนดให้ทัวริงแมปเป็น $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ ภาษาของทัวริงแมปเป็น M ถูกนิยามโดย

$$L(M) = \{s \in \Sigma^* - \{\epsilon\} : q_0s \vdash^* s_1q_f s_2 \text{ โดยที่ } q_f \in F \text{ และ } s_1, s_2 \in \Gamma^*\}$$

ภาษาของทัวริงแมปเป็น M ประกอบด้วยสตริง s ซึ่งเกิดจาก $\Sigma^* - \{\epsilon\}$ จะเห็นได้ว่าทัวริงแมปเป็นโดยทั่วไปจะไม่รับสตริงว่าง (ϵ) เนื่องจากจะเกิดความสับสนในการระบุขอบเขตของภาษาของสตริงว่างที่ถูกล้อมด้วยช่องว่าง \square แต่หากเราต้องการรับสตริงว่างจริง ๆ ก็สามารถใช้ไฟฟ้าโนต์อโตมาตาหรือพุทธาน័งอโตมาตราของรับได้

นอกเหนือนี้ จากนิยามของ $L(M)$ หากเรามีอัน s ลงไว้บนอินพุตเทปโดยเริ่มต้นจากสถานะ q_0 และปล่อยให้เกิดการเปลี่ยนแปลงขึ้น ถ้าทัวริงแมปเป็นทำงานจนหยุดที่สถานะ q_f ซึ่งเป็นสถานะสุดท้ายเราจะถือว่า s เป็นสตริงที่ถูกของรับโดยทัวริงแมปเป็น M โดยไม่สนใจว่าสตริง s_1 และ s_2 จะมีลักษณะเป็นอย่างไร

ตัวอย่างที่ 8.4 จงสร้างทัวริงแมปเป็นสำหรับภาษาดังต่อไปนี้

$$L = \{0^n 1^n : n \geq 1\}$$

วิธีทำ :

แนวคิด : เช่นเดียวกับการเขียนโปรแกรมคอมพิวเตอร์ให้ทำงานหนึ่ง ๆ ก่อนจะสร้างทัวริงแมปเป็น จะต้องเริ่มต้นด้วยการสร้างอัลกอริทึมสำหรับการรับสตริงที่อยู่ภายใต้ภาษา L ตัวอย่างของสตริงในภาษา L เช่น 01, 0011, 000111 เป็นต้น แนวคิดคือพยายามสร้างความสมดุลระหว่างจำนวนของ 0 และ 1 โดยหากเจอ 0 ให้เปลี่ยนจาก 0 เป็น a และหากเจอ 1 ให้เปลี่ยนจาก 1 เป็น b เคลื่อนหัวอ่านไป-กลับเพื่อจับคู่ของ 0 และ 1 ทีละคู่จนหมด แนวคิดนี้สามารถแนบได้ด้วยการอธิบาย ณ ชั่วขณะะเวลาหนึ่งดังนี้

$$\begin{array}{l} q_00011 \vdash aq_1011 \vdash a0q_111 \vdash aq_20b1 \vdash \\ q_2a0b1 \vdash aq_00b1 \vdash aaq_1b1 \vdash aabq_11 \vdash \\ aaq_2bb \vdash aq_2abb \vdash aaq_0bb \vdash aabq_3b \vdash \\ aabbq_3 \vdash aabq_rb \end{array}$$

สร้างทั่วเริงแมชีน M สำหรับภาษา L ได้ดังนี้

$$M = (\{q_0, q_1, q_2, q_3, q_f\}, \{0, 1\}, \{0, 1, a, b, \square\}, \delta, q_0, \square, \{q_f\})$$

$$\delta : \quad \delta(q_0, 0) = (q_1, a, R),$$

$$\delta(q_1, 0) = (q_1, 0, R),$$

$$\delta(q_1, b) = (q_1, b, R),$$

$$\delta(q_1, 1) = (q_2, b, L),$$

$$\delta(q_2, b) = (q_2, b, L),$$

$$\delta(q_2, 0) = (q_2, 0, L),$$

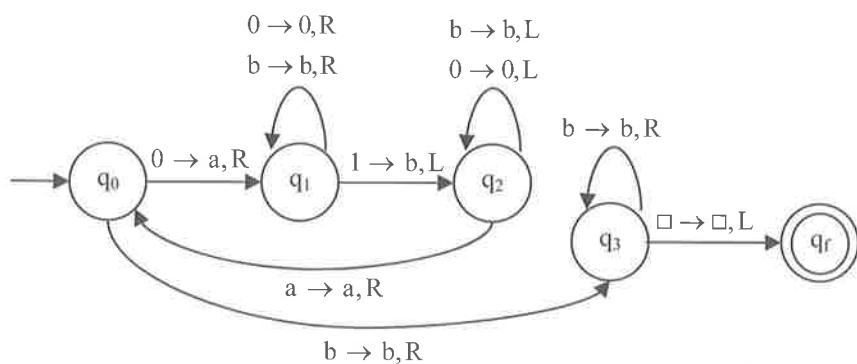
$$\delta(q_2, a) = (q_0, a, R),$$

$$\delta(q_0, b) = (q_3, b, R),$$

$$\delta(q_3, b) = (q_3, b, R),$$

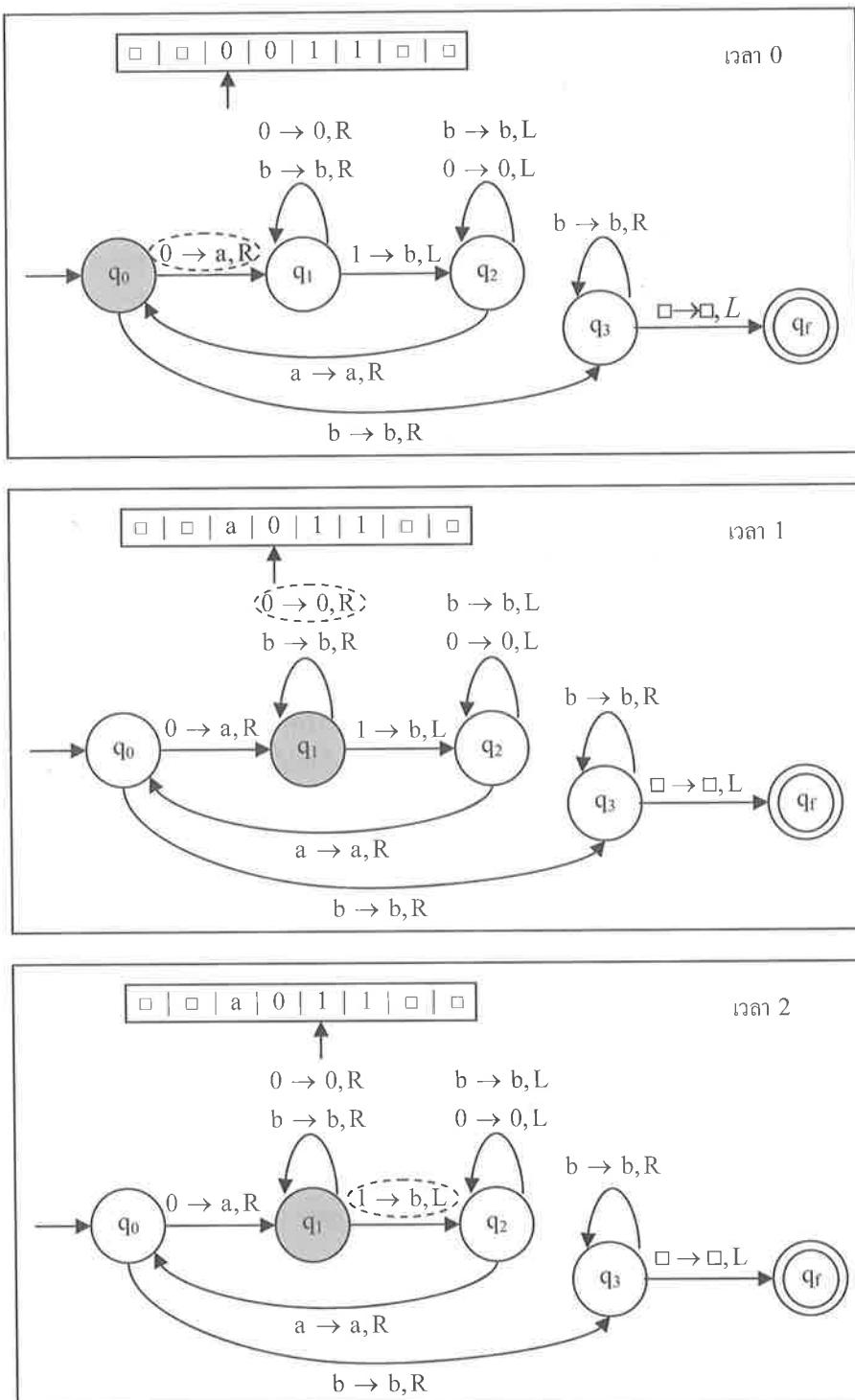
$$\delta(q_3, \square) = (q_f, \square, L)$$

นอกจากนี้ ยังสามารถแสดงทั่วเริงแมชีน M ในรูปของกราฟได้ดังนี้

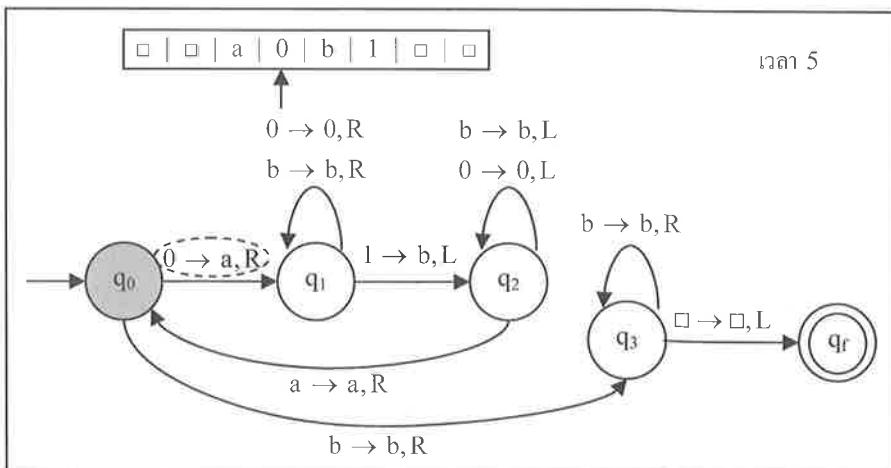
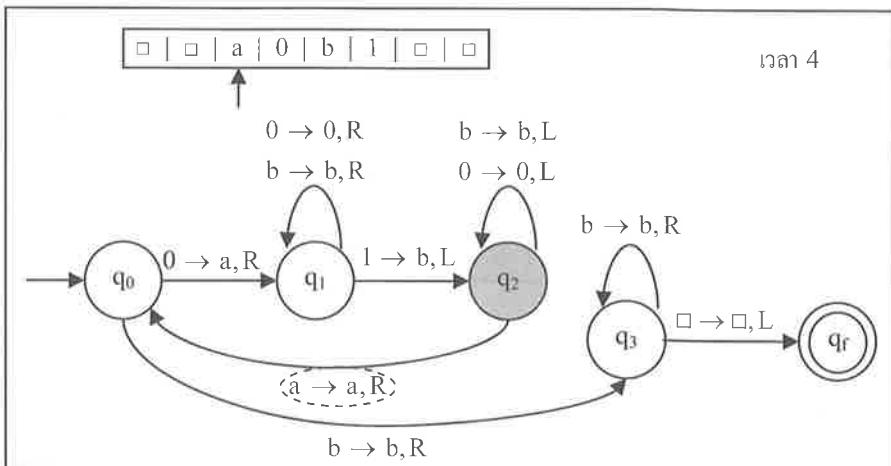
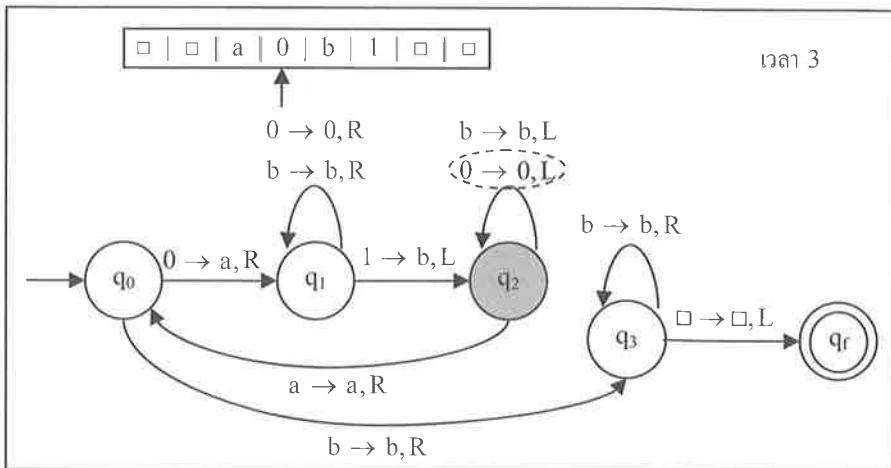


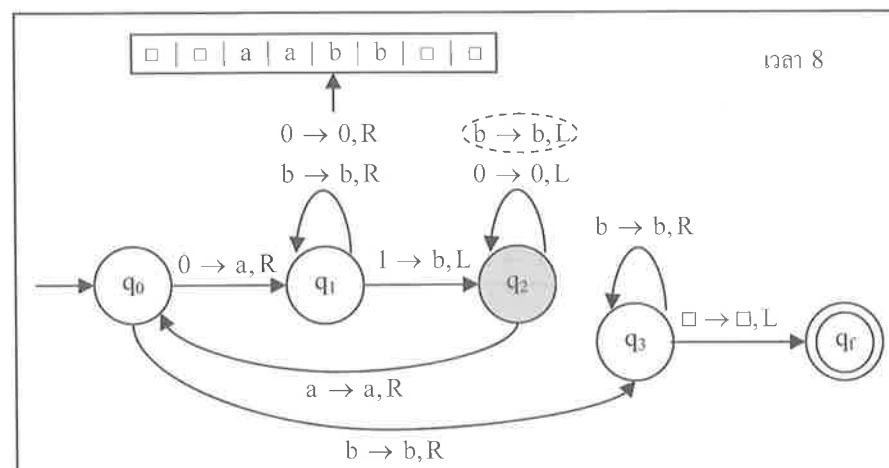
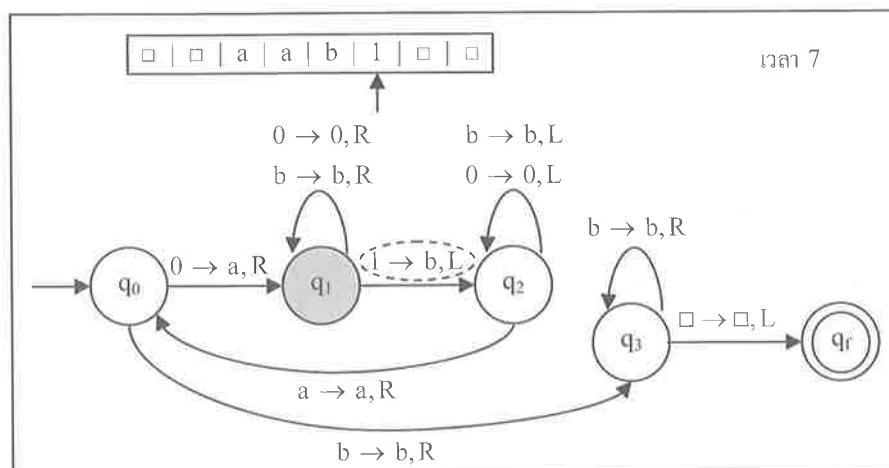
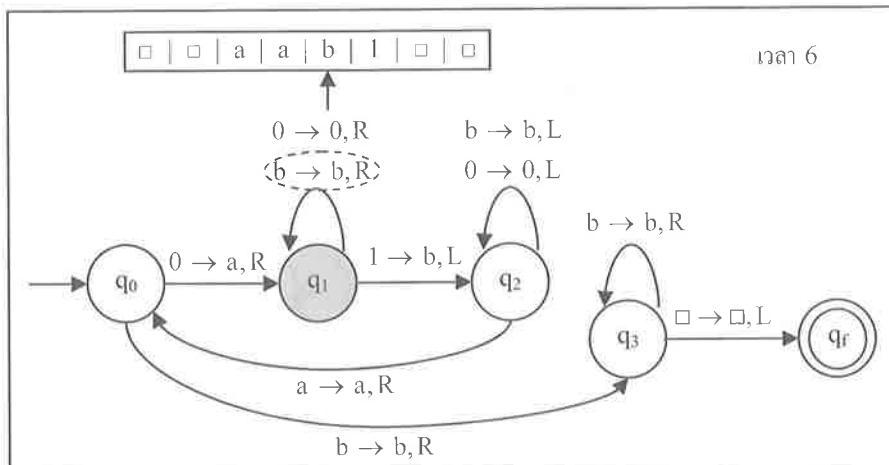
รูปที่ 8.9 ทั่วเริงแมชีน M

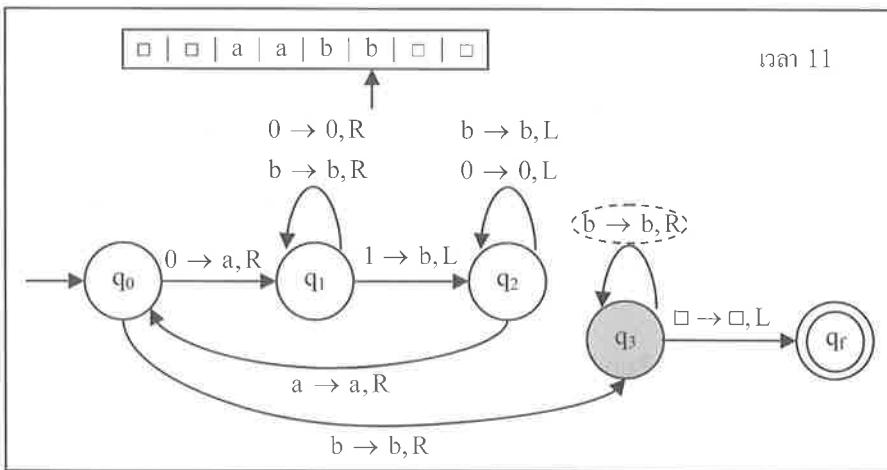
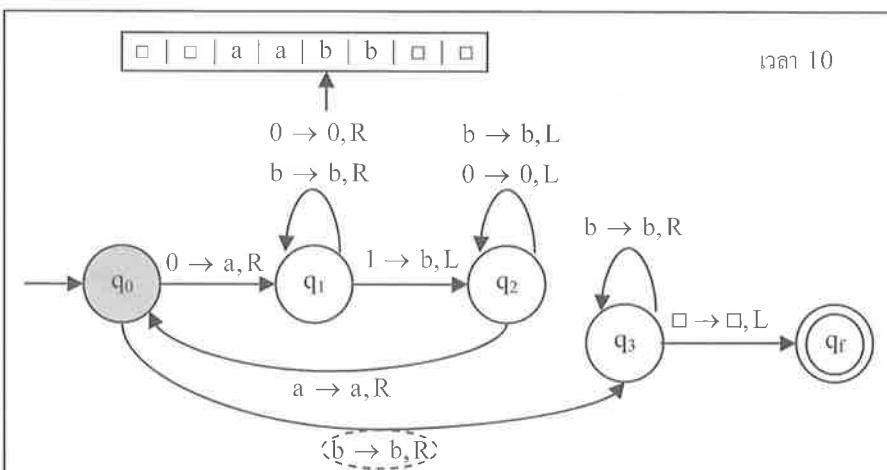
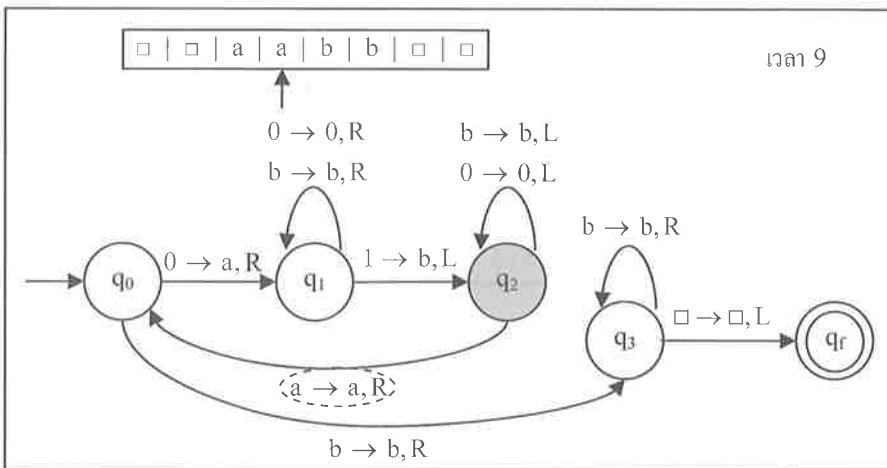
เพื่อเป็นการเสริมความเข้าใจในการทำงานของทั่วเริงแมชีน M การจำลองการทำงานของทั่วเริงแมชีนนี้กับอินพุตสตริง 0011 สามารถแสดงได้ตามลำดับของเวลาดังนี้

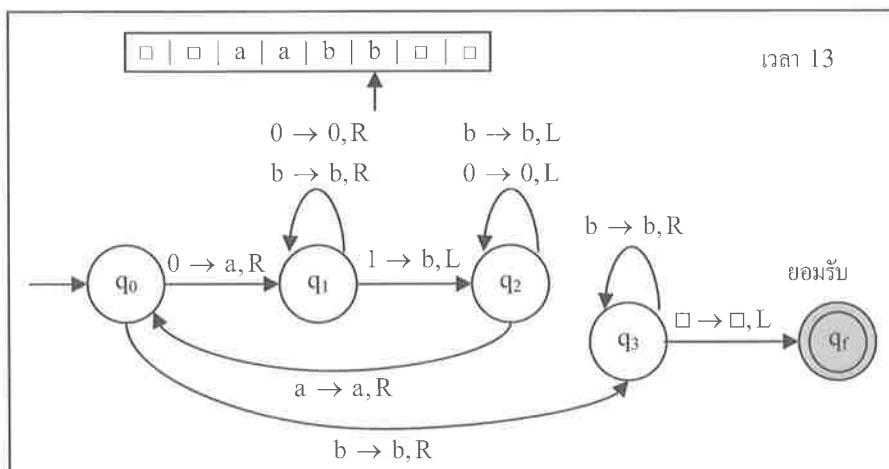
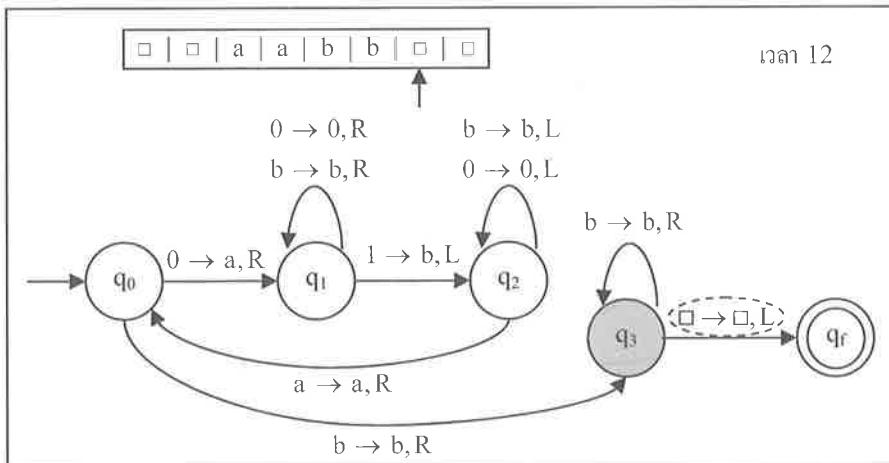


รูปที่ 8.10 การทำงานของทัวริงแมชีนที่เวลาต่าง ๆ









รูปที่ 8.10 (ต่อ)

จากการทำงานของทัวริ่งแมชีนในตัวอย่างนี้ เราสามารถประยุกต์แนวความคิดเดียวกันในการสร้างทัวริ่งแมชีนที่ยอมรับภาษา $\{a^n b^n c^n\}$ ซึ่งไม่ใช่ภาษาค่อนเท็กซ์ฟรี เนื่องจากเรามิ่งสามารถสร้างพุชดาวน์อโตมาตาเพื่อรองรับได้ สำหรับการสร้างทัวริ่งแมชีนเพื่อรองรับภาษา $\{a^n b^n c^n\}$ ผู้เขียนจะให้เป็นแบบฝึกหัดสำหรับผู้อ่าน ซึ่งอย่างน้อยผู้อ่านควรทำความเข้าใจหลักการสำหรับแก้ปัญหา แต่ไม่จำเป็นต้องสร้างทรานซิชันให้ครบทั้งหมดเนื่องจากทัวริ่งแมชีนของภาษานี้มีขนาดใหญ่มาก

ตัวอย่างที่ 8.5 จงสร้างทัวริ่งแมชีนสำหรับภาษาดังต่อไปนี้

$$L = \{w \in \{a, b\}^+ : n_a(w) = n_b(w)\}$$

วิธีทำ :

แนวคิด : สร้างทัวริ่งแมชีนให้หัวอ่านเคลื่อนที่จากซ้ายไปขวาโดยแยกเป็น 2 กรณี

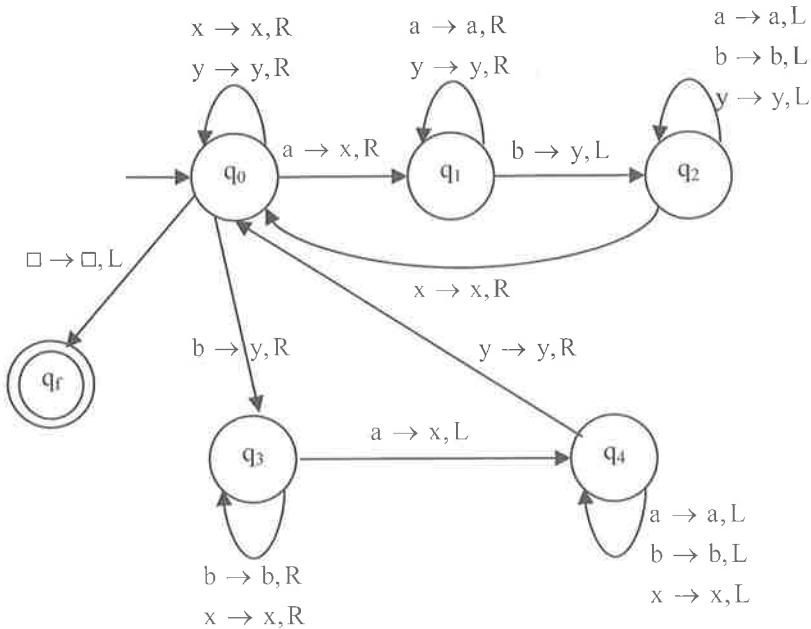
- (1) หากเจอ a ให้เปลี่ยน a เป็น x จากนั้นเคลื่อนหัวอ่านไปทางขวาเรื่อย ๆ จนกระทั่งเจอ b ก็ให้เปลี่ยน b เป็น y และเคลื่อนหัวอ่านกลับมาทางซ้ายจนเลอซ่องว่าง จากนั้นก็จะไปทางขวาเพื่อค้นหา a และ b คู่ต่อไป
- (2) หากเจอ b ให้เปลี่ยน b เป็น y จากนั้นเคลื่อนหัวอ่านไปทางขวาเรื่อย ๆ จนกระทั่งเจอ a ก็ให้เปลี่ยน a เป็น x และเคลื่อนหัวอ่านกลับมาทางซ้ายจนเลอซ่องว่าง จากนั้นก็จะไปทางขวาเพื่อค้นหา a และ b คู่ต่อไป

เงื่อนไขในการจับการทำงานคือ หากวิ่งมาทางขวาเพื่อหาคู่ของ a และ b แต่ไม่เจอ a หรือ b เลย แต่กลับไปเจอซ่องว่าง แสดงว่าอินพุตสตริงนั้นมีจำนวนของ a และ b เท่ากัน ก็ให้จบการทำงานโดยเข้าสู่สถานะสุดท้าย

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_f\}, \{a, b\}, \{a, b, x, y, \square\}, \delta, q_0, \square, \{q_f\})$$

$$\begin{aligned} \delta: \quad \delta(q_0, a) &= (q_1, x, R), \\ \delta(q_0, b) &= (q_3, y, R), \\ \delta(q_0, x) &= (q_0, x, R), \\ \delta(q_0, y) &= (q_0, y, R), \\ \delta(q_1, a) &= (q_1, a, R), \\ \delta(q_1, y) &= (q_1, y, R), \\ \delta(q_1, b) &= (q_2, y, L), \\ \delta(q_2, a) &= (q_2, a, L), \\ \delta(q_2, b) &= (q_2, b, L), \\ \delta(q_2, y) &= (q_2, y, L), \\ \delta(q_2, x) &= (q_0, x, R), \\ \delta(q_3, a) &= (q_4, x, L), \\ \delta(q_3, b) &= (q_3, b, R), \\ \delta(q_3, x) &= (q_3, x, R), \\ \delta(q_4, a) &= (q_4, a, L), \\ \delta(q_4, b) &= (q_4, b, L), \\ \delta(q_4, x) &= (q_4, x, L), \\ \delta(q_4, y) &= (q_0, y, R), \\ \delta(q_0, \square) &= (q_f, \square, L) \end{aligned}$$

สำหรับทรานซิชันกราฟแทนทัวริงแมชีน M ในตัวอย่างนี้ คือ



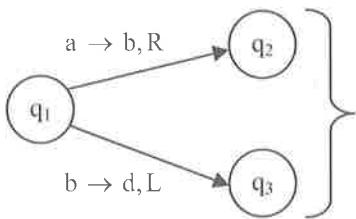
รูปที่ 8.11 ทัวริงแมชีน M

สามารถจำลองการทำงานของทัวริงแมชีน M โดยป้อนอินพุตเป็น bbabaa

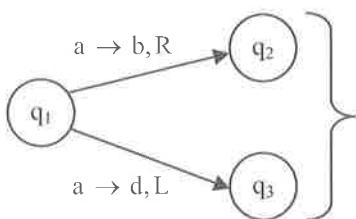
$$\begin{aligned}
 &q_0bbabaa \vdash yq_3babaa \vdash ybq_3abaa \vdash yq_4bxbaa \vdash \\
 &q_4ybxbaa \vdash yq_0bxbaa \vdash yyq_3xbaa \vdash yyxq_3baa \vdash \\
 &yyxbq_3aa \vdash yyxq_4bxa \vdash yyq_4xbxa \vdash yq_4yxbsa \vdash \\
 &yyq_0xbxa \vdash yyxq_0bxa \vdash yyxyq_3xa \vdash yyxyxq_3a \vdash \\
 &yyxyq_4xx \vdash yyxq_4yxx \vdash yyxyq_0xx \vdash yyxyxq_0x \vdash \\
 &yyxyxxq_0 \vdash yyxyxq_fx
 \end{aligned}$$

8.3 กัวริงแมชีนที่คาดเดาไม่ได้

ทัวริงแมชีนจากตัวอย่างที่ผ่านมาล้วนเป็นทัวริงแมชีนที่ทำงานแบบคาดเดาพฤษติกรรมได้ (Deterministic Turing Machine) นั่นคือ ณ สถานะและอินพุตใด ๆ ทรานซิชันฟังก์ชันจะทำให้เกิดการเปลี่ยนแปลงໄດ้เพียง 1 รูปแบบเท่านั้น เช่น $\delta(q_0, a) = (q_1, x, R)$ ซึ่งทัวริงแมชีนในลักษณะนี้จะมีปัญหาภัยภัยที่มีความซับซ้อนมากขึ้น ซึ่งอาจทำให้ทัวริงแมชีนมีขนาดใหญ่มากและทำให้มีสัดส่วนในการทำงาน ดังนั้นจึงต้องการทัวริงแมชีนที่มีความคล่องตัวกว่าทัวริงแมชีนที่คาดเดาได้



อนุญาตให้เกิดขึ้นได้ทั้งในทั่วเริงแมชีนที่ค่าเดาได้ และค่าเดาไม่ได้



ไม่อนุญาตให้เกิดขึ้นในทั่วเริงแมชีนที่ค่าเดาได้ แต่สามารถเกิดขึ้นได้ในทั่วเริงแมชีนที่ค่าเดาไม่ได้ เนื่องจากที่สถานะ q_1 และอินพุตเป็น a ทั่วเริงแมชีนสามารถเลือกไปที่สถานะ q_2 หรือ q_3 ก็ได้ นอกจากนี้ ทั่วเริงแมชีนที่ค่าเดาได้ไม่อนุญาตให้เกิดทรานซิชันโดยไม่อ่านอินพุตเข้ามา ซึ่งแตกต่างจากทั่วเริงแมชีนที่ค่าเดาไม่ได้ที่อนุญาตให้ทรานซิชันสตริงว่างเกิดขึ้นได้

นิยาม 8.3.1 : ทั่วเริงแมชีนที่ค่าเดาไม่ได้ (Nondeterministic Turing Machine) มีลักษณะคล้ายกับทั่วเริงแมชีนที่ค่าเดาได้ ยกเว้นทรานซิชันฟังก์ชันซึ่งถูกนิยามดังนี้

$$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{R, L\}}$$

เนื่องจาก $2^{Q \times \Gamma \times \{R, L\}}$ คือ เชตของ การเปลี่ยนแปลงทั้งหมดที่เป็นไปได้ภายในทั่วเริงแมชีน ดังนั้นความสัมพันธ์ของทรานซิชันฟังก์ชันจึงเป็นแบบ 1 : m (หรือ one-to-many) ทำให้ทั่วเริงแมชีนในลักษณะนี้ไม่สามารถค่าเดาการเปลี่ยนแปลงได้ ซึ่งเป็นผลติดกันทั่วเริงแมชีนที่ค่าเดาไม่ได้ เนื่องจากเราไม่จำเป็นต้องใช้ทรานซิชันฟังก์ชันจำนวนมากในการอธิบายภาษา

ตัวอย่างที่ 8.6 จงสร้างทั่วเริงแมชีนสำหรับภาษา

$$L = \{ww : w \in \{a, b\}^+\}$$

วิธีทำ :

แนวคิด : ภาษานี้เป็นภาษาที่พูดดาวน์อัล莫ตาไม่สามารถรับได้ เนื่องจากพฤติกรรมของสแต็กซึ่งมีการทำงานแบบเข้าที่หลัง-ออกก่อน (Last-In First-Out) ดังนั้นภาษาหนึ่งไม่ใช่ภาษาคอนเทกซ์ฟรี หากใช้ทั่วเริงแมชีนที่ค่าเดาได้ในการแก้ปัญหาโจทย์ข้อนี้ จะต้องใช้ชุดกับปัญหาการเดาจุดกึ่งกลางของสตริง ww ซึ่งแม้จะแก้ปัญหาได้แต่จะทำให้ทั่วเริงแมชีนที่ค่าเดาได้มีขนาดที่ใหญ่มาก แต่ถ้าใช้ทั่วเริงแมชีนที่ค่าเดาไม่ได้ จะสามารถแก้ปัญหาจุดกึ่งกลางของสตริง ww ได้อย่างง่ายดาย โดยอาศัยคุณสมบัติการค่าเดาพฤติกรรมไม่ได้ของทั่วเริงแมชีนชนิดนี้ให้เป็นประโยชน์

- (1) ในการหาจุดกึ่งกลางสตริง ww ให้สร้างทรานซิชันฟังก์ชันเพื่อเป็นทางเลือก โดยทางเลือกแรกใช้ สำหรับจุดที่เป็นกึ่งกลางสตริง ทางเลือกที่ 2 ใช้สำหรับจุดที่ยังไม่ถึงกึ่งกลางสตริง
- (2) แทนตำแหน่งเริ่มต้นของสตริง>y อย w แรกด้วยตัวอักษร x และแทนตำแหน่งเริ่มต้นของสตริง>y อย w หลังด้วยตัวอักษร y (ดังนั้น x แทนจุดเริ่มต้นของสตริง-w ส่วนแรก และ y แทนจุดเริ่มต้นของสตริง w ส่วนหลัง)

- (3) ตรวจสอบความเหมือนกันของสตริง w ทั้งสองส่วน โดยหากเจอ a ให้แทนด้วย 0 และหากเจอ y หัวอ่านไปทางขวาผ่าน y จนกระทั่งเจอ a แล้วให้เปลี่ยน a เป็น 0 สำหรับกรณี b ก็ทำคล้ายกัน แต่เปลี่ยน b เป็น 1

เราสามารถจำลองการทำงานของทัวริงแมชีน โดยป้อนอินพุตเป็น abbabbb

$$\begin{aligned}
 q_0abbabb &\dashv xq_1bbabb \dashv xbq_1babbb \dashv xbbq_1abb \dashv xbq_3bybb \dashv \\
 xq_3bbbybb &\dashv q_3xbbbybb \dashv xq_4bbbybb \dashv x1q_8bybb \dashv x1bq_8ybb \dashv \\
 x1byq_9bb &\dashv x1bq_9y1b \dashv x1q_7by1b \dashv xq_71by1b \dashv q_7x1by1b \dashv \\
 x q_41by1b &\dashv x1q_4by1b \dashv x11q_8y1b \dashv x11yq_91b \dashv x11y1q_9b \dashv \\
 x11yq_711 &\dashv x11q_7y11 \dashv x1q_71y11 \dashv xq_711y11 \dashv q_7x11y11 \dashv \\
 xq_411y11 &\dashv x1q_41y11 \dashv x11q_4y11 \dashv x11yq_{10}11 \dashv x11y1q_{10}1 \dashv \\
 x11y11q_{10} &\dashv x11y1q_f1
 \end{aligned}$$

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_f\},$$

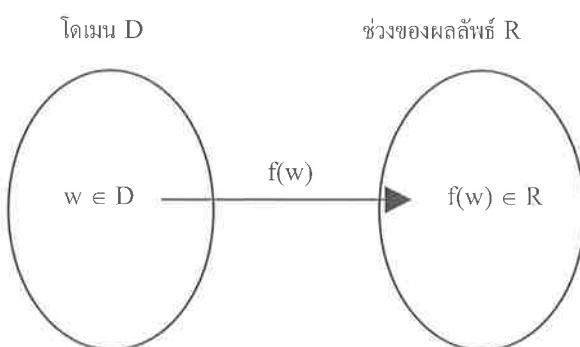
$$\{a, b\}, \{a, b, x, y, 0, 1, \square\}, \delta, q_0, \square, \{q_f\})$$

$\delta : \delta(q_0, a) = (q_1, x, R),$	/* กำหนดจุดเริ่มต้นของสตริงย่อย w ส่วนแรก */
$\delta(q_0, b) = (q_2, x, R),$	
$\delta(q_1, a) = \{(q_1, a, R), (q_3, y, L)\},$	
$\delta(q_1, b) = (q_1, b, R),$	
$\delta(q_2, b) = \{(q_2, b, R), (q_3, y, L)\},$	
$\delta(q_2, a) = (q_2, a, R),$	
$\delta(q_3, a) = (q_3, a, L),$	
$\delta(q_3, b) = (q_3, b, L),$	
$\delta(q_3, x) = (q_4, x, R),$	/* เสริ่งลืนการกำหนดจุดเริ่มต้นของสตริงย่อย w ทั้งสองส่วน */
$\delta(q_4, a) = (q_5, 0, R),$	/* แทน a ด้วย 0 และสถานะเป็น q_5 */
$\delta(q_4, b) = (q_8, 1, R),$	/* แทน b ด้วย 1 และสถานะเป็น q_8 */
$\delta(q_4, y) = (q_{10}, y, R),$	/* แสดงว่าสตริงย่อย w ส่วนแรกหมดแล้ว */
$\delta(q_4, 0) = (q_4, 0, R),$	
$\delta(q_4, 1) = (q_4, 1, R),$	
$\delta(q_5, a) = (q_5, a, R),$	
$\delta(q_5, b) = (q_5, b, R),$	
$\delta(q_5, y) = (q_6, y, R),$	/* เมื่อพบกึ่งกลางสตริง ให้เปลี่ยนสถานะเป็น q_6 */
$\delta(q_6, 0) = (q_6, 0, R),$	
$\delta(q_6, 1) = (q_6, 1, R),$	
$\delta(q_6, b) = (q_6, b, R),$	
$\delta(q_6, a) = (q_7, 0, L),$	

$\delta(q_7, a) = (q_7, a, L),$
 $\delta(q_7, b) = (q_7, b, L),$
 $\delta(q_7, 0) = (q_7, 0, L),$
 $\delta(q_7, 1) = (q_7, 1, L),$
 $\delta(q_7, y) = (q_7, y, L),$
 $\delta(q_7, x) = (q_4, x, R),$
 $\delta(q_8, a) = (q_8, a, R),$
 $\delta(q_8, b) = (q_8, b, R),$
 $\delta(q_8, y) = (q_9, y, R),$ /* ถึงกี่กlongสตริง ให้เปลี่ยนสถานะเป็น q_9 */
 $\delta(q_9, 0) = (q_9, 0, R),$
 $\delta(q_9, 1) = (q_9, 1, R),$
 $\delta(q_9, a) = (q_9, a, R),$
 $\delta(q_9, b) = (q_7, 1, L),$
 $\delta(q_{10}, 0) = (q_{10}, 0, R),$
 $\delta(q_{10}, 1) = (q_{10}, 1, R),$
 $\delta(q_{10}, \square) = (q_5, \square, L)$

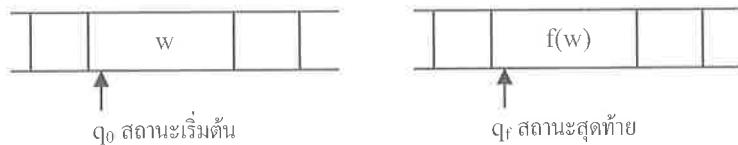
8.4 การใช้งานทั่วไปและเชิงเส้นเพื่อกำหนัดในฟังก์ชันการคำนวณ

นอกจากการใช้ทั่วไปและเชิงเส้นเป็นเครื่องมือรับรู้ภาษาแล้ว เรายังสามารถใช้ทั่วไปและเชิงเส้นเป็นเครื่องมือในการคำนวณค่าจากฟังก์ชันได้อีกด้วย กำหนดให้สตริง w เป็นสมาชิกอยู่ในโดเมน D ฟังก์ชัน $f(w)$ จะทำหน้าที่แปลงสตริง w ไปเป็นผลลัพธ์ซึ่งอยู่ภายในช่วง R ดังรูป



รูปที่ 8.12 ความสัมพันธ์ระหว่างโดเมนและช่วงของผลลัพธ์

นิยาม 8.4.1 : ฟังก์ชัน f สามารถคำนวณได้ (Computable) ถ้ามีทั่วไปและเชิงเส้น M ซึ่งสามารถรับอินพุตสตริง w จากนั้นให้ผลลัพธ์ $f(w)$ อยู่ในช่วงเดิม โดยที่ $w ∈ D$ และ $f(w) ∈ R$

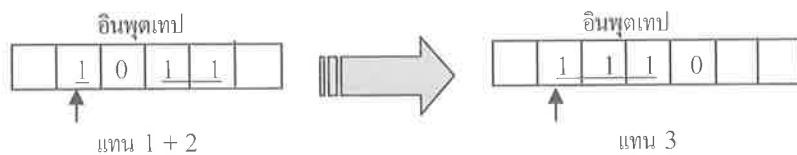


รูปที่ 8.13 ทัวริงแมชีนสำหรับฟังก์ชันที่สามารถคำนวณได้

ตัวอย่างที่ 8.7 จงออกแบบทัวริงแมชีนที่บวกตัวเลขจำนวนเต็มบวก 2 จำนวน $(x + y)$ ในรูปแบบของยูนารี (Unary Notation)

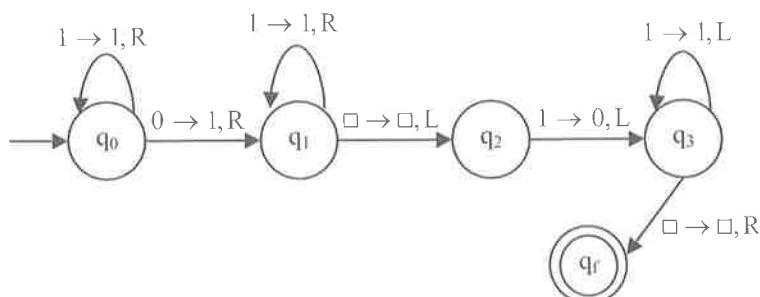
วิธีทำ : ก่อนอื่นจะต้องทำความเข้าใจตัวเลขในรูปแบบยูนารี การแทนตัวเลขในรูปแบบยูนารี (Unary Notation) คือ การใช้เลข 1 มาเรียงต่อกันเพื่อแทนจำนวนเต็มบวกที่ต้องการแทนค่า เช่น 5 สามารถแทนในรูปแบบของยูนารีได้คือ 1111 เป็นต้น

การบวกในรูปแบบของยูนารีสามารถทำได้โดยการย้ายตำแหน่งเลข 0 ซึ่งค้นอยู่ระหว่างกล่องของห้องสองจำนวนไปไว้ ณ ตำแหน่งท้ายสุด ดังนั้นอินพุตที่ลูกป้อนเข้ามาและเอาต์พุตจะมีลักษณะดังรูป



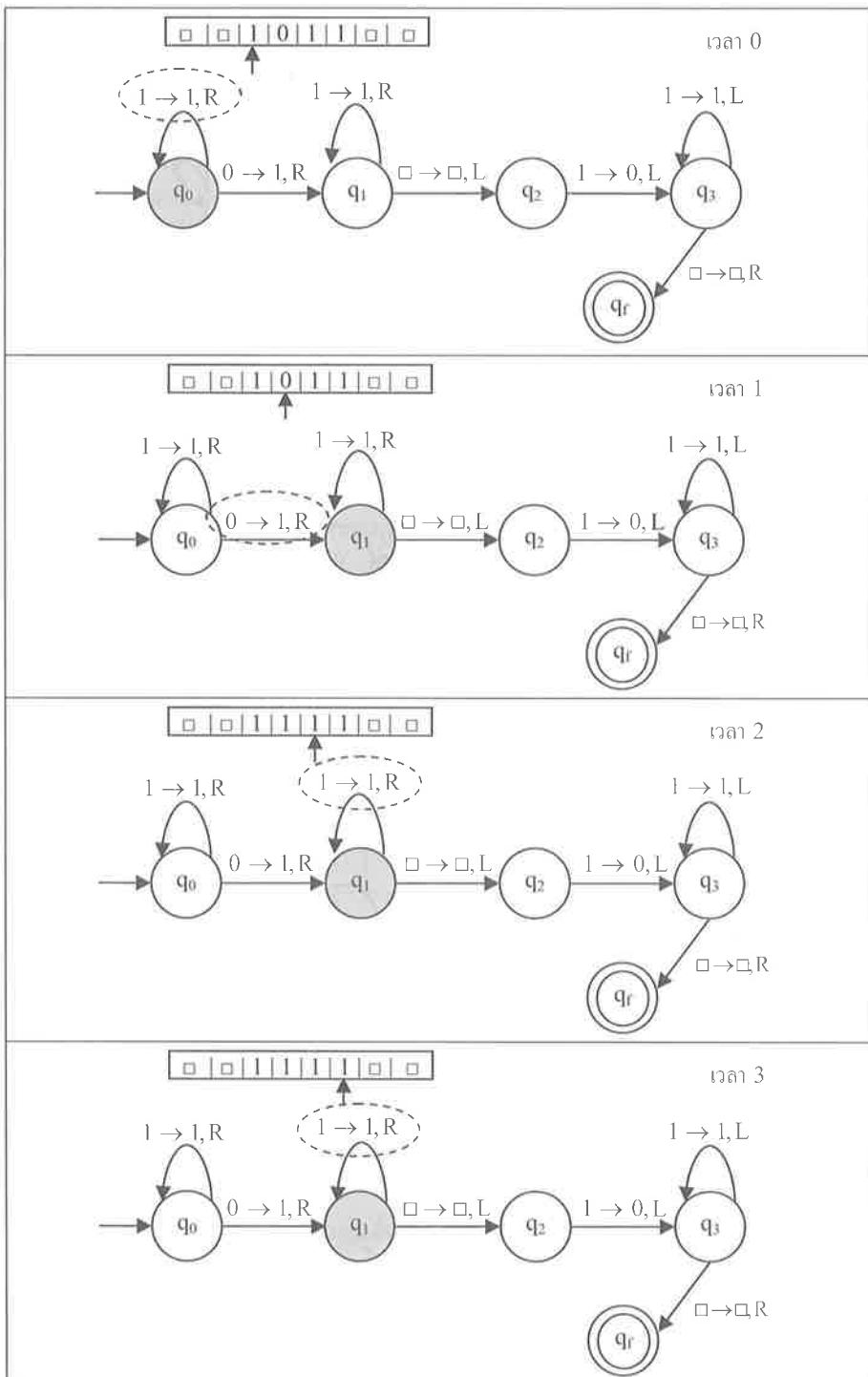
รูปที่ 8.14 การบวกเลขยูนารีในทัวริงแมชีน

จากหลักการตั้งกล่าวสามารถสร้างทัวริงแมชีนเพื่อร้องรับการทำงานของการบวกเลขแบบยูนารี 2 จำนวนดังนี้

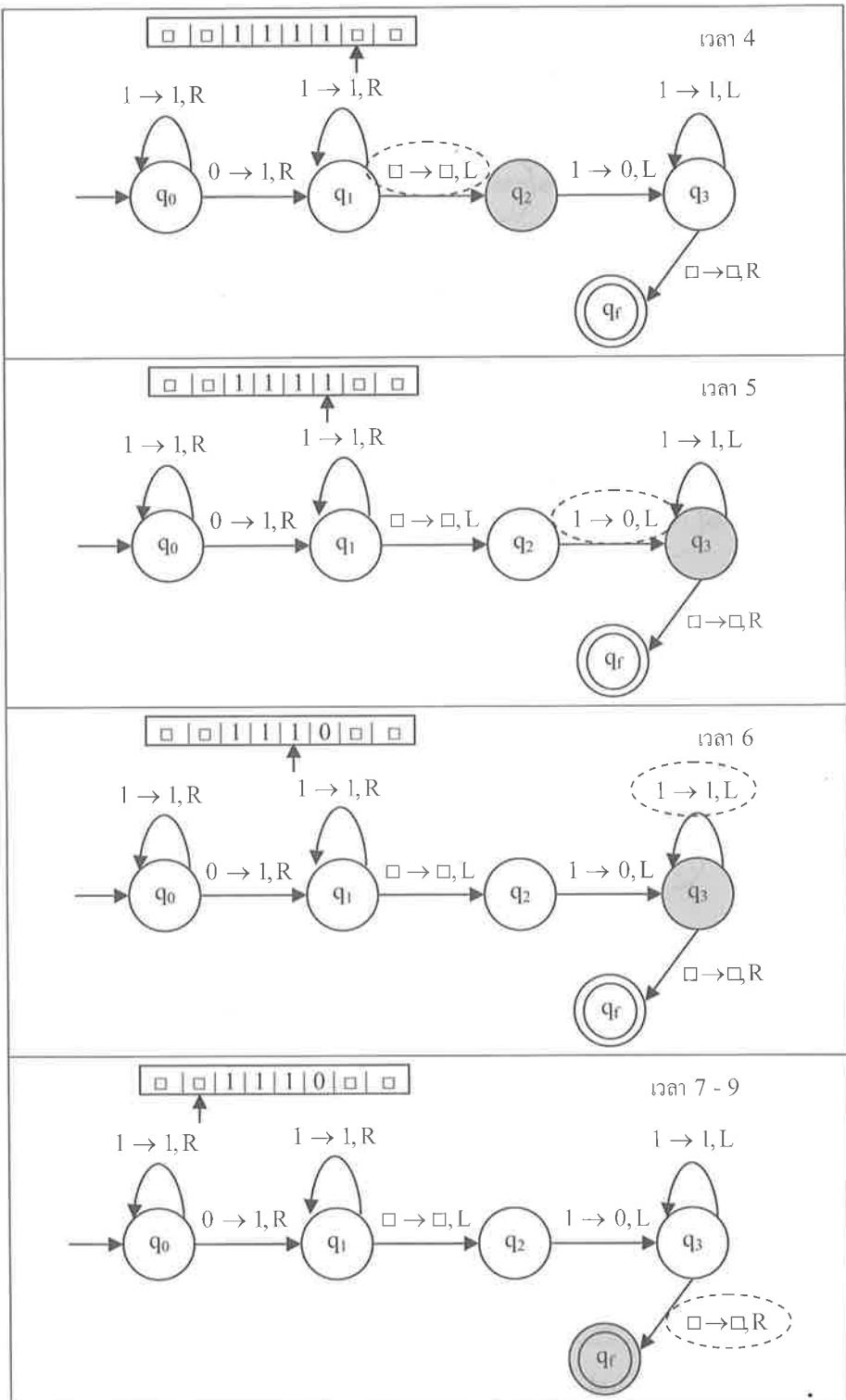


รูปที่ 8.15 ทัวริงแมชีนสำหรับการบวกเลขยูนารี

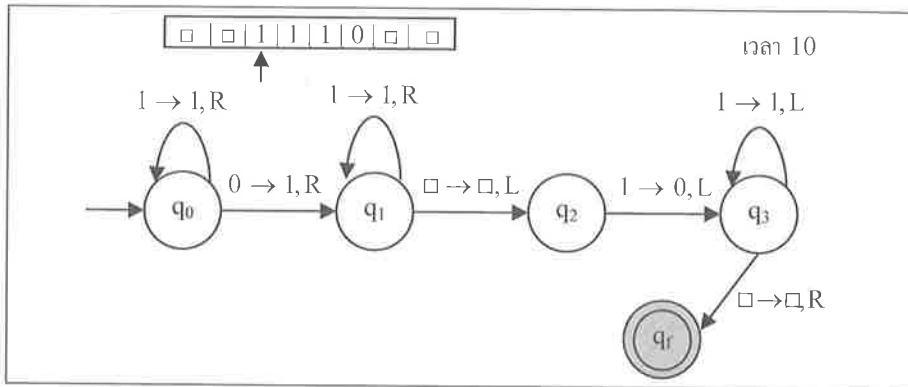
ตัวอย่างการทำงานของทัวริงแมชีนนี้แสดงได้โดยการป้อนอินพุต 1011 เข้าไป



รูปที่ 8.16 การทำงานของทั่วเริงแมชชีนที่เวลาต่าง ๆ
 (ที่มา : Konstantin Busch (<http://www.csc.lsu.edu/~busch/>)



ສັບຖິ່ນ 8.16 (ຕ່ອ)



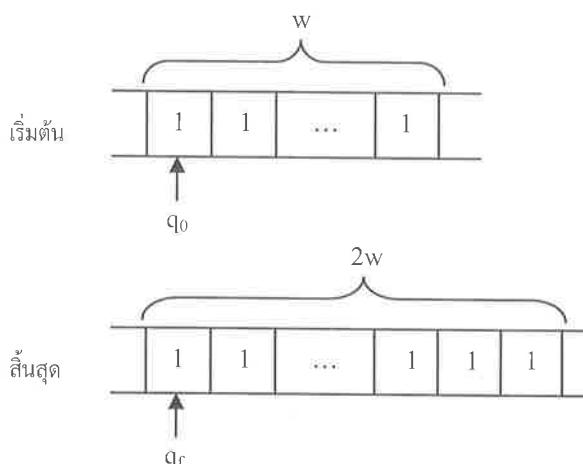
รูปที่ 8.16 (ต่อ)

ตัวอย่างที่ 8.8 กำหนดให้ฟังก์ชัน $f(w)$ ถูกนิยามดังนี้

$$f(w) = 2w \quad \text{โดยที่ } w \text{ คือ จำนวนเต็มในรูปแบบบัญญาตี}$$

จะพิสูจน์ว่าฟังก์ชัน $f(w)$ สามารถคำนวณได้

วิธีทำ : พิสูจน์โดยการสร้างทั่วเรียงแมชีนที่สามารถแปลงอินพุตสตริงซึ่งอยู่ในรูปแบบบัญญาตี เช่น 1111 ให้กลายเป็นผลลัพธ์ของฟังก์ชัน โดยสร้างอินพุตสตริงที่มีความยาวเป็นสองเท่าของเดิม เช่น จากอินพุต 1111 จะถูกแปลงไปเป็น 11111111 ตัวอย่างการทำงานในรูปแบบที่นำไปแสดงดังรูป



รูปที่ 8.17 การเริ่มต้นและสิ้นสุดของทั่วเรียงแมชีนสำหรับฟังก์ชัน $f(w)$

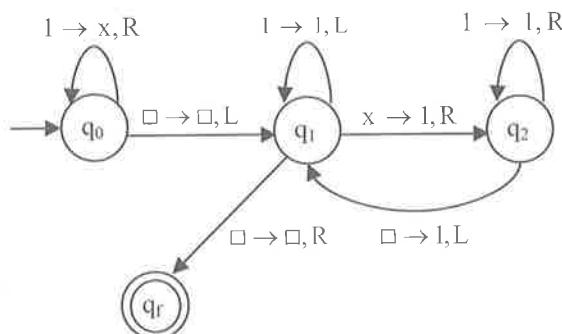
แนวคิด :

- แทนตัวเลข 1 ทุกตัวด้วยตัวอักษร x
- ทำซ้ำ

- คืนหาตัวอักษร x ตัวใดมีอุปสรรค และแทนตัวอักษร x นั้นด้วย 1
- เคลื่อนหัวอ่านไปยังช่องว่างของขาสุดของสตริงที่อยู่บนเทปและเขียนเลข 1 ทับลงไปบนช่องว่างนั้น

จึงกระทำทั้งไม่มีตัวอักษร x เหลืออยู่บนเทป

จากแนวคิดนี้ เราสามารถสร้างทัวริ่งแมชีนสำหรับรองรับฟังก์ชัน $f(w)$ ได้ดังนี้



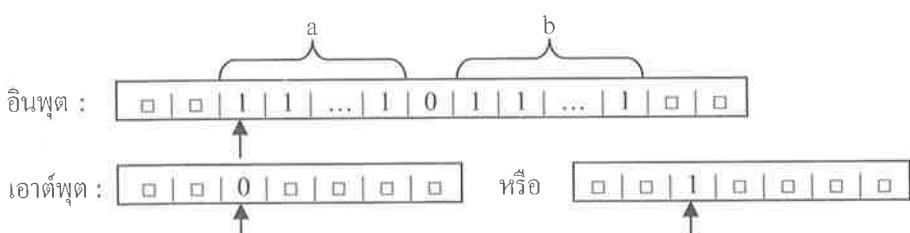
รูปที่ 8.18 ทัวริ่งแมชีนสำหรับฟังก์ชัน $f(w)$

ตัวอย่างที่ 8.9 กำหนดให้ฟังก์ชัน $f(a, b)$ ถูกนิยามดังนี้

$$f(a, b) = \begin{cases} 0 & \text{ถ้า } a > b \\ 1 & \text{ถ้า } a \leq b \end{cases}$$

โดยที่ a และ b คือ จำนวนเต็มในรูปแบบบูนารี จงแสดงแนวคิดเพื่อพิสูจน์ว่าฟังก์ชัน $f(a, b)$ สามารถคำนวณได้

วิธีทำ : พิสูจน์ว่าฟังก์ชัน $f(a, b)$ สามารถคำนวณได้ โดยการสร้างทัวริ่งแมชีนรองรับฟังก์ชันดังกล่าว (ในที่นี้จะแสดงเพียงแนวความคิดในการสร้างเท่านั้น) ทัวริ่งแมชีนจะต้องทำงานรองรับอินพุตและเอาต์พุตตามตัวอย่างดังนี้



รูปที่ 8.19 อินพุตและเอาต์พุตของทัวริ่งแมชีนสำหรับฟังก์ชัน $f(a, b)$

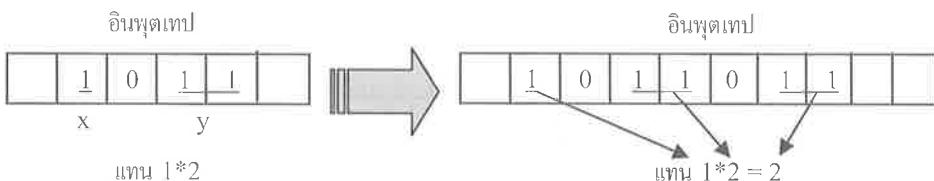
แนวคิด :

- จับคู่เลข 1 จากสตริง a กับเลข 1 จากสตริง b จนกระทั่งเลข 1 ของสตริง a หรือ b ถูกจับคู่จนหมด
- ถ้ายังคงมีเลข 1 เหลืออยู่ภายในสตริง a ให้ลบตัวอักษรทุกตัวภายในเทปแล้วเขียนเลข 0 ลงไปหนึ่งตัว (แทนกรณี $a > b$)
- หรือให้ลบตัวอักษรทุกตัวภายในเทปแล้วเขียนเลข 1 ลงไปหนึ่งตัว (แทนกรณี $a \leq b$)

จากแนวความคิดดังกล่าว แสดงให้เห็นว่าเราสามารถสร้างทัวริنجแมชีนซึ่งรองรับการคำนวณฟังก์ชัน $f(a, b)$ ได้ ดังนั้นฟังก์ชัน $f(a, b)$ จึงสามารถคำนวณได้

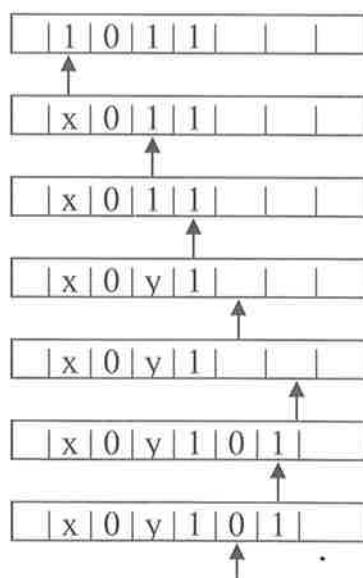
ตัวอย่างที่ 8.10 จงออกแบบทัวริنجแมชีนที่คูณตัวเลขจำนวนเต็มบวกสองจำนวน (x, y) ในรูปแบบของยูนารี (Unary Notation)

วิธีทำ : การคูณในรูปแบบของยูนารีสามารถทำได้โดยใช้การคัดลอก สมมุติว่าอินพุตที่จะถูกป้อนเข้ามา และเอาต์พุตมีลักษณะดังรูป (เรายังใช้เลขศูนย์เป็นตัวค่าน้ำหน่วงจำนวน)

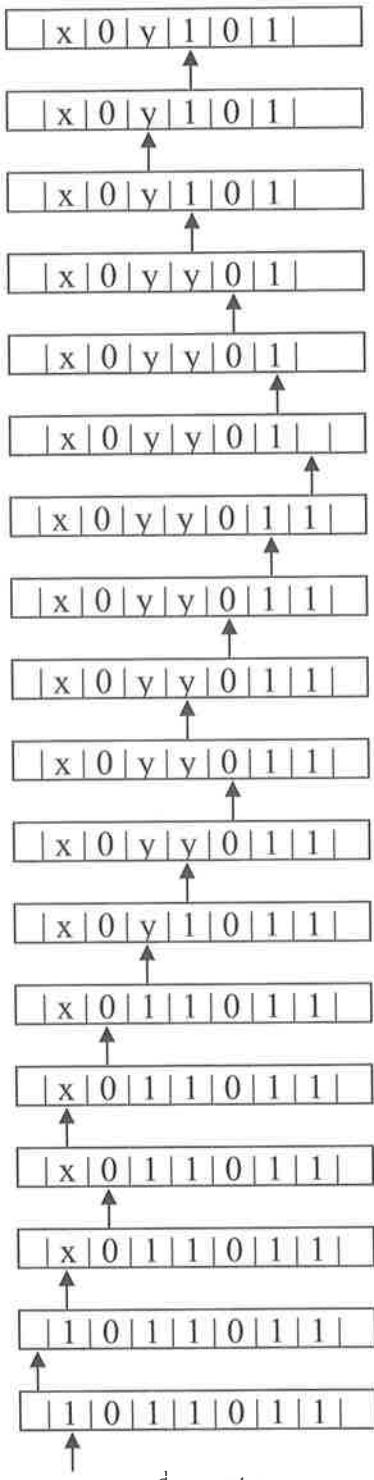


รูปที่ 8.20 อินพุตและเอาต์พุตของทัวริنجแมชีนสำหรับการคูณเลขยูนารี

การคูณดังกล่าวสามารถแทนได้ด้วยการคัดลอก y เป็นจำนวนครึ่งเท่ากับจำนวนของเลข 1 ที่ปรากฏอยู่ใน x ดังนั้นสามารถจำลองการทำงานของการคูณภายในทัวริنجแมชีนได้ดังนี้



รูปที่ 8.21 ทัวริنجแมชีนสำหรับการคูณเลขยูนารี



ចុច 8.21 (ពេល)

แม้จะเป็นเพียงแนวคิดจำลองการทำงานอย่างง่าย ๆ โดยไม่ได้แสดงการเปลี่ยนแปลงสถานะภายในหน่วยประมวลผลของทั่วไป แต่ผู้อ่านจะสามารถเข้าใจได้อย่างง่ายดายว่าทั่วไปแม้ชีนสามารถดำเนินการได้อย่างไร

8.5 สมมุติฐานของทั่วไป

จากเนื้อหาที่ผ่านมาเกี่ยวกับทั่วไปแม้ชีน สามารถประยุกต์ใช้งานทั่วไปแม้ชีนกับปัญหาต่าง ๆ มากมาย เช่น บัญชีง่าย ๆ ที่สามารถแก้ไขโดยไฟฟ้าในต่ออโตมาตาหรือพุชดาวน์อโตมาตาไปจนถึงปัญหาที่มีความซับซ้อนมาก ๆ ดังนี้ จึงดูเหมือนว่าทั่วไปแม้ชีนมีความสามารถเหลืออโตมาตแบบอื่น ๆ ที่ผ่านมาจนกระทั่งสามารถใช้แก้ปัญหา การคำนวณได้ ก็ได้ อย่างไรก็ตาม การจะสรุปเช่นนี้ได้นั้นไม่ใช่เรื่องง่าย เนื่องจากเราต้องสร้างขั้นตอนการพิสูจน์ ที่ละเอียดและครอบคลุมในทุก ๆ ปัญหาที่สามารถคำนวณได้ วิธีการนี้ที่อาจใช้เป็นข้อพิสูจน์ได้ก็คือ การพิสูจน์ ว่าคำสั่งในชุดคำสั่งของภาษาคอมพิวเตอร์ เช่น ภาษาซี, ภาษาปาล์ม หรือทั่วไปแม้ชีนรองรับการทำงาน แม้ว่าวิธีการพิสูจน์นี้จะมีความเป็นไปได้ แต่การสร้างทั่วไปแม้ชีนสำหรับทุก ๆ คำสั่งให้ครบถ้วนนั้นไม่ใช่เรื่องที่จะทำได้ในเวลาที่จำกัด

ในช่วงกลางทศวรรษที่ 1930 เกิดสมมุติฐานเกี่ยวกับทั่วไปแม้ชีน (Turing Thesis) ขึ้นว่า “การคำนวณ ได้ ที่สามารถแทนให้อยู่ในรูปแบบของเครื่องจักรใด สามารถแทนได้ด้วยการทำงานของทั่วไปแม้ชีนเท่านั้นเดียว” สมมุติฐานนี้ยังเป็นที่ต้องยังคงตั้งแต่อดีตมาจนถึงปัจจุบัน เนื่องจากไม่มีใครสามารถพิสูจน์สมมุติฐานของทั่วไปได้ และ ยังไม่มีใครสร้างตัวอย่างขึ้นมาขัดแย้งกับสมมุติฐานของทั่วไปได้เช่นกัน นอกจากนี้จากทั่วไปแม้ชีนแล้ว ยังมีการคิดค้น โมเดลในการคำนวณอื่น ๆ ขึ้นอีกมากมาย แต่ก็ไม่มีโมเดลใดที่มีประสิทธิภาพการทำงานเทียบเท่ากับทั่วไปแม้ชีน ซึ่งข้อเท็จจริงนี้อาจเป็นเหตุผลที่ทำให้นักวิทยาศาสตร์คอมพิวเตอร์ในยุคปัจจุบันยังคงเชื่อมั่นในสมมุติฐานของทั่วไป

หากต้องการยอมรับสมมุติฐานของทั่วไปจะต้องให้นิยามคำว่า “อัลกอริทึม” ให้สอดคล้องกับการทำงาน ของทั่วไปแม้ชีนดังนี้

นิยาม 8.5.1 : อัลกอริทึมสำหรับฟังก์ชัน f ซึ่งเป็นความสัมพันธ์ที่เกิดจากโดเมน D ไปยังช่วงของ ผลลัพธ์ R คือ ทั่วไปแม้ชีน M ซึ่งรับอินพุตสติง s ที่อยู่บนเทพ โดยที่ $s \in D$ และทำให้ทั่วไปแม้ชีนทำงานไปจน หยุดการทำงานที่สถานะสุดท้าย q_f พร้อมทั้งให้ผลลัพธ์ออกมาเป็น $f(s)$ อยู่บนเทพ โดยที่ $f(s) \in R$

$$\begin{array}{l} q_0 s \vdash \dots \vdash q_f f(s) \\ \text{นั่นคือ } q_0 s \vdash^* q_f f(s) \quad \text{โดยที่ } s \in D \text{ และ } f(s) \in R \end{array}$$

8.6 ทั่วไปแม้ชีนกับการแปลง

นอกเหนือจากทั่วไปแม้ชีนมาตรฐานที่ได้ศึกษามาแล้ว ปัจจุบันยังมีการตัดแปลงทั่วไปแม้ชีนให้มีลักษณะที่แตกต่างไปจากเดิมเพื่ออำนวยความสะดวกในการทำงาน เช่น

- ทัวริงแมชีนที่เลือกหยุดหัวอ่านอยู่กับที่ได้
- ทัวริงแมชีนที่มีเทปซึ่งปลายเทปข้างซ้ายถูกจำกัดความยาว
- ทัวริงแมชีนที่ใช้เทปที่มีหลาย ๆ ชั้น
- ทัวริงแมชีนแบบอฟไลน์ (Off-line)
- ทัวริงแมชีนที่ใช้หลายเทป
- ทัวริงแมชีนที่ใช้เทปแบบสองมิติ

แม้ว่าทัวริงแมชีนที่ถูกตัดแปลงนี้จะมีล่วงประคอนเพิ่มขึ้นจากเดิม แต่เป็นที่น่าสนใจว่าความสามารถของทัวริงแมชีนเหล่านี้ไม่ได้เหนือไปกว่าทัวริงแมชีนแบบมาตรฐานเลย ขอได้เปรียบของทัวริงแมชีนที่ถูกตัดแปลงเหล่านี้ (ยกเว้นทัวริงแมชีนที่มีเทปซึ่งปลายเทปข้างซ้ายถูกจำกัดความยาวและทัวริงแมชีนแบบอฟไลน์) เมื่อเทียบกับทัวริงแมชีนแบบมาตรฐานก็คือการอำนวยความสะดวกในการทำงาน ซึ่งส่งผลให้เวลาที่ใช้ในการทำงานลดลงไปด้วย

เนื้อหาในรายละเอียดของหัวข้อนี้ถือว่าเป็นเนื้อหาขั้นสูงของทัวริงแมชีน ซึ่งจะได้ศึกษาในระดับที่สูงกว่าปริญญาตรี ดังนั้นหนังสือเล่มนี้จึงไม่ได้กล่าวรายละเอียดในเชิงลึก ผู้อ่านสามารถศึกษาเพิ่มเติมได้จากตำราของต่างประเทศในบรรณานุกรมที่ระบุไว้ในตอนท้ายของหนังสือเล่มนี้

แบบฝึกหัด

1. ทั่วไปแมชีนมีความแตกต่างจากพุทธดาวน์อโถมาตาอย่างไรบ้าง
สำหรับข้อ 2 - 6 จะสร้างหรือให้แนวคิดในการสร้างทั่วไปแมชีนสำหรับรองรับภาษาดังต่อไปนี้
 2. $L = \{a^n b^m \mid n \geq m \text{ และ } m, n \neq 0\}$
 3. $L = \{ww^R w \mid w \in \{a, b\}^+\}$
 4. $L = \{w \in \{0, 1, 2\}^+ \mid n_0(w) = n_1(w) = n_2(w)\}$
 5. $L =$ เซตของสตริงปีกภาษาในภาษา C ซึ่งมีจำนวนของปีกภาษาเปิดและปีกภาษิดเท่ากัน เช่น $\{\{\}\}, \{\{\}\{\{\}\}\}$, เป็นต้น
 6. $L = \{a^n b^{2n} c^{3n} \mid n \geq 1\}$
สำหรับข้อ 7 - 10 จะออกแบบหรือให้แนวคิดในการสร้างทั่วไปแมชีนสำหรับคำนวนฟังก์ชันดังต่อไปนี้
 7. $f(n) = 3n$
 8. $f(n) = n + 3$
 9. $f(n) = \begin{cases} 1 & \text{ถ้า } n \text{ เป็นจำนวนเท่าของ } 3 \text{ เช่น } 0, 3, 6, 9, \dots \\ 0 & \text{ถ้า } n \text{ ไม่เป็นจำนวนเท่าของ } 3 \text{ เช่น } 1, 2, 4, 5, 7, \dots \end{cases}$
 10. $f(n) = \max(f_1(n), f_2(n))$

ເນື້ອທາຂັ້ນສູງຂອງ ທດປະກົງກາრຄໍານວນ

ສໍາຫລັບເນື້ອທາທີ່ຈະກ່າວລືງໃນນັ້ນທີ່ເປັນເນື້ອທາຂັ້ນສູງຂອງວິชาທາຄຸມຄືການຄໍານວນ ຜຶ່ງນັກຈະບຽນຮູ້ໄວ້ໃນຫລັກສູງຕະຫຼາດ ທີ່ສູງກວ່າຮະດັບປະໂຮງຢາຕົວ ດັ່ງນັ້ນກາຮອບຍາຍຫ້ວໜ້ອຕ່າງ ຖ້າໃນນັ້ນຈະເປັນເພີ່ມການສຽງສາມາດດ້ວຍຫຼັງເພື່ອເປັນພື້ນຖານກາຮືອມໃນຮະດັບທີ່ສູງເກີນໄປ ຫ້ວໜ້ອທີ່ຄ່ອບຄຸມໃນນັ້ນເກີ່ວຂ້ອງກັນການແລະໄວຍາກຮົນທີ່ຮ່ອງນັ້ນໂດຍຫ້ວົງແນ່ນເຊີນ ປໍ່ມີການຕັດສິນໃຈທີ່ແກ້ໄຂແລະແກ້ໄມໄດ້ ປໍ່ມີການຕັດສິນໃຈທີ່ແກ້ໄຂແລະແກ້ໄມໄດ້ ປໍ່ມີການຕັດສິນໃຈທີ່ແກ້ໄຂແລະແກ້ໄມໄດ້ ແລະປໍ່ມີການຕັດສິນໃຈທີ່ແກ້ໄຂແລະແກ້ໄມໄດ້

9.1 ການເວີຍແບັນເກີດແລະການເວີຍແບັນເກີດທີ່ນັບໄດ້

ນິຍາມ 9.1.1 : ຈະເຮີຍການ L ວ່າ ການເວີຍແບັນເກີດທີ່ນັບໄດ້ (Recursively Enumerable Language) ຄໍາມີຫ້ວົງແນ່ນທີ່ຮ່ອງຮັບການ L

ນິຍາມນີ້ຮັບນຸ້ມີເພີ່ມວ່າການ L ຈະເປັນການເວີຍແບັນເກີດໄດ້ເມື່ອມີຫ້ວົງແນ່ນທີ່ຮ່ອງຮັບການ L ໄດ້ ນັ້ນຄືອ້າຫຼວງແນ່ນຮັບສຕຣີໃດ ທີ່ອຸ່ນໃນການ L ແລະເປີ່ຍືນສຳຄັນຈາກສານະເຮັນຕົ້ນໄປຢັງສານະສຸດທ້າຍ ສິ່ງທີ່ສັງເກດໄດ້ຈາກນິຍາມນີ້ຄື່ອ້າຫຼວງແນ່ນທີ່ມີໝູ້ໃນການ L ຫ້ວົງແນ່ນຈາກຈະຫຼຸດທ່ານໃນສານະທີ່ມີໃຊ້ສານະສຸດທ້າຍ ຢ້ອງຈະຫຼຸດທ່ານໃນສານະສຸດທ້າຍ ຮູ່ອາຈຈະໄມ້ຫຼຸດທ່ານໃນສານະສຸດທ້າຍ (ຕິດອຸ່ນໃນລູບທີ່ໄມ້ລື້ນສຸດ) ເລັກໄດ້

ນິຍາມ 9.1.2 : ເຮີຍການ L ວ່າ ການເວີຍແບັນເກີດ (Recursive Language) ຄໍາມີຫ້ວົງແນ່ນທີ່ຮ່ອງຮັບການ L ແລະຫ້ວົງແນ່ນທີ່ຫຼຸດທ່ານໃນທຸກ ຖ້າສຕຣີຂອງ Σ^+

ນິຍາມນີ້ດູ້ເໝືອນຈະຄ້າຍກັນນິຍາມຂອງການເວີຍແບັນເກີດທີ່ນັບໄດ້ ແຕ່ມີຄວາມແຕກຕ່າງກັນທຽບທີ່ຫ້ວົງແນ່ນຂອງການເວີຍແບັນເກີດຈະຕ້ອງຫຼຸດທ່ານຫລັງຈາກທີ່ຮັບສຕຣີທຸກ ຖ້າສຕຣີທີ່ເປັນສາມາດໃຊ້ໃນ Σ^+ ທັງນີ້ເພື່ອປຶ້ອງກັນປໍ່ມີການຕັດສິນໃຈທີ່ໄມ້ລື້ນສຸດທີ່ສາມາຮັດເກີດທີ່ໄດ້ກັບການເວີຍແບັນເກີດທີ່ນັບໄດ້

ຈາກທັງສອງນິຍາມເກີດທີ່ກັບການເວີຍແບັນເກີດທີ່ນັບໄດ້ແລະການເວີຍແບັນເກີດ ສຽງໄດ້ວ່າການເວີຍແບັນເກີດທຸກການເປັນການເວີຍແບັນເກີດທີ່ນັບໄດ້ ແຕ່ໃນທາງທຽບກັນ ການເວີຍແບັນເກີດທີ່ນັບໄດ້ຈະໄວ້ໃຊ້ການເວີຍແບັນເກີດ ດັ່ງນັ້ນ ການເວີຍແບັນເກີດຈຶ່ງເປັນເຫັນຍ່ອງຂອງການເວີຍແບັນເກີດທີ່ນັບໄດ້

ນອກຈາກນີ້ ການເວີຍແບັນເກີດທີ່ນັບໄດ້ໄມ້ໃຊ້ເຫັນຍ່ອງການທີ່ມີຂາດໃຫຍ່ທີ່ສຸດ ເນື່ອຈາກຍັງມີການບາງຄຸມທີ່ອຸ່ນອາກເຫັນຈາກເຫັນນີ້ ເຫຼຸດຜົດຄືການເວີຍແບັນເກີດທີ່ນັບໄດ້ໄຈ້ຈັບຄູ່ກັບຈຳນວນນັ້ນ $\{1, 2, 3, 4, \dots\}$ ໄດ້ແນບທີ່ນັ້ນ ຕ່ອ້ອນນີ້ ທີ່ສັງເກດຈຶ່ງເປັນເຫັນຍ່ອງການທີ່ມີໃຊ້ສາມາດໃຊ້ໃນເຫັນຍ່ອງການເວີຍແບັນເກີດທີ່ນັບໄດ້ໂດຍຄ້າຍຂ້ອໂຕແຍ້ງໃນແນວ

ทแยงมุม (Diagonalization Argument) และจะมีผลทำให้ภาษาที่ถูกสร้างขึ้นมาใหม่ไม่เป็นสมาชิกในเซตของภาษา เวียนบังเกิดที่นั่นได้ ข้อโต้แย้งในแนวทแยงมุมมีอยู่ว่า

กำหนดให้เซตของตัวอักษร Σ เป็นเซตที่ไม่จำกัดแต่สามารถนับได้ เมื่อนำ Σ มาทำซ้ำตั้งแต่ 0 ครั้งขึ้นไป (Σ^*) จะได้เซตของสตริงทั้งหมดที่ถูกสร้างขึ้นจาก Σ ดังนี้

$$\Sigma^* = \{s_1, s_2, s_3, \dots\}$$

เนื่องจากภาษาเป็นเซตย่อของ Σ^* ดังนั้นจำนวนภาษาทั้งหมดที่เกิดจากเซตของตัวอักษร Σ คือ 2^N โดยที่ $N = |\Sigma|$ เพื่อพิสูจน์ว่าภาษาทั้งหมดนี้มีเพียงล่วงหนึ่งเท่านั้นที่เป็นภาษาเวียนบังเกิดที่นั่นได้ จะสมมุติเพื่อให้เกิดการขัดแย้งว่า “เซตของภาษาทั้งหมด 2^N เป็นเซตที่นับได้” จากนั้นเรานำภาษาที่อยู่ใน 2^N มาขับคู่กันจำนวนนับแบบหนึ่งต่อหนึ่ง เช่น ภาษา l_1 จับคู่กับเลข 1, ภาษา l_2 จับคู่กับเลข 2 เป็นต้น จากนั้นสำหรับสตริง $s_i \in \Sigma^*$ หาก $s_i \in l_j$ ให้ใส่เลข 1 ไว้ ณ จุดที่ตัดกัน แต่ถ้า $s_i \notin l_j$ ให้ใส่เลข 0 ดังแสดงในตารางต่อไปนี้

	s_1	s_2	s_3	s_4	...
l_1	1	0	1	0	...
l_2	0	1	0	1	...
l_3	1	1	0	0	...
l_4	0	0	1	0	...
...

จากตาราง จะเห็นได้ว่าภาษา l_j (ซึ่งเป็นสมาชิกอยู่ใน 2^N) ถูกจับคู่กับจำนวนนับแบบหนึ่งต่อหนึ่งจนหมดทุกภาษา อย่างไรก็ตาม เราสามารถยกตัวอย่างภาษาหนึ่งซึ่งไม่สามารถจับคู่กับจำนวนนับได้ ๆ ได้ หรืออีกนัยหนึ่งคือภาษาที่จะยกตัวอย่างต่อไปนี้ไม่เท่ากับภาษา l_j ได้ ๆ ที่อยู่ในตารางนั่นเอง ภาษาหนึ่งคือ ภาษาที่ประกอบด้วยสมาชิกที่อยู่ในแนวเส้นทแยงมุมของตาราง โดยให้เปลี่ยนตัวเลขในแนวเส้นทแยงมุมกลับกันกับเลขเดิม เช่น จากตารางนี้จะได้ภาษาที่จะยกตัวอย่างเป็น 0011... (จากเดิมคือ 1100...) ซึ่งคือภาษาที่ประกอบด้วยสตริง s_3 และ s_4 และสตริงอื่น ๆ ที่เหลือไว้ แต่ไม่มี s_1 และ s_2 เป็นสมาชิกอยู่ หากเราสังเกตดูดี ๆ แล้ว ภาษานี้ไม่ซ้ำกับภาษา l_j ได้ ๆ ที่อยู่ในตารางเลย ดังนั้นจึงขัดแย้งกับสมมุติฐานที่บอกว่า “เซตของภาษาทั้งหมด 2^N เป็นเซตที่นับได้” สิ่งที่ทำให้เราได้ข้อสรุปที่ผิดพลาดก็คือสมมุติฐานที่ไม่ถูกต้องนั่นเอง จึงสรุปได้ว่าเซตของภาษาทั้งหมดที่เกิดขึ้นจากเซตของตัวอักษร Σ นั้นเป็นเซตที่นับไม่ได้ ซึ่งส่งผลให้เซตของภาษาล่วงหนึ่งใน 2^N ไม่เป็นภาษาเวียนบังเกิดที่นั่นได้

การพิสูจน์โดยใช้ข้อโต้แย้งในแนวทแยงมุมนี้เป็นวิธีการพิสูจน์ที่คิดค้นในช่วงศตวรรษที่ 19 โดยนักคณิตศาสตร์ชื่อ จอร์จ แคนเตอร์ (George Cantor) ซึ่งต่อมาได้ใช้กันอย่างแพร่หลายในการพิสูจน์ทฤษฎีทางคณิตศาสตร์ โดยเฉพาะการพิสูจน์เกี่ยวกับการนับได้และนับไม่ได้

ตัวอย่างที่ 9.1 จงพิสูจน์ว่าเซตของทุกภาษาที่ไม่ใช่ภาษาเวียนบังเกิดที่นับได้เป็นเซตที่นับไม่ได้

พิสูจน์ : เนื่องจากเซตของภาษาเวียนบังเกิดที่นับได้เป็นเซตที่นับได้ (เพราะมีทั่วเรียงแมชีนรองรับ) แต่เซตของภาษาทั้งหมดเป็นเซตที่นับไม่ได้ ดังนั้นผลต่างของเซตที่นับไม่ได้กับเซตที่นับได้จึงเป็นเซตที่นับไม่ได้

ตัวอย่างที่ 9.2 จงพิสูจน์ว่าเซตของภาษาเวียนบังเกิดที่นับได้มีคุณสมบัติปิดภายใต้การยุเนียน

พิสูจน์ : กำหนดให้ภาษาเวียนบังเกิดที่นับได้ L_1 และ L_2 มีทัวริงแมชีน M_1 และ M_2 รองรับตามลำดับ สำหรับภาษาที่เกิดจากการยุเนียนภาษา L_1 และ L_2 สามารถสร้างทัวริงแมชีนขึ้นมารองรับได้ เช่นกัน โดยสร้างสถาณะเริ่มต้นใหม่พร้อมกับสร้างทรานซิชันที่ทำให้สามารถเลือกได้ว่าจะเข้าไปทำงานที่ M_1 หรือ M_2 ก็ได้ ดังนี้ ทัวริงแมชีนใหม่ที่สร้างขึ้นมาสามารถรองรับผลของการยุเนียนภาษา L_1 และ L_2 จึงเป็นการพิสูจน์คุณสมบัติปิดภายใต้การยุเนียนของภาษาเวียนบังเกิดที่นับได้

9.2 ไวยากรณ์ไม่จำกัดและไวยากรณ์ค่อนเท็กซ์เชนสิทิฟ

จากนิยามของไวยากรณ์สำหรับภาษาแต่ละชนิดในบทที่ผ่านมา มักจะถูกจำกัดในเรื่องลักษณะของโปรดักชันภาษาในไวยากรณ์ หากไม่มีข้อจำกัดใด ๆ กับโปรดักชันภาษาในไวยากรณ์ เราจะได้พบไวยากรณ์ใหม่ที่สามารถสร้างภาษาที่มีความซับซ้อนขึ้นอีกด้วย

นิยาม 9.2.1 : ไวยากรณ์ $G = (V, T, S, P)$ จะถูกเรียกว่า ไวยากรณ์ไม่จำกัด (Unrestricted Grammar) ถ้าโปรดักชันของไวยากรณ์มีรูปแบบดังนี้

$$x \rightarrow y$$

โดยที่ $x \in (V \cup T)^+$ และ $y \in (V \cup T)^*$

ไวยากรณ์ไม่จำกัดเมิดโอกาสให้โปรดักชันมีอิสระในการสร้างสตริงโดยไม่มีข้อจำกัดใด ๆ ซึ่งทำให้เซตของภาษาที่ถูกสร้างจากไวยากรณ์ไม่จำกัดเป็นเซตที่มีขนาดใหญ่มาก และยังมีความสัมพันธ์อย่างแน่นแฟ้นกับภาษาเวียนบังเกิดที่นับได้ซึ่งกำลังจะกล่าวถึงในทฤษฎีต่อไป

ทฤษฎี 9.2.1 : ภาษาที่ถูกสร้างขึ้นจากไวยากรณ์ไม่จำกัดเป็นภาษาเวียนบังเกิดที่นับได้

พิสูจน์ : สร้างสตริงจากภาษาของไวยากรณ์ไม่จำกัด โดยให้มีความยาวของสตริงเพิ่มขึ้นทีละ 1 ตัวอักษร (ทำให้ภาษาไม่สามารถนับได้) จากนั้นจำลองการทำงานของทัวริงแมชีนสำหรับแต่ละสตริงที่สร้างขึ้น ซึ่งแน่นอนว่า ทัวริงแมชีนจะรองรับสตริงทั้งหมดที่ถูกสร้างขึ้นในที่สุด และทำให้ภาษาจากไวยากรณ์ไม่จำกัดเป็นภาษาเวียนบังเกิดที่นับได้

นิยาม 9.2.2 : จะเรียกไวยากรณ์ $G = (V, T, S, P)$ ว่า ไวยากรณ์ค่อนเท็กซ์เชนสิทิฟ (Context-Sensitive Grammar) ถ้าโปรดักชันของไวยากรณ์มีรูปแบบดังนี้

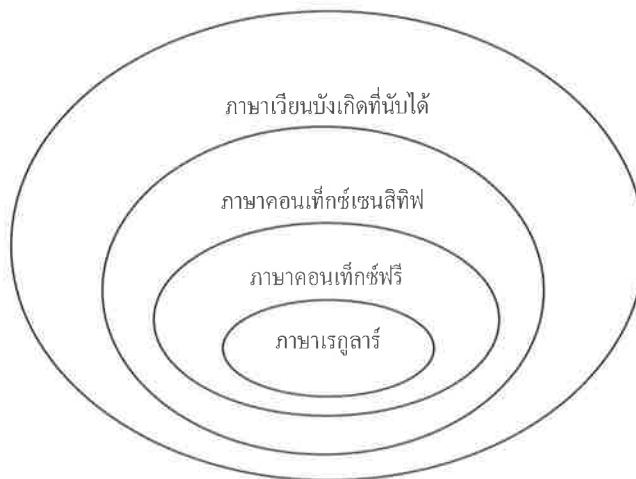
$$x \rightarrow y$$

โดยที่ x และ $y \in (V \cup T)^+$ และขนาดของ y จะต้องมากกว่าขนาดของ x เสมอ

นิยาม 9.2.3 : ภาษาใด ๆ จะถูกเรียกว่า ภาษาค่อนเท็กซ์เชนสิทิฟ ถ้ามีไวยากรณ์ค่อนเท็กซ์เชนสิทิฟรองรับภาษานั้น

ภาษาค่อนเทิกซ์เซนลิทิฟทุกภาษาจะมีทั่วไปแม้ชื่อแบบพิเศษรองรับเสมอ เราเรียกทั่วไปแม้ชื่อแบบพิเศษนี้ว่า ออโตมาตาแบบลิเนียร์บาร์ด (Linear Bounded Automata) ซึ่งมีข้อจำกัดที่ขอบเขตของอินพุตเทป โดยทั่วไปแม้ชื่อจะเขียนตัวอักษรลงไปในเทปได้เฉพาะภายในขอบเขตของอินพุตสตริงที่ถูกป้อนเข้ามาเท่านั้น นอกจากนี้ ภาษาค่อนเทิกซ์เซนลิทิฟทุกภาษาบางเป็นภาษาเวียนบังเกิดอีกด้วย

จากภาษาทั้งหมดที่ได้ศึกษามาตั้งแต่ต้น คือ ภาษาเรกูลาร์, ภาษาค่อนเทิกซ์ฟรี, ภาษาค่อนเทิกซ์เซนลิทิฟ, ภาษาเวียนบังเกิดที่นับได้ และภาษาเวียนบังเกิด เมื่อนำมาจัดลำดับขั้นของภาษาตามวิธีของ โนม ชอมสกี (Noam Chomsky) จะมีลักษณะดังรูป



รูปที่ 9.1 ลำดับขั้นของชอมสกี

รูปลำดับขั้นนี้เป็นลำดับขั้นดังเดิมของชอมสกี ซึ่งจะสังเกตได้ว่าไม่ได้แสดงความล้มเหลวของภาษาเวียนบังเกิดเอาไว้ สำหรับตำแหน่งของภาษาเวียนบังเกิดภายในลำดับขั้นของชอมสกี จะอยู่ระหว่างภาษาเวียนบังเกิดที่นับได้กับภาษาค่อนเทิกซ์เซนลิทิฟนั่นเอง

ตัวอย่างที่ 9.3 จงแสดงภาษาที่ไวยากรณ์ค่อนเทิกซ์เซนลิทิฟรองรับ

$$S \rightarrow XYZS_1 | XYZ, \quad S_1 \rightarrow XYZS_1 | XYZ,$$

$$YX \rightarrow XY, \quad ZX \rightarrow XZ, \quad ZY \rightarrow YZ,$$

$$X \rightarrow x, \quad xX \rightarrow xx, \quad xY \rightarrow xy,$$

$$yY \rightarrow yy, \quad yZ \rightarrow yc, \quad zZ \rightarrow zz$$

ภาษาที่ไวยากรณ์ค่อนเทิกซ์เซนลิทิฟรองรับคือ $\{x^n y^n z^n : n \geq 1\}$

9.3 ปัญหาการตัดสินใจที่แก้ได้และแก้ไม่ได้

จากสมมุติฐานของทั่วเริงที่ผ่านมาในบทที่ 8 อาจดูเหมือนว่าทั่วเริงแมชีนมีความสามารถในการรองรับภาษาต่าง ๆ ได้อย่างมาก many อย่างไรก็ตาม ยังมีอีกหลายภาษาที่ไม่สามารถสร้างทั่วเริงแมชีนเพื่อมารองรับได้ ซึ่งการพิสูจน์ว่าทั่วเริงแมชีนใด ๆ ไม่สามารถแก้ปัญหาหนึ่งได้นั้น จะต้องอาศัยการพิสูจน์ที่มีเหตุผลที่เชื่อถือได้ และการพิสูจน์นั้น ต้องอยู่บนพื้นฐานของปัญหาการตัดสินใจที่แก้ได้และปัญหาการตัดสินใจที่แก้ไม่ได้ โดยจะเริ่มต้นศึกษาปัญหาการตัดสินใจที่แก้ได้ก่อนดังต่อไปนี้

1. ปัญหาการตัดสินใจที่แก้ได้ (Solvable, or Decidable, Decision Problem)

นิยาม 9.3.1 : ปัญหาซึ่งมีภาษาเป็นลักษณะเวียนบังเกิดเป็นปัญหาที่แก้ได้

นิยามนี้สามารถกล่าวอีกนัยหนึ่งได้ว่า เมื่อแทนปัญหาให้อยู่ในรูปแบบภาษาที่เหมาะสมแล้ว ภาษาันจะถูกเรียกว่า ภาษาที่แก้ได้ ถ้ามีอัลกอริทึม (หรือทั่วเริงแมชีน) ที่รับอินพุตสตริง และสามารถบอกได้ว่าอินพุตสตริงที่รับเข้ามาเป็นสมาชิกอยู่ในภาษาันหรือไม่ ซึ่งคำตอบจะต้องออกมายังรูปแบบ “ใช่” หรือ “ไม่ใช่” เท่านั้น

ปัญหาการตัดสินใจที่แก้ได้นั้นได้เป็นประเด็นที่นำสันใจมากันกันในตอนนี้ เมื่อจากนี้อยู่กับความสามารถในการสร้างทั่วเริงแมชีนของรับงานดังกล่าว แต่สิ่งที่ได้รับความสนใจเป็นพิเศษในศาสตร์ของทฤษฎีการคำนวณคือปัญหาการตัดสินใจที่แก้ไม่ได้ ซึ่งจะต้องใช้วิธีการพิสูจน์ที่นำสันใจและแตกต่างจากปัญหาการตัดสินใจที่แก้ได้

2. ปัญหาการตัดสินใจที่แก้ไม่ได้ (Unsolvable, or Undecidable, Decision Problem)

นิยาม 9.3.2 : ปัญหาซึ่งมีภาษาไม่เป็นลักษณะเวียนบังเกิดเป็นปัญหาที่แก้ไม่ได้

นั่นคือ ภาษาันจะถูกเรียกว่า ภาษาที่แก้ไม่ได้ ถ้าไม่มีทั่วเริงแมชีนใด ๆ ที่รับอินพุตสตริงแล้วสามารถบอกได้ว่าอินพุตสตริงที่รับเข้ามาเป็นสมาชิกอยู่ในภาษาันหรือไม่

การพิสูจน์ว่าปัญหาที่กำลังพิจารณาเป็นปัญหาที่แก้ไม่ได้ จะใช้วิธีการลดthonปัญหาพื้นฐานซึ่งเราทราบดีว่า เป็นปัญหาที่แก้ไม่ได้ให้อยู่ในรูปแบบของปัญหาที่เรากำลังพิจารณาอยู่ หากลดthonปัญหานั้นได้สำเร็จ ก็แสดงว่า ปัญหาที่กำลังพิจารณาอยู่นั้นเป็นปัญหาที่แก้ไม่ได้ ดังนั้นจึงควรศึกษาลึกลึกลับของปัญหาพื้นฐานก่อนเป็นอันดับแรก ปัญหาพื้นฐานที่ควรทราบว่าเป็นปัญหาที่แก้ไม่ได้คือปัญหาการหยุดของทั่วเริงแมชีน

ปัญหาการหยุดของทั่วเริงแมชีน

กำหนดให้สตริง W_M เป็นสตริงที่ถูกเข้ารหัสซึ่งประกอบด้วย 0 และ 1 เพื่อใช้อธิบายเซตของทรานซิชัน พังก์ชันของทั่วเริงแมชีน M เช่น ให้สถานะ q_1 ถูกแทนด้วย 1 และตัวอักษร a และ b ถูกแทนด้วย 1 และ 11 ตามลำดับ ส่วนการเคลื่อนที่หัวอ่านไปทางซ้ายถูกแทนด้วย 1 ดังนั้นทรานซิชัน $\delta(q_1, a) = (q_1, b, L)$ สามารถแทนได้ด้วยการเข้ารหัส 1010101101 โดยเราจะใช้ 0 เป็นตัวคั่นกลางระหว่างสัญลักษณ์ต่าง ๆ

นอกจากนี้ กำหนดให้สตริง w เป็นอินพุตสตริงที่จะถูกป้อนให้ทั่วเริงแมชีน M โดยสตริง w ถูกเข้ารหัสในลักษณะเดียวกันกับ W_M จากนั้นสร้างทั่วเริงแมชีน N ที่สามารถคำนวณดังนี้

- $q_0 W_M w \xrightarrow{*} u q_{f1} v$ ถ้าสตริง w ทำให้ทั่วเริงแมชีน M หยุดทำงาน
- $q_0 W_M w \xrightarrow{*} x q_{f2} y$ ถ้าสตริง w ไม่ทำให้ทั่วเริงแมชีน M หยุดทำงาน

โดยที่หัวสถานะ q_{f1} และ q_{f2} เป็นสถานะสุดท้ายของทั่วเริงแมชีน N

ทฤษฎี 9.3.1 : ปัญหาการหยุดของทัวริงแมชีนเป็นปัญหาที่แก้ไม่ได้ หรืออีกนัยหนึ่งคือ ไม่มีทัวริงแมชีนใด ๆ ที่สามารถแก้ปัญหาการหยุดของทัวริงแมชีนได้

- พิสูจน์ : พิสูจน์โดยการขัดแย้ง ซึ่งสมมุติฐานที่จะต้องเพื่อให้เกิดความขัดแย้งคือ สมมุติว่ามีทัวริงแมชีน H ที่สามารถแก้ปัญหาการหยุดของทัวริงแมชีนได้ โดยทัวริงแมชีน H จะต้องรับอินพุตสตริง $q_0 W_M w$ จากนั้นทัวริงแมชีน H จะคำนวณและหยุดทำงานที่สถานะ q_1 ถ้า w ทำให้ทัวริงแมชีน M หยุดทำงาน มิฉะนั้นทัวริงแมชีน H จะหยุดทำงานที่สถานะ q_{12} การทำงานของทัวริงแมชีน H สามารถแสดงได้โดยการอธิบาย ณ ช่วงขณะเวลาหนึ่ง (Instantaneous Description) ดังนี้

- $q_0 W_M w \vdash^*_H q_{12} v$ ถ้าสตริง w ทำให้ทัวริงแมชีน M หยุดทำงาน
- $q_0 W_M w \vdash^*_H x q_{12} y$ ถ้าสตริง w ไม่ทำให้ทัวริงแมชีน M หยุดทำงาน

ขั้นตอนต่อไป สร้างทัวริงแมชีน \bar{H} จากทัวริงแมชีน H โดยที่ทัวริงแมชีน \bar{H} จะไม่หยุดทำงานถ้าสตริง w ทำให้ทัวริงแมชีน M หยุดทำงาน สามารถทำได้โดยการบังคับให้เกิดลูปกลับไปกลับมาระหว่างสถานะ q_1 และสถานะพิเศษที่สร้างเพิ่มขึ้นโดยไม่มีการเคลื่อนหัวอ่าน (อ่านอินพุตเป็นสตริงว่าง) นอกจากนี้ทัวริงแมชีน M จะหยุดทำงานที่สถานะ q_{12} ถ้าสตริง w ไม่ทำให้ทัวริงแมชีน M หยุดทำงาน การทำงานของทัวริงแมชีน \bar{H} สามารถแสดงได้โดยการอธิบาย ณ ช่วงขณะเวลาหนึ่งดังนี้

- $q_0 W_M w \vdash^*_{\bar{H}} \text{Infinite_Loop}$ ถ้าสตริง w ทำให้ทัวริงแมชีน M หยุดทำงาน
- $q_0 W_M w \vdash^*_{\bar{H}} x q_{12} y$ ถ้าสตริง w ไม่ทำให้ทัวริงแมชีน M หยุดทำงาน

เนื่องจากทัวริงแมชีน \bar{H} สามารถรับอินพุตเป็นสตริงที่แทนทัวริงแมชีน M ได้ ๆ และอินพุตสตริง w ได้โดยทัวริงแมชีน \bar{H} จะไม่หยุดทำงานเมื่ออินพุตสตริง w ทำให้ทัวริงแมชีน M หยุดทำงาน และในทางตรงข้าม ทัวริงแมชีน \bar{H} จะหยุดทำงานเมื่ออินพุตสตริง w ไม่สามารถทำให้ทัวริงแมชีน M หยุดทำงาน ประเด็นสำคัญของการพิสูจน์นี้อยู่ตรงที่ว่า จะเกิดอะไรขึ้น ถ้าเราป้อนทัวริงแมชีน \bar{H} และอินพุตสตริงเข้าไปในทัวริงแมชีน \bar{H} สิ่งที่ผิดพลาดก็จะเกิดขึ้นดังนี้

- $q_0 W_{\bar{H}} w \vdash^*_{\bar{H}} \text{Infinite_Loop}$ ถ้าสตริง w ทำให้ทัวริงแมชีน \bar{H} หยุดทำงาน
- $q_0 W_{\bar{H}} w \vdash^*_{\bar{H}} x q_{12} y$ ถ้าสตริง w ไม่ทำให้ทัวริงแมชีน \bar{H} หยุดทำงาน

ความผิดพลาดที่เกิดขึ้นก็คือ “ทัวริงแมชีน \bar{H} จะไม่หยุดทำงาน ถ้าสตริง w ทำให้ทัวริงแมชีน \bar{H} หยุดทำงาน” ประโยคนี้มีความขัดแย้งในตัวเองและเป็นไปไม่ได้ (เบริญเลเมื่อประโภคที่ว่า “ฉันจะกินข้าว ถ้าฉันไม่กินข้าว” ซึ่งเป็นไปไม่ได้) ความผิดพลาดที่เกิดขึ้นนี้เนื่องจากสมมุติฐานที่ตั้งไว้ตอนต้นว่า “สมมุติให้มีทัวริงแมชีน H ที่สามารถแก้ปัญหาการหยุดของทัวริงแมชีนได้” ดังนั้นสมมุติฐานนี้เองผิดพลาดและทำให้สรุปได้ว่าไม่มีทัวริงแมชีน H ใด ๆ ที่สามารถแก้ปัญหาการหยุดของทัวริงแมชีนได้

สำหรับวิธีการพิสูจน์ปัญหาที่แก้ไม่ได้ของปัญหาอื่น ๆ จะใช้วิธีการลดthonปัญหาจากปัญหาการหยุดของทัวริงแมชีน (ซึ่งเราทราบแล้วว่าเป็นปัญหาที่แก้ไม่ได้) ให้ออกในรูปแบบเดียวกับปัญหาที่กำลังพิจารณา หากสามารถลดthonปัญหานี้ได้ ก็สามารถสรุปได้ว่าปัญหานี้เป็นปัญหาที่แก้ไม่ได้เช่นเดียวกับปัญหาการหยุดของทัวริงแมชีน

ตัวอย่างที่ 9.4 กำหนดให้ปัญหาการเข้าสู่สถานะถูกนิยามดังนี้ สำหรับทัวริงแมชีน M ใด ๆ ให้ตรวจสอบว่า เมื่อป้อนอินพุตสตริง w เข้าไปในทัวริงแมชีน M แล้วจะทำให้ M ทำงานผ่านสถานะ q จงพิสูจน์ว่าปัญหาการเข้าสู่สถานะเป็นปัญหาที่แก้ไม่ได้

พิสูจน์ : พิสูจน์โดยลดตอนปัญหาการหยุดของทัวริงแมชีนให้เข้ากับปัญหาการเข้าสู่สถานะ โดยสมมุติว่า สามารถถ่วงทัวริงแมชีน S ใน การ รองรับปัญหาการเข้าสู่สถานะได้ และมีการทำงานดังนี้

- $q_0 W_M w \vdash^*_S u q_{f1} v$ ถ้าสตริง w ทำให้ทัวริงแมชีน M ทำงานผ่านสถานะ q
- $q_0 W_M w \vdash^*_S x q_{f2} y$ ไม่ทำให้ทัวริงแมชีน M ทำงานผ่านสถานะ q

ขั้นตอนต่อไป จะต้องดัดแปลงทัวริงแมชีน M เพื่อเดิมแบบปัญหาการหยุดของทัวริงแมชีน โดยสามารถดัดแปลงการทำงานของทัวริงแมชีน M ทึ้งสองกรณีดังนี้

กรณีที่ 1 ถ้าสตริง w ทำให้ทัวริงแมชีน M ทำงานผ่านสถานะ q ได้ ก็จะทำให้ทัวริงแมชีน S หยุดทำงานที่สถานะ q_{f1} ดัดแปลงทรานซิชันของทัวริงแมชีน M โดยเปลี่ยนสถานะ q ให้เป็นสถานะสุดท้าย เพื่อให้ M หยุดทำงานเมื่อเข้าสู่สถานะ q

กรณีที่ 2 ถ้าสตริง w ไม่ทำให้ทัวริงแมชีน M ทำงานผ่านสถานะ q อาจเกิดขึ้นได้ 3 เงื่อนไข คือ เงื่อนไขที่ 1 ทัวริงแมชีน M เกิดการวนลูปภายในและไม่ผ่านสถานะ q ทำให้ทัวริงแมชีน S เปลี่ยนสถานะเข้าสู่สถานะ q_{f2} ซึ่งสามารถทำให้เกิดการวนลูประหว่างสถานะ q_{f2} กับสถานะใหม่ที่สร้างขึ้นเพิ่มเติมได้ (เป็นผลให้เงื่อนไขแรกของทัวริงแมชีน S มีพฤติกรรมเหมือนกับปัญหาการหยุดทำงานของทัวริงแมชีนเมื่อสตริง w ไม่สามารถทำให้ทัวริงแมชีนหยุดทำงานได้) เงื่อนไขที่ 2 ทัวริงแมชีน M ทำงานจนไปหยุดอยู่ที่สถานะสุดท้ายแต่ไม่ผ่านสถานะ q ซึ่งกรณีนี้สามารถดัดแปลงทัวริงแมชีน M ด้วยการเปลี่ยนสถานะสุดท้าย (ที่ไม่ใช่ q) ให้กลายเป็นสถานะที่ไม่ใช่สถานะสุดท้าย และสร้างทรานซิชันเพิ่มเติมเพื่อทำให้เกิดการวนลูปภายใน M โดยไม่ต้องอ่านอินพุตระหว่างสถานะสุดท้ายเดิมที่เพิ่งถูกเปลี่ยนไปและสถานะพิเศษที่ถูกสร้างขึ้นเพิ่มเติม (กรณีนี้จะมีพฤติกรรมเหมือนกับปัญหาการหยุดทำงานของทัวริงแมชีนเมื่อสตริง w ไม่อาจทำให้ทัวริงแมชีนหยุดทำงานได้) เงื่อนไขที่ 3 ทัวริงแมชีน M ทำงานไม่ผ่านสถานะ q และปฏิเสธสตริง w ด้วยการหยุดทำงาน สำหรับเงื่อนไขนี้ทัวริงแมชีน M จะหยุดทำงานเนื่องจากไม่มีทรานซิชันถูกกำหนดไว้สำหรับตัวอักษรภายในอินพุตสตริง ดัดแปลงทัวริงแมชีน M โดยการสร้างทรานซิชันสำหรับตัวอักษรที่ไม่ได้กำหนดไว้ โดยทำให้เกิดลูปภายในทัวริงแมชีนเหมือนกับที่ดัดแปลงในเงื่อนไขที่ 1 และ 2

จากการดัดแปลงทัวริงแมชีน M ในทุก ๆ กรณี เป็นที่ชัดเจนว่าทัวริงแมชีน M ก็คือทัวริงแมชีนที่หยุดทำงานเมื่อสตริง w สามารถทำให้เกิดการทำงานผ่านสถานะ q ได้ และจะไม่หยุดทำงานเมื่อสตริง w ไม่สามารถทำให้เกิดการทำงานผ่านสถานะ q ได้ ซึ่งคล้ายคลึงกับปัญหาการหยุดทำงานของทัวริงแมชีน ดังนั้นทัวริงแมชีน S จึงสามารถใช้แก้ปัญหาการหยุดการทำงานของทัวริงแมชีนได้ด้วย อย่างไรก็ตาม เรายารบماก่อนหน้านี้แล้วว่าปัญหาการหยุดการทำงานของทัวริงแมชีนเป็นปัญหาที่แก้ไม่ได้ (ไม่มีทัวริงแมชีนที่สามารถแก้ปัญหาได้) ดังนั้นจึงขัดแย้งกับลิ่งที่สรุปได้ สมมุติฐานที่ตั้งไว้ตอนต้นน่าว่า “สามารถสร้างทัวริงแมชีน S ใน การ รองรับปัญหาการเข้าสู่สถานะได้” จึงไม่มีทางเป็นไปได้ เราจึงพิสูจน์ได้ว่าปัญหาการเข้าสู่สถานะเป็นปัญหาที่แก้ไม่ได้

ตัวอย่างที่ 9.5 ปัญหาการหยุดเมื่อเทปว่างคือปัญหาที่ต้องการทราบว่าทัวริงแมชีน M จะหยุดการทำงานหรือไม่หากเริ่มต้นการทำงานด้วยเทปที่ว่างเปล่า งพิสูจน์ว่าปัญหานี้เป็นปัญหาที่แก้ไม่ได้

พิสูจน์ : สมมุติเพื่อให้เกิดการขัดแย้งว่าสามารถสร้างทัวริงแมชีน B เพื่อรับปัญหาการหยุดเมื่อเทปว่างได้ จากนั้นเริ่มต้นสร้างทัวริงแมชีน M' จากทัวริงแมชีน M โดย M' เริ่มต้นจากเทปที่ว่างเปล่า จากนั้น M' เขียนสตริง w ลงในปุ่มพิมพ์เทปและเลื่อนหัวอ่านมาอยู่ที่ตำแหน่งตัวอักษรตัวแรกของสตริง w พร้อมทั้งเปลี่ยนสถานะไปเป็นสถานะเริ่มต้น q_0 ดังแสดงด้วย $q_0 w$ หลังจากนี้ ให้ทัวริงแมชีน M' ทำงานเหมือนกับทัวริงแมชีน M ทุกประการ

จะเห็นได้ว่าทัวริงแมชีน M' จะหยุดทำงานก็ต่อเมื่อทัวริงแมชีน M หยุดทำงานโดยการป้อนอินพุตสตริง w ดังนั้นจึงมีพิสูจน์ว่า M' สามารถใช้ทัวริงแมชีน B มาแก้ปัญหาการหยุดการทำงานของทัวริงแมชีนได้ เช่นกัน แต่เนื่องจากทราบมาก่อนหน้านี้แล้วว่าปัญหาการหยุดการทำงานของทัวริงแมชีนเป็นปัญหาที่แก้ไม่ได้ (ไม่มีทัวริงแมชีนที่สามารถแก้ปัญหาได้) ดังนั้นจึงขัดแย้งกับลิ่งที่สรุป ทำให้สมมุติฐานที่ตั้งไว้ต้องถูกตัด除ว่า “สามารถสร้างทัวริงแมชีน B เพื่อรับปัญหาการหยุดเมื่อเทปว่างได้” ไม่มีทางเป็นไปได้ จึงพิสูจน์ได้ว่าปัญหาการหยุดเมื่อเทปว่างเป็นปัญหาที่แก้ไม่ได้

9.4 ปัญหาแทร็กเทเบิล อินแทร็กเทเบิล และปัญหาเอ็นพี-สมบูรณ์

เราอาจแบ่งประเภทของปัญหาตามลำดับขั้นของซอมลิกดังที่ได้กล่าวมาแล้วในหัวข้อก่อนหน้านี้ อย่างไรก็ตาม เราอาจสามารถแบ่งประเภทของปัญหาตามระดับความซับซ้อนในการทำงานของทัวริงแมชีนที่รองรับปัญหานั้นได้อีกด้วย ลิ่งแรกที่เราจะต้องทำความเข้าใจก็คือการทำความรู้จักกับประเภทของปัญหา DTIME และ NTIME ก่อน เนื่องจากปัญหาทั้งสองประเภทนี้จะถูกใช้ในการปัญหาที่เรากำลังจะศึกษาต่อไป

นิยาม 9.4.1 :

- ภาษา L จะถูกเรียกว่าเป็นสมาชิกของปัญหาประเภท DTIME($T(n)$) ถ้ามีทัวริงแมชีนแบบคาดเดาได้ (Deterministic Turing Machine) ยอมรับภาษา L ในเวลา $O(T(n))$
 - ภาษา L จะถูกเรียกว่าเป็นสมาชิกของปัญหาประเภท NTIME($T(n)$) ถ้ามีทัวริงแมชีนแบบคาดเดาไม่ได้ (Nondeterministic Turing Machine) ยอมรับภาษา L ในเวลา $O(T(n))$
- ความสัมพันธ์ระหว่างปัญหาประเภท DTIME และ NTIME คือ

$$\text{DTIME}(T(n)) \subseteq \text{NTIME}(T(n))$$

นอกเหนือจากนี้

$$\begin{array}{lll} \text{DTIME}(n^k) & \subset \text{DTIME}(n^{k+1}) & \text{สำหรับจำนวนเต็ม } k \geq 1 \\ \text{เช่น} & \text{DTIME}(n^2) & \subset \text{DTIME}(n^3) \end{array}$$

ตัวอย่างที่ 9.6 กำหนดให้ภาษา $L = \{ww : w \in \{a, b\}^*\}$ งพิสูจน์ว่าภาษา L เป็น NTIME(n)

สำหรับตัวอย่างนี้เป็นการพิสูจน์ว่าภาษา L เป็น NTIME(n) หรืออีกนัยหนึ่งคือ จะต้องสร้างทัวริงแมชีนแบบคาดเดาไม่ได้ (ในที่นี้จะใช้ทัวริงแมชีนแบบหลายเทป) เพื่อรองรับการทำงานของ L ภายในเวลา $O(n)$ แนวคิดในการสร้างทัวริงแมชีนดังกล่าวมีดังนี้

กำหนดให้อินพุตสตริงมีความยาวเท่ากับ n และถูกบรรจุอยู่ในเทป 1

- ขั้นตอนที่ 1 เดาถูกใจของสตริงนี้และคัดลอกสตริงครึ่งหลังลงไปยังเทป 2

- ขั้นตอนที่ 2 เมริยบเทียบสัญลักษณ์ครึ่งแรกบนเทป 1 กับสัญลักษณ์บนเทป 2 แบบหนึ่งต่อหนึ่งไปเรื่อยๆ

อัลกอริทึมที่นำเสนอนี้ใช้ประโยชน์จากการคาดเดาไม่ได้ของทัวริงแมชีนเพื่อเดาถูกใจของสตริงบนเทป 1 ซึ่งในแต่ละขั้นตอนจะใช้เวลาไม่เกิน $O(n)$ ดังนั้นจึงเป็นการพิสูจน์ได้ว่าสามารถสร้างทัวริงแมชีนแบบคาดเดาไม่ได้เพื่อรองรับการทำงานของ L ภายในเวลา $O(n)$ ซึ่งส่งผลให้ภาษา L เป็น NTIME(n)

จากประเภทของปัญหา DTIME และ NTIME จะนำมานิยามประเภทของความซับซ้อนที่เรารู้ดีกันดีในวิชาอัลกอริทึมนั้นคือประเภทของความซับซ้อน P และ NP ซึ่งถูกนิยามดังนี้

$$P = \bigcup_{i \geq 1} \text{DTIME}(n^i)$$

$$NP = \bigcup_{i \geq 1} \text{NTIME}(n^i)$$

ความหมายของ P คือ การยืนยันประเภทของปัญหา DTIME ที่มีระดับความซับซ้อนเป็นเวลาพหุนาม (Polynomial Time) เช่น n^1, n^2, n^3, \dots หรืออาจจะนิยามได้อีกอย่างหนึ่งว่า P คือ ประเภทของภาษาที่แก้ได้ (Decidable) ภายในเวลาพหุนามโดยใช้ทัวริงแมชีนที่คาดเดาได้แบบเทปเดียว

ส่วนความหมายของ NP (ย่อมาจาก Nondeterministic Polynomial Time) ก็จะมีลักษณะที่คล้ายคลึงกัน แต่แตกต่างกันตรงที่เราพิจารณาที่ NTIME ลำบากความสัมพันธ์ระหว่าง P และ NP คือ $P \subseteq NP$

ในทางทฤษฎี แม้จะแบ่งความซับซ้อนออกเป็น P และ NP แต่ในทางปฏิบัติ เราคลับให้ความสนใจที่เวลาที่ใช้ในการทำงานของอัลกอริทึมที่ไม่ควรจะนานจนเกินไป ซึ่งมีผลอย่างยิ่งที่จะทำให้อัลกอริทึมนั้นเป็นที่ยอมรับว่าสามารถใช้งานได้จริงในเชิงปฏิบัติ

นิยาม 9.4.2 : ปัญหาใด ๆ จะถูกเรียกว่า แทร็กเทเบิล (Tractable) ถ้าปัญหานั้นใช้เวลาในการแก้ปัญหาไม่นานจนเกินไป หรืออีกนัยหนึ่งคือใช้เวลาในการแก้ปัญหามิ่งเกินเวลาพหุนาม

หมายเหตุ : สำหรับนิยามของปัญหาอินแทริกเทเบิล (Intractable Problem) จะตรวจสอบข้างบนนี้ปัญหาแทร็กเทเบิล

ตัวอย่างที่ 9.7 พิจารณาปัญหา SAT (Satisfiability Problem) ซึ่งถูกนิยามดังนี้ กำหนดให้นิพจน์ e อยู่ในรูปแบบ CNF (Conjunctive Normal Form) ซึ่งเป็นการนำนิพจน์ย่อยที่เกิดจากการเชื่อมด้วยเครื่องหมายหรือ (OR) มาเชื่อมต่อกันโดยใช้เครื่องหมายและ (AND) เช่น

$$e = (x_1 \vee x_2) \wedge (x_3 \vee \overline{x_1})$$

คำถาวรในปัญหา SAT คือ ให้คันหาค่าบูลีน (Boolean) ที่เหมาะสมของตัวแปรต่าง ๆ ที่ทำให้นิพจน์ e มีค่าความจริงเป็นจริง จึงพิสูจน์ว่าปัญหา SAT เป็นปัญหาประเภท NP

พิสูจน์ : วิธีการพิสูจน์คือ จะต้องสร้างทั่วจริงแบบชีนแบบคาดเดาไม่ได้และใช้ทฤษฎีบทปัจจุบันแก้ปัญหา SAT และเวลาที่ใช้โดยทั่วจริงแบบชีนจะต้องเป็นเวลาพหุนาม

อัลกอริทึม

1. เข้ารหัสนิพจน์ e ให้อยู่ในรูปแบบที่เข้าใจได้โดยทั่วจริงแบบชีน โดยเข้ารหัสตัวห้อ (Subscript) ของตัวแปรแต่ละตัวเป็นเลขฐานสอง และแทนลัญลักษณ์คอมพลีเมนต์ด้วยเครื่องหมาย # ดังนี้ $e = (x_1 \vee x_{10}) \wedge (x_{11} \vee x \#1)$

เนื่องจากความยาวของนิพจน์ e ที่ถูกเข้ารหัสจะขึ้นอยู่กับความยาวของรหัสตัวห้อของตัวแปร ซึ่งถ้าเรามีตัวแปร n ตัว จะต้องใช้สตริงที่มีความยาวเท่ากับ $\log(n)$ ในการเข้ารหัสตัวห้อ ดังนั้นความยาวมากที่สุดของนิพจน์ e ที่ถูกเข้ารหัสคือ $n\log(n)$

2. ขั้นตอนต่อไปคือการเดาค่า (จริง/เท็จ) ของตัวแปรแต่ละตัว ซึ่งขั้นตอนนี้จะใช้เวลาอีก $O(n)$

3. นำค่าที่เดาขึ้นมาในขั้นตอนที่ 2 ไปแทนค่าลงในนิพจน์ e เพื่อตรวจสอบค่าความจริง ขั้นตอนนี้จะใช้เวลา $O(n^2\log(n))$

จะเห็นได้ว่า $O(n^2)$ เป็นเวลาที่เพียงพอที่ใช้ในการแก้ปัญหา SAT ดังนั้นสามารถสร้างทั่วจริงแบบชีนแบบคาดเดาไม่ได้ขึ้นมาแก้ปัญหา SAT โดยใช้เวลาภายใน $O(n^3)$ ซึ่งทำให้ปัญหา SAT เป็นสมาชิกในปัญหา NP

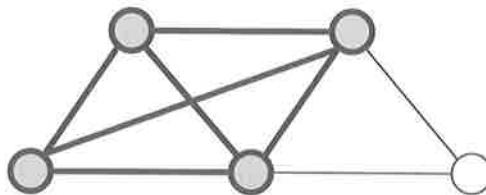
ในการที่จะบอกว่าภาษาใดเป็นภาษาที่แทรกเทเบิลหรือไม่แทรกเทเบิลนั้น นอกจากเราจะใช้نيยามที่ผ่านมาในการอธิบายแล้ว เรายังสามารถใช้เอ็นพี-สมบูรณ์เพื่อช่วยในการพิจารณาได้อีกด้วยหนึ่ง

นิยาม 9.4.3: ภาษา L ถูกเรียกว่า เอ็นพี-สมบูรณ์ (NP-Complete) ถ้า $L \in NP$ และทุก ๆ ภาษาที่เป็นสมาชิกอยู่ใน NP สามารถลดทอนเข้าสู่ภาษา L ได้ภายในเวลาพหุนาม

ตัวอย่างที่ 9.8 จงให้แนวคิดในการพิสูจน์ว่าปัญหา SAT (Satisfiability Problem) เป็นเอ็นพี-สมบูรณ์

พิสูจน์ : สิ่งที่พิสูจน์ไปแล้ว ก็คือ $SAT \in NP$ แนวคิดในการพิสูจน์ในขั้นตอนต่อไปคือการลดทอนปัญหา N (โดยที่ $N \in NP$) ให้กลายเป็นปัญหา SAT โดยสามารถเข้ารหัสอินพุตติ้ง w ของทั่วจริงแบบชีนที่รองรับภาษา N ให้อยู่ในรูปแบบ CNF (Conjunctive Normal Form) จากนั้นป้อนให้กับทั่วจริงแบบชีนของ N เพื่อคำนวณ ถ้าทั่วจริงแบบชีนยอมรับ ก็แสดงว่านิพจน์ CNF นั้นเป็นจริง แต่ถ้าทั่วจริงแบบชีนไม่ยอมรับ ก็แสดงว่านิพจน์ CNF นั้นที่จ (สำหรับการพิสูจน์แบบสมบูรณ์จะมีความซับซ้อนและมีรายละเอียดมาก ผู้อ่านสามารถศึกษาเพิ่มเติมได้จากตำราดังประเทศซึ่งระบุอยู่ในภาคผนวกบรรณานุกรม) ดังนั้นจึงทำให้ปัญหา SAT เป็นเอ็นพี-สมบูรณ์

ตัวอย่างที่ 9.10 กำหนดให้ Clique คือ กราฟย่อยที่อยู่ภายใต้กราฟที่ไม่มีทิศทาง โดยที่ทุกคู่ของโหนดที่อยู่ภายใต้กราฟย่อยจะต้องเชื่อมต่อกัน สำหรับ k-Clique คือ Clique ที่มีโหนดอยู่ k โหนด เช่น กราฟที่มี 4-Clique แสดงดังรูป



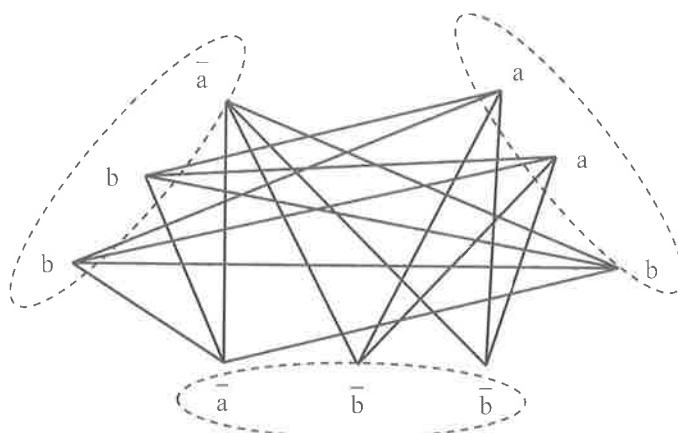
รูปที่ 9.2 ปัญหา 4-Clique

ปัญหา k -Clique คือ กำหนดให้กราฟ G เป็นกราฟแบบไม่มีทิศทาง ให้ตรวจสอบว่าภายในกราฟ G มี k -Clique อยู่หรือไม่ จงให้แนวคิดในการพิสูจน์ว่าปัญหา Clique เป็นอันพี-สมบูรณ์

พิสูจน์ : ก่อนอื่นเราจะต้องพิสูจน์ว่าปัญหา Clique เป็นปัญหา NP โดยสร้างทัวริงแมชีนที่คาดเดาไม่ได้ ตรวจสอบภายในเวลาพหุนาม ซึ่งสามารถสร้างได้ตามแนวคิดดังต่อไปนี้

1. เดาเซตย่อย s ซึ่งมีสมาชิกเป็นโหนดภายในกราฟ G โดยที่ $|s| = k$
2. ตรวจสอบว่ากราฟ G มีเส้นเชื่อมทุก ๆ โหนดที่อยู่ภายใน s หรือไม่ (ตอบใช่ หรือไม่ใช่)

ขั้นตอนต่อไป เนื่องจากปัญหา SAT เป็นปัญหาอันพี-สมบูรณ์ ดังนั้นเราจึงจะลดทอนปัญหา SAT ให้เป็นปัญหา Clique โดยเราจะจับกลุ่มโหนดภายในกราฟให้ได้ k กลุ่ม แต่ละกลุ่มมีจำนวนโหนดเท่า ๆ กัน นอกจากนี้ โหนดที่จะมาอยู่ในกลุ่มเดียวกันได้จะต้องไม่ใช่คอมพลีเมนต์ของกันและกัน และจะต้องไม่มีเส้นเชื่อมถึงกัน เช่น กราฟดังรูปสามารถจับกลุ่มเพื่อสร้างนิพจน์ CNF ได้ดังนี้



รูปที่ 9.3 กราฟที่จะนำมาเข้ารหัสเพื่อสร้างนิพจน์ CNF

จากนี้นป้อนนิพจน์ CNF คือ $(\bar{a} \vee b \vee b) \wedge (a \vee \bar{a} \vee b) \wedge (\bar{a} \vee \bar{b} \vee \bar{b})$ เข้าไปในทัวริงแมชีนของปัญหา SAT โดยในแต่ละวงเล็บให้เลือกตัวแปร 1 ตัว ซึ่งเมื่อกำหนดค่าบูลีน (จริง/เท็จ) ให้แล้วสามารถทำให้วงเล็บนั้นเป็นจริงได้ หากเราสามารถทำให้วงเล็บทุกวงเป็นจริง ผลที่ได้คือนิพจน์ CNF จะเป็นจริง ซึ่งก็หมายความว่า k -Clique อยู่ภายในกราฟ G นั้นเอง

แบบฝึกหัด

1. นิพจน์ดังต่อไปนี้สามารถทำให้มีค่าเป็นจริงได้หรือไม่

$$e = (a \vee b \vee c) \wedge (a \vee \bar{b} \vee c) \wedge (\bar{a} \vee b \vee \bar{c})$$

2. จงเขียนนิพจน์ $e = (a \wedge b) \vee c$ ในรูปแบบ CNF (Conjunctive Normal Form)

3. จากปัญหา SAT (Satisfiability Problem) ที่มีจำนวนตัวแปร n ตัว ซึ่งสามารถมีค่าบูลีน (Boolean) ที่เป็นไปได้ทั้งสิ้น 2^n จงแสดงว่าอัลกอริทึมในการเดาค่าต่าง ๆ ภายในนิพจน์สามารถทำได้ภายในเวลา $O(n)$

4. จงแสดงว่าภาษา $L = \{ww^R : w \in \{0, 1\}^+\}$ เป็น NTIME(n)

5. จงแสดงว่าภาษา $L = \{www : w \in \{0, 1\}^+\}$ เป็น NTIME(n)

6. จงยกตัวอย่างภาษาที่ซับซ้อนกว่าปัญหาประภาค NTIME(n^2)

7. จงยกตัวอย่างปัญหาแทร็กเทเบิลและปัญหาอินแทร็กเทเบิลอย่างละ 2 ปัญหา

8. ในการสร้างทัวริจเมชีนแบบคาดเดาไม่ได้เพื่อนำมาแก้ปัญหาการตรวจสอบว่าภายในกราฟมีลูปหรือไม่ จะต้องแทนกราฟให้อยู่ในรูปแบบของสตริงที่เข้ารหัส จงออกแบบวิธีการเข้ารหัสของกราฟนี้

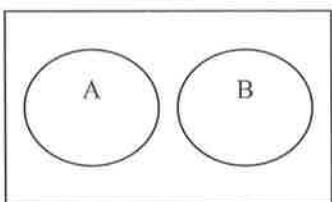
9. จงแสดงว่าปัญหา TSP (Traveling Salesman Problem) เป็นເອັນຝີ-ສມງຽນ

10. จงให้เหตุผลว่า ปัญหาการระบายสีแผนที่โดยใช้จำนวนสีที่จำกัด และจังหวัดที่มีพื้นที่อยู่ติดกันจะใช้สีเดียวกันไม่ได้ เป็นເອັນຝີ-ສມງຽນ

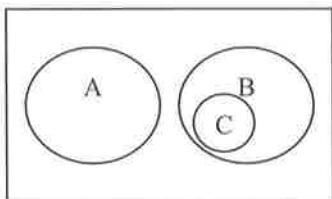
ເວລຍແບບຝຶກທັດກ້າຍບທ

ນທີ 1

1. (1) $\{0, 2, 4, 6, 8, \dots\}$
 (2) $\{0, 10, 100, 110, 1000, \dots\}$
 (3) $\{b, bb, abb, ababb, bbab, \dots\}$
 (4) $\{x, 0x0, 0x1, 10x, 011x010, \dots\}$
2. (1) $\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\}$
 (2) $\{a, b, c\}$
 (3) $\{(a, d), (a, e), (a, f), (b, d), (b, e), (b, f), (c, d), (c, e), (c, f)\}$
3. ແນວທາງໃນການພິສູຈຳ ເຮົາສາມາດໃຊ້ແພນກາພຂອງເວນນີ້ປະກອບການພິສູຈຳດັ່ງນີ້



$$A \cap B = \emptyset$$



$$C \subseteq B$$

4. ດຽວວ່າ \sim ໄດ້ມີຄວາມສັນເກີດແບບສົມມາຕຣ (Symmetric) ໃນມີຄວາມສັນເກີດແບບສະທ້ອນ (Reflexive) ເນື່ອງຈາກ
 ເຫດນີ້ຢັ້ງຊາດຄວາມສັນເກີດ $\{(1, 1), (3, 3)\}$
5. $\{w \in \{0, 1\}^*\mid w \text{ ເປັນເລກฐานສອງທີ່ເປັນຈຳນວນເລພາະ}\}$ ທັງນີ້ ເນື່ອງຈາກ
 $\{(1)_2, (11)_2, (101)_2, (111)_2, (1011)_2, (1101)_2, (10001)_2, (10011)_2, \dots\} =$
 $\{(1)_{10}, (3)_{10}, (5)_{10}, (7)_{10}, (11)_{10}, (13)_{10}, (17)_{10}, (19)_{10}, \dots\}$

6. ออโตมาตาดีอิเครื่องมือที่ใช้ในการรับรู้ภาษา มีประโยชน์ในการสร้างตัวรับรู้ภาษา เช่น ตัวแปลภาษา ผู้อ่านสามารถค้นหาส่วนประกอบของออโตมาตาแต่ละชนิดได้จากเนื้อหาในหนังสือเล่มนี้

7. เลขคู่คือเลขที่สามารถหารด้วย 2 ลงตัว เช่น $\{0, 2, 4, 6, \dots\}$ เราสามารถพิสูจน์คำนุมัตินี้ได้โดยใช้การพิสูจน์โดยอินดักชัน โดยมีแนวทางดังนี้

กรณีพื้นฐาน : พิสูจน์กรณี $0 * 0 = 0$

กรณีเฉพาะ : สมมุติว่า ถ้า a และ b เป็นเลขคู่ แล้ว $a * b$ เป็นเลขคู่ จากนั้นพิสูจน์ 3 กรณีย่อยดังนี้

- กรณีย่อยที่ 1 : $(a + 2) * b$ เป็นเลขคู่ เนื่องจาก $a + 2$ เป็นเลขคู่ และ b เป็นเลขคู่
- กรณีย่อยที่ 2 : $a * (b + 2)$ เป็นเลขคู่ เนื่องจาก a เป็นเลขคู่ และ $b + 2$ เป็นเลขคู่
- กรณีย่อยที่ 3 : $(a + 2) * (b + 2)$ เป็นเลขคู่ เนื่องจากทั้ง $a + 2$ และ $b + 2$ เป็นเลขคู่

8. พิสูจน์โดยอินดักชัน (เป็นการพิสูจน์แบบย่อโดยแสดงให้ดูเฉพาะจุดที่สำคัญเท่านั้น)

$$\text{สมมุติให้ } \sum_{i=1}^m i \cdot 2^i = (m - 1) \cdot 2^{m+1} + 2 \text{ เป็นจริงสำหรับ } i \leq m$$

$$\text{ถ้า } i = m + 1, \sum_{i=1}^{m+1} i \cdot 2^i = 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + 4 \cdot 2^4 + \dots + m \cdot 2^m + (m + 1) \cdot 2^{m+1}$$

$$\text{จากสมมุติฐาน } 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + 4 \cdot 2^4 + \dots + m \cdot 2^m = (m - 1) \cdot 2^{m+1} + 2$$

$$\text{ดังนั้น } \sum_{i=1}^{m+1} i \cdot 2^i = (m - 1) \cdot 2^{m+1} + 2 + (m + 1) \cdot 2^{m+1}$$

$$\sum_{i=1}^{m+1} i \cdot 2^i = (m - 1 + m + 1) \cdot 2^{m+1} + 2$$

$$\sum_{i=1}^{m+1} i \cdot 2^i = (2m) \cdot 2^{m+1} + 2$$

$$\sum_{i=1}^{m+1} i \cdot 2^i = (m) \cdot 2^{m+2} + 2$$

$$\sum_{i=1}^{m+1} i \cdot 2^i = (m + 1 - 1) \cdot 2^{m+1+1} + 2$$

9. แนวคิด : ใช้การพิสูจน์โดยอินดักชัน โดยพิสูจน์ที่ค่าของ n สำหรับกรณีพื้นฐาน ให้ n เท่ากับ 1 และให้พิสูจน์ว่า $\{1\}$ มีจำนวนเซตย่อยเท่ากับ 2^1 จากนั้นให้พิสูจน์กรณีเฉพาะ โดยให้พิสูจน์ว่าถ้า $n = n + 1$ สมมุติฐานยังคงถูกต้อง นั่นคือ ต้องได้จำนวนเซตย่อยเท่ากับ $2^{(n+1)}$

10. ใช้การพิสูจน์โดยอินดักชัน โดย

กรณีพื้นฐาน : ให้ $n = 4$ จะได้ว่า $4! > 2^4$

กรณีเฉพาะ : สมมุติว่า $n! > 2^n$ เป็นจริง จากนั้น สำหรับกรณี $n = n + 1$ เราต้องพิสูจน์ให้ได้ว่า $(n + 1)! > 2^{(n+1)}$

เนื่องจาก $1 * 2 * 3 * 4 * \dots * n > 2^n$ (จากสมมุติฐาน)

นอกจากนี้ จากการคำนวณทางคณิตศาสตร์ เราทราบว่า $(n + 1) > 2$ เนื่องจาก $n \geq 4$

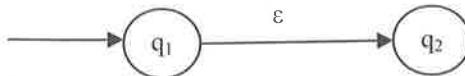
ดังนั้น $1 * 2 * 3 * 4 * \dots * n * (n + 1) > 2^n * 2$

นั่นคือ $(n + 1)! > 2^{(n+1)}$

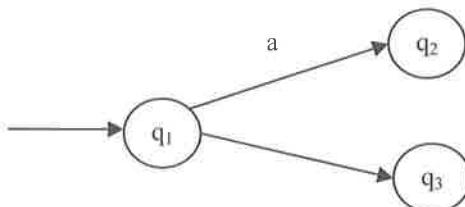
บทที่ ๒

๑. ความแตกต่างระหว่าง DFA กับ NFA คือ

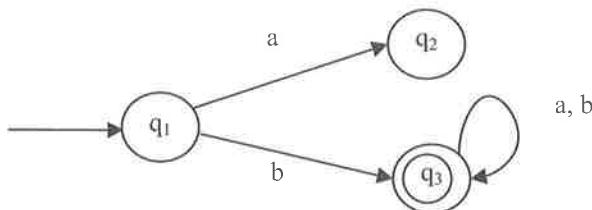
- NFA สามารถเปลี่ยนสถานะได้โดยไม่จำเป็นต้องรับอินพุตใด ๆ เช่น $q_1 \times \{\epsilon\} \rightarrow q_2$ (เรารู้ว่าเราสามารถเปลี่ยนสถานะได้โดยไม่ต้องรับอินพุตใด ๆ เช่น $q_1 \times \{a\} \rightarrow \{q_2, q_3\}$)



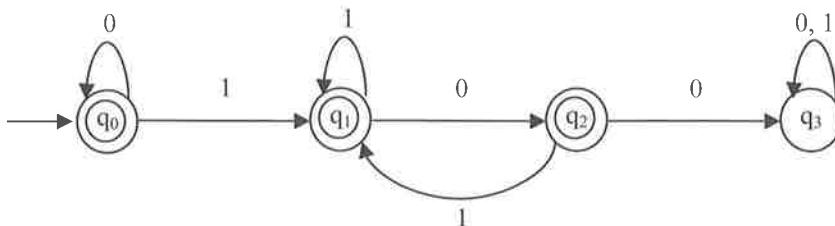
- จำนวนสถานะของ NFA (เพาเวอร์เซตของ Q) มีมากกว่าจำนวนสถานะของ DFA ทั้งนี้เนื่องจาก ณ สถานะใด ๆ ของ NFA เราสามารถเปลี่ยนไปยังสถานะอื่น ๆ ได้มากกว่าหนึ่งสถานะสำหรับอินพุตเดียวกัน เช่น $q_1 \times \{a\} \rightarrow \{q_2, q_3\}$



- สำหรับสถานะใด ๆ ของ NFA เราไม่จำเป็นต้องกำหนดทรานซิชันให้ครบสำหรับทุก ๆ อินพุตภายในเซต Σ เช่น ให้ $\Sigma = \{a, b\}$

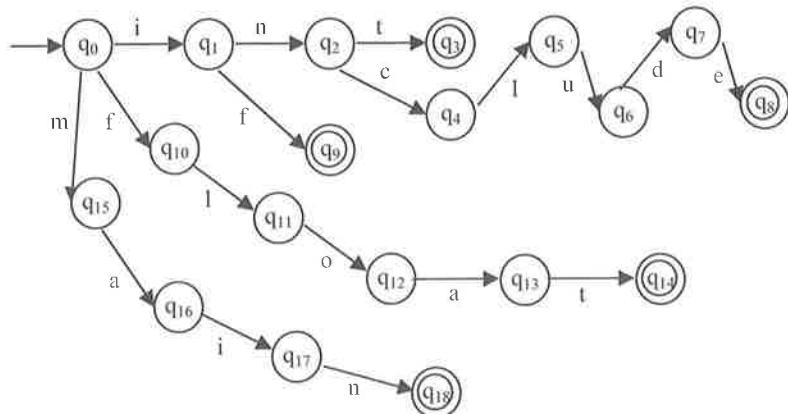


2.



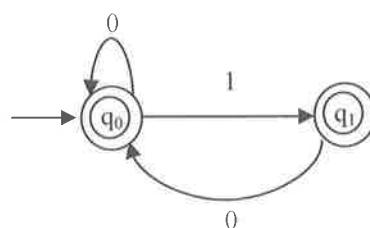
3. $\{01, 001, 101, 0001, 0101, 1001, 1101, \dots\}$

4.

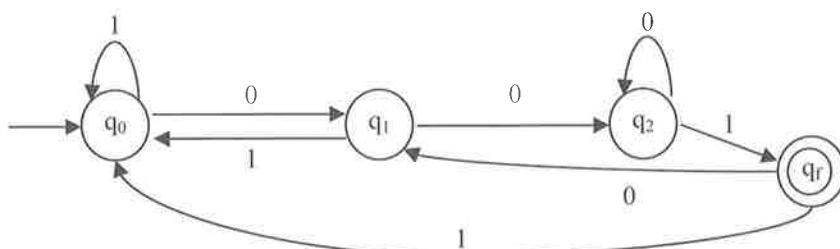


5. $L = \{x \in \{0, 1\}^* \mid x \text{ មិនមែនសញ្ញា } 11\}$

6.



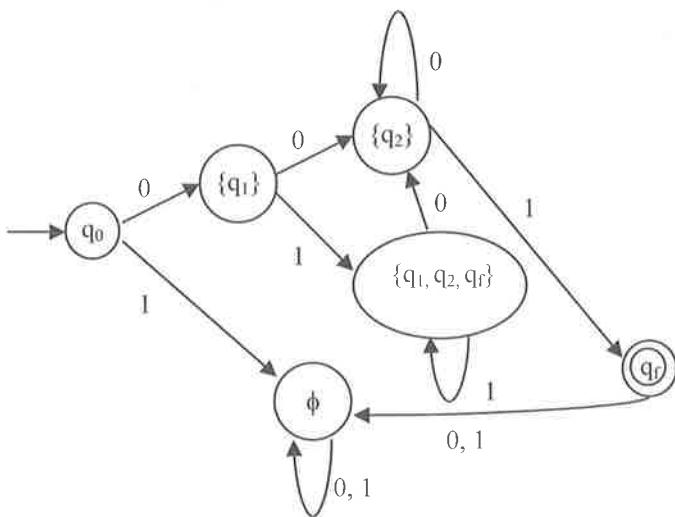
7.



8. $L = \{w \in \{0, 1\}^* \mid w \text{ លងឃាយតួយ } 1\}$

9.

δ	0	1
q_0	$\{q_1\}$	\emptyset
q_1	$\{q_2\}$	$\{q_1, q_2, q_f\}$
q_2	$\{q_2\}$	$\{q_f\}$
q_f	\emptyset	\emptyset



10. สร้างไม้ได้ เนื่องจากไฟในเตื้อต่อตามาให้มีหน่วยความจำที่จำกัด ทำให้ไม่สามารถจำว่าได้อ่านวงเล็บเปิดเข้ามาเกี่ยด้วยแล้ว ทั้งนี้เพื่อใช้ตรวจสอบว่ามีจำนวนวงเล็บเท่ากันกับจำนวนวงเล็บปิดที่อ่านเข้ามากายหลังหรือไม่

บทที่ ๓

$$(aaaa^*)^* = (aaa + aa)^*$$

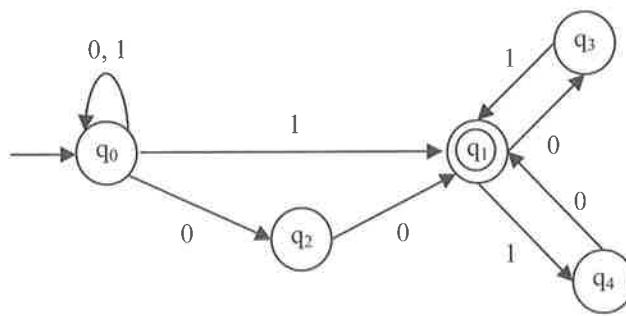
$$L((aaaa^*)^*) = \{a^n : n \geq 3\}$$

เหตุผลที่หั้งสองนิพจน์เรกูลาร์เท่าเทียมกัน สามารถอธิบายได้โดยการยกตัวอย่างการสร้างสตริงให้ดูจำนวนหนึ่งดังนี้

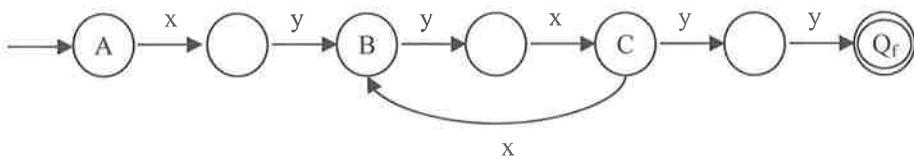
สตริงที่ได้จาก $\{a^n : n \geq 3\}$	สตริงที่ได้จาก $(aaa + aa)^*$
a^3	aaa
a^4	aa.aa
a^5	aaa.aa
a^6	aaa.aaa
a^7	aaa.aa.aa
a^8	aa.aa.aa.aa
a^9	aaa.aa.aa.aa
a^{10}	aa.aa.aa.aa.aa

จากตัวอย่างการสร้างสตริง หาก n เป็นเลขคู่ เราสามารถใช้ aa ที่อยู่ในนิพจน์เรกูลาร์ทางขวาสร้างสตริง a^n ได้เสมอ แต่ถ้า n เป็นเลขคี่ เราสามารถใช้ aaa หนึ่งนิพจน์ มาสมนกับการทำซ้ำ aa เพื่อให้ได้สตริงซึ่งมีจำนวน a เป็นเลขคี่ ดังนั้นไม่ว่า n จะมีค่าเป็นเท่าใด เราสามารถสร้างสตริง a^n จากนิพจน์เรกูลาร์ $(aaa + aa)^*$ ได้เสมอ

2. (1) $\{\epsilon, 0, 1, 10, 110, 11, \dots\}$
 (2) $\{\epsilon, 0, 1, 00, 10, 111, \dots\}$
 (3) $\{\epsilon, a, ba, ab, abaab, \dots\}$
 (4) $\{a, bb, aa, abb, ba, bbb, \dots\}$
 (5) $\{b, aab, bbb, aabbb, \dots\}$
 (6) $\{00, 000, 001, 100, 1000, \dots\}$
 (7) $\{\epsilon, 0, 1, 10, 01, 010, \dots\}$
3. $(a + b)^*(aa + ba + bb)$
- 4.



5. เนื่องจาก V_2 เป็นสถานะกับดัก ดังนั้นการเขียนนิพจน์เรกุляร์จึงไม่ต้องพิจารณา V_2 ดังนั้นคำตอบสำหรับข้อนี้คือ $(aa)^*(bb(bb)^* + \epsilon)$
6. $\{10, 010110, 010101110, 010101101110, \dots\}$
7. $L = \{1^n 0^m \mid n \geq 1, m \geq 2\}$
 $S \rightarrow 1A$
 $A \rightarrow 1A \mid 0B$
 $B \rightarrow 0 \mid 0B$
8. $S \rightarrow B00$
 $B \rightarrow B0 \mid A1$
 $A \rightarrow \epsilon \mid A1$
9. $A \longrightarrow xyB,$
 $B \longrightarrow yxC,$
 $C \longrightarrow xB \mid yy$



- ไวยากรณ์เรกูลาร์ สร้างภาษาเรกูลาร์
- นิพจน์เรกูลาร์ อธิบายภาษาเรกูลาร์
- ไฟฟ้าโนต์อโตมาตาเป็นเครื่องมือที่ใช้จำลองการยอมรับสตริงที่อยู่ในภาษาเรกูลาร์

บทที่ 4

1. การพิสูจน์ความเป็นเรกูลาร์ของภาษา สามารถทำได้ 4 วิธี คือ
 - สร้างไฟฟ้าโนต์อโตมาตารองรับภาษานั้น
 - สร้างไวยากรณ์เรกูลาร์รองรับภาษานั้น
 - สร้างนิพจน์เรกูลาร์รองรับภาษานั้น
 - การพิสูจน์โดยใช้คุณสมบัติปิดของภาษาเรกูลาร์
2. การพิสูจน์ความไม่เป็นเรกูลาร์ของภาษา สามารถทำได้ 2 วิธี คือ
 - การพิสูจน์โดยใช้คุณสมบัติปิดของภาษาเรกูลาร์
 - การใช้ปืนปิงเล่มมาสำหรับภาษาเรกูลาร์
3. $L = \{a^i b^j : j = i \text{ หรือ } j = 2i\}$
 แยกการพิสูจน์ปืนปิงเล่มมาเป็น 2 กรณี คือ
 - กรณีที่ $j = i$ โดยใช้สตริง $a^m b^m$
 - กรณีที่ $j = 2i$ โดยใช้สตริง $a^m b^{2m}$

ห้องสองกรณี ให้ทำซ้ำ $xy^i z$ ด้วยค่า $i = 0$ ก็จะทำให้สตริง $xy^i z \notin L$
4. $L = \{0^p 1^q : p \neq q\}$
 พิสูจน์โดยการใช้คุณสมบัติปิดของภาษาเรกูลาร์ดังนี้
 สมมุติว่า L เป็นภาษาเรกูลาร์
 เนื่องจาก $0^* 1^*$ เป็นภาษาเรกูลาร์ ดังนั้น $0^* 1^* - L$ จะต้องเป็นภาษาเรกูลาร์ด้วย
 $0^* 1^* - L = \{0^p 1^q : p = q\}$
 เราทราบมาแล้วว่าภาษา $\{0^p 1^q : p = q\}$ ไม่ใช่ภาษาเรกูลาร์ ดังนั้นสมมุติฐานที่เราตั้งไว้ว่า L เป็นภาษาเรกูลาร์จึงผิด
 \therefore ภาษา L จึงไม่ใช่ภาษาเรกูลาร์

5. $L = \{a^n b a^{2n} : n \geq 0\}$
 พิสูจน์โดยการใช้ปืนปิงเล่มมา โดยยกตัวอย่างสตริง $a^m b a^{2m}$ จากนั้นทำซ้ำสตริง $xy^i z$ ด้วยค่า $i = 0$ ผลจากการทำซ้ำจะทำให้สตริง $a^{m-k} b a^{2m} \notin L$ เนื่องจากทำให้จำนวนของ a ส่วนหนึ่งไม่เป็นครึ่งหนึ่งของจำนวน a ส่วนหลัง
6. $L = \{w \in \{a, b\}^* : \text{จำนวนตัวอักษร } a \text{ ภายในสตริง } w \text{ น้อยกว่าสองเท่าของจำนวนตัวอักษร } b \text{ ภายในสตริง } w\}$

พิสูจน์โดยการใช้ปืนปิงเล่มมาโดยยกตัวอย่างสตริง $a^{2m-1} b^m$ จากนั้นทำซ้ำสตริง $xy^i z$ ด้วยค่า $i = 2$ ผลจากการทำซ้ำจะทำให้สตริง

$$a^{2m-1+k} b^m \notin L$$

เนื่องจาก $k \geq 1$ จึงทำให้จำนวนของ a มีโอกาสที่จะเท่ากับสองเท่าของจำนวน b ได้ ซึ่งเป็นผลให้สตริงดังกล่าวไม่เป็นสมาชิกอยู่ภายใน L

$$7. L = \{b^{n,m!} : n \geq 1\}$$

พิสูจน์โดยการใช้ปั๊มปิงเลนมาโดยยกตัวอย่างสตริง $b^{m,m!}$ จากนั้นทำชี้สตริง xy^iz ด้วยค่า $i = 2$ ผลจากการทำซ้ำจะได้สตริง $b^{m,m! + k}$ ซึ่งเราต้องพิสูจน์ต่ออีกว่า

$$m.m! + k \stackrel{?}{=} p.p!$$

ค่าของ p ที่มีโอกาสทำให้สมการเป็นจริงที่สุดคือ $m + 1$ ดังนั้นเราริบพยายามจัดรูปทางขวาของสมการให้เป็น $m + 1$ ให้ได้ นั่นคือ

$$m.m! + k \stackrel{?}{=} (m + 1).(m + 1)!$$

$$\text{เนื่องจาก } m.m! + k \leq m.m! + m \text{ สำหรับ } 1 \leq k \leq m$$

$$< m.(m + 1)! + (m + 1)!$$

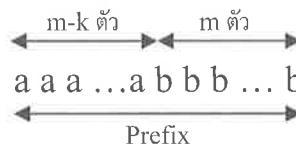
$$< (m + 1).(m + 1)!$$

$$\therefore m.m! + k < (m + 1).(m + 1)!$$

ดังนั้นจึงไม่มีค่า p ใด ๆ ที่จะทำให้ $m.m! + k = p.p!$ ได้ ซึ่งเป็นผลให้ $b^{m,m! + k} \notin L$

$$8. L = \{w \in \{a, b\}^* : \text{ไม่มี Prefix ใด ๆ ของ } w \text{ ที่มีจำนวนตัวอักษร } b \text{ มากกว่า } a\}$$

พิสูจน์โดยการใช้ปั๊มปิงเลนมาโดยยกตัวอย่างสตริง $a^m b^m$ จากนั้นทำชี้สตริง xy^iz ด้วยค่า i เท่ากับ 0 ผลจากการทำซ้ำจะได้ $a^{m-k} b^m$ โดยที่ $k \geq 1$



เมื่อพิจารณาดูจาก Prefix ของสตริงดังรูป จะเห็นได้อย่างชัดเจนว่าจำนวนของตัวอักษร b มีมากกว่าตัวอักษร a อยู่อย่างน้อยหนึ่งตัว ดังนั้นทำให้ $a^{m-k} b^m \notin L$

$$9. L = \{0^i 1^j : j \text{ เป็นจำนวนเท่าของ } i\}$$

พิสูจน์โดยการใช้ปั๊มปิงเลนมาโดยยกตัวอย่างสตริง $0^m 1^{2m}$ จากนั้นทำชี้สตริง xy^iz ด้วยค่า i เท่ากับ 0 ผลจากการทำซ้ำจะได้ $0^{m-k} 1^{2m}$ โดยที่ $k \geq 1$ ซึ่งเป็นที่ชัดเจนว่า $0^{m-k} 1^{2m} \notin L$

$$10. L = \{a^i b^j a^k : k > i + j\}$$

พิสูจน์โดยการใช้ปั๊มปิงเลนมาโดยยกตัวอย่างสตริง $a^m b^m a^{2m+1}$ จากนั้นทำชี้สตริง xy^iz ด้วยค่า i เท่ากับ 2 ผลจากการทำซ้ำจะได้ $a^{m+p} b^m a^{2m+1}$ โดยที่ $1 \leq p \leq m$

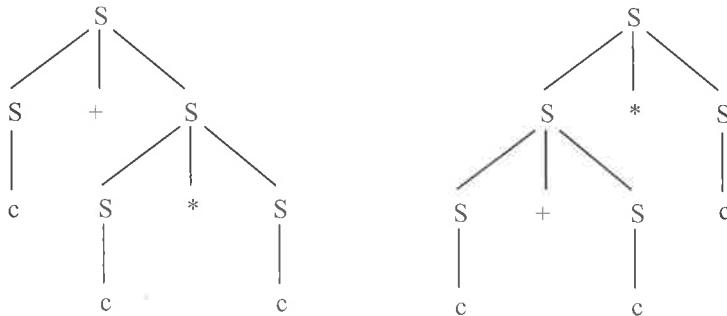
เนื่องจาก $m + p + m$ ไม่มีโอกาสที่จะน้อยกว่า $2m + 1$ ได้เลย ดังนั้นจึงทำให้ $a^{m+p} b^m a^{2m+1} \notin L$

บทที่ ๕

- การพิสูจน์ว่าภาษาหนึ่งเป็นภาษาค่อนเท็กซ์ฟรีสามารถทำได้ ๓ วิธีดังนี้
 - สร้างไวยากรณ์ค่อนเท็กซ์ฟรีของรับภาษานั้น
 - สร้างพุทธawan อิตามาตราของรับภาษานั้น ซึ่งเราจะได้ศึกษาในบทต่อไป
 - พิสูจน์โดยอาศัยคุณสมบัติปิดของภาษาค่อนเท็กซ์ฟรี

- ยกตัวอย่างสตริง $w = c + c^*c$

เราสามารถสร้างต้นไม้แสดงการได้มา (Derivation Tree) ที่แตกต่างกันได้สองต้นดังนี้



ดังนั้นไวยากรณ์ดังกล่าวจึงเป็นไวยากรณ์ที่ลับสน

- $G_1 = (\{S\}, \{a, b\}, S, P_1)$

$$P_1: S \rightarrow aSb \mid SS \mid \epsilon$$

$$G_2 = (\{S\}, \{a, b\}, S, P_2)$$

$$P_2: S \rightarrow aSb \mid abS \mid \epsilon$$

ไวยากรณ์ G_1 และ G_2 มีความคล้ายคลึงกันมาก แต่มีอย่างสตริงที่สามารถบ่งบอกถึงความแตกต่างของไวยากรณ์ทั้งสอง นั่นคือ $aabbab$ เมื่อจาก $aabbab \in L(G_1)$ แต่ $aabbab \notin L(G_2)$ ดังนั้นจึงเป็นการพิสูจน์ได้ว่า $L(G_1) \neq L(G_2)$

- $L = \{w \in \{a, b\}^*: \text{จำนวนตัวอักษร } a \text{ ภายในสตริง } w \text{ มีจำนวนเป็น } 2 \text{ เท่าของจำนวนตัวอักษร } b \text{ ภายในสตริง } w\}$

พิสูจน์โดยการสร้างไวยากรณ์ค่อนเท็กซ์ฟรีขึ้นมาของรับดังนี้

$$\begin{aligned} S &\rightarrow aabS \mid aaSb \mid aSab \mid Saab \\ &\quad abaS \mid abSa \mid aSba \mid Saba \\ &\quad baaS \mid baSa \mid bSaa \mid Sbaa \mid \epsilon \end{aligned}$$

- $L = \{a^i b^j \mid i \leq j\}$

พิสูจน์โดยการสร้างไวยากรณ์ค่อนเท็กซ์ฟรีขึ้นมาของรับดังนี้

$$S \rightarrow aSb \mid B \mid \epsilon$$

$$B \rightarrow bB \mid b$$

6. $L = \{a^i b^j c^k \mid k = i + j\}$

พิสูจน์โดยการสร้างไวยากรณ์ค่อนเทิกซ์ฟรีชื่นมารองรับดังนี้

$$S \rightarrow aSc \mid B \mid \epsilon$$

$$B \rightarrow bBc \mid \epsilon$$

7. $L = \{a^i b^j c^k \mid j = i + k\}$

พิสูจน์โดยการสร้างไวยากรณ์ค่อนเทิกซ์ฟรีชื่นมารองรับดังนี้

$$S \rightarrow XY \mid X \mid Y \mid \epsilon$$

$$X \rightarrow aXb \mid ab$$

$$Y \rightarrow bYc \mid bc$$

8. $S \rightarrow aSb \mid bSa \mid \epsilon$

$L = \{w \in \{a, b\}^*: \text{จำนวนตัวอักษร } a \text{ ใน } w \text{ เท่ากับจำนวนตัวอักษร } b \text{ ใน } w\}$

9. $S \rightarrow aSa \mid bSb \mid a \mid b$

$L = \{vav^R \text{ หรือ } vbv^R \text{ โดยที่ } v \in \{a, b\}^*\}$

10. $S \rightarrow aA \mid bA \mid a \mid b$

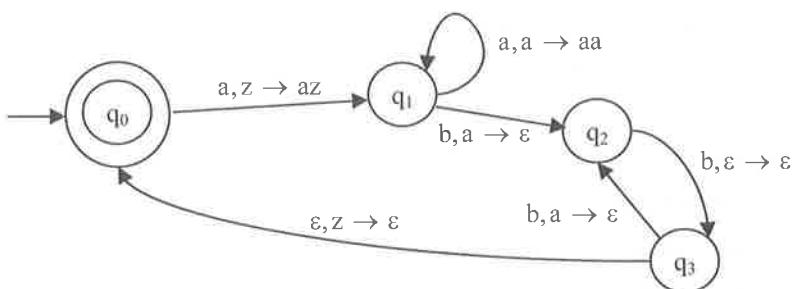
$$A \rightarrow aS \mid bS$$

$L = \{w \in \{a, b\}^* : |w| \in \{1, 3, 5, 7, 9, 11, \dots\}\}$

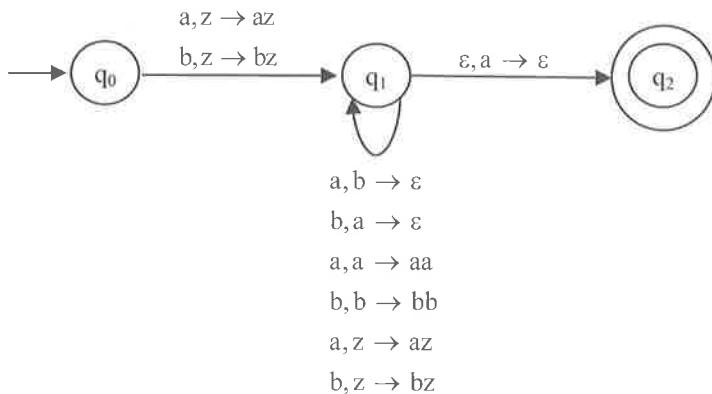
บทที่ 6

1. ผู้อ่านสามารถค้นหาคำตอบได้จากหัวข้อ “พุชดาวน์อโตมาตาที่คาดเดาไม่ได้”

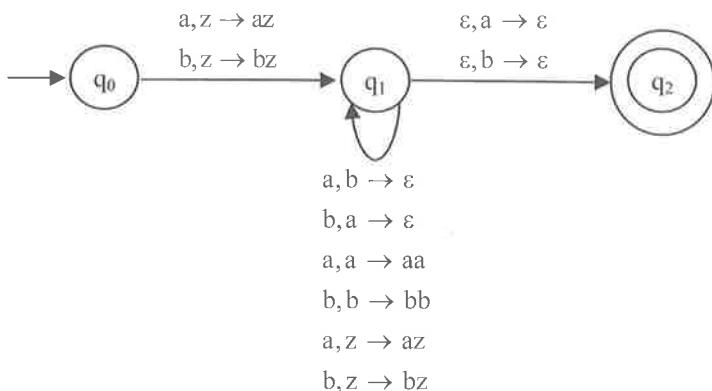
2. $L = \{a^n b^{2n} \mid n \geq 0\}$



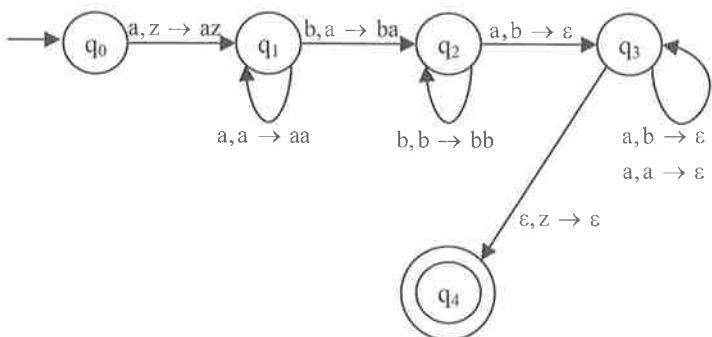
$$3. \quad L = \{w \in \{a, b\}^* \mid n_a(w) > n_b(w)\}$$



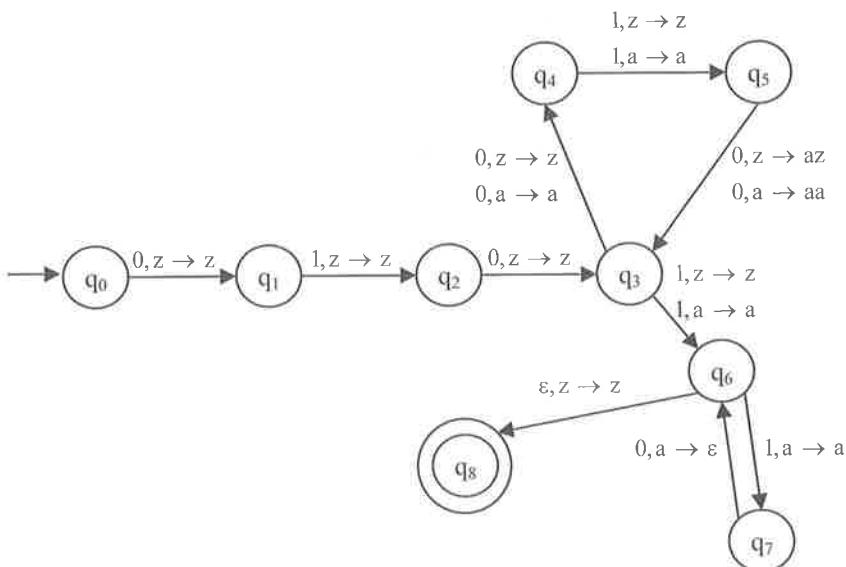
$$4. \quad L = \{w \in \{a, b\}^* \mid n_a(w) \neq n_b(w)\}$$



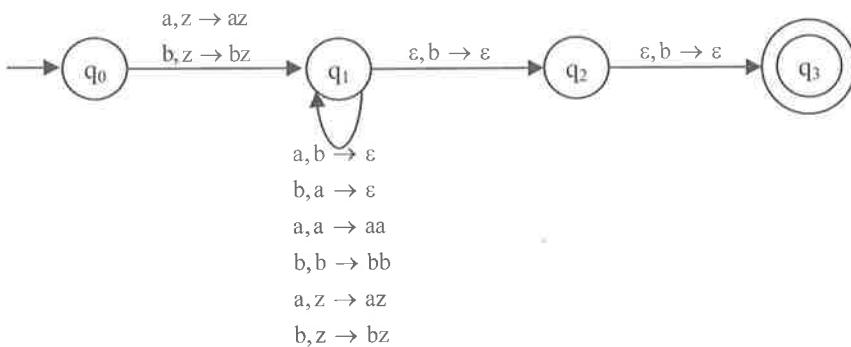
$$5. \quad L = \{a^n b^m a^{n+m} \mid n, m \geq 1\}$$



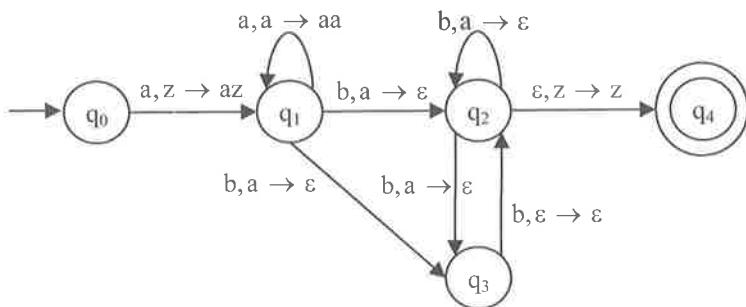
6. $L = \{010(010)^m 1(10)^n : m \geq 0\}$



7. $L = \{w \in \{a, b\}^* \mid n_a(w) + 1 < n_b(w)\}$



8. $L = \{a^n b^m \mid n \leq m \leq 2n\}$



$$9. L = \{ww^Rw \mid w \in \{a, b\}^*\}$$

ผู้อ่านคงยังจำได้ว่า ww^R เป็นภาษาตอนเทิร์กฟรีซึ่งเราสามารถสร้าง NPDA ขึ้นมารองรับได้ โดยเมื่อ NPDA ยอมรับสตริง ww^R แล้วสแต็กจะว่าง (อาจจะคงเหลือแต่ลัญตักขยับแสดงจุดเริ่มต้นของสแต็ก) ดังนั้นสำหรับสตริง w อีกส่วนหนึ่งที่ตามหลัง ww^R มา เราไม่สามารถทราบได้เลยว่า w ส่วนหน้ามีลักษณะเป็นอย่างไร เนื่องจากถูกนำออกไปจากสแต็กจนหมดแล้ว ดังนั้นจึงทำให้พูดตามนี้ไม่สามารถรองรับภาษา L ได้

$$10. \text{NPDA } M = (\{q_0, q_1, q_f\}, \{0, 1\}, \{0, 1, z\}, \delta, q_0, z, \{q_f\})$$

$$\delta : \delta(q_0, 0, z) = \{(q_1, 0), (q_f, \epsilon)\},$$

$$\delta(q_1, 1, 1) = (q_1, 1),$$

$$\delta(q_1, 1, 0) = (q_1, 1),$$

$$\delta(q_1, 0, 1) = (q_f, 1)$$

$$\text{ดังนั้น } L(M) = \{0\} \cup L(011^*0)$$

บทที่ 7

1. การพิสูจน์ความไม่เป็นค่อนเทิร์กฟรีของภาษา สามารถทำได้ 2 วิธีคือ

- การพิสูจน์โดยใช้ปั๊มปิงเลนมาสำหรับภาษาตอนเทิร์กฟรี
- การพิสูจน์โดยอาศัยคุณสมบัติพิเศษของภาษาตอนเทิร์กฟรี

สำหรับข้อที่ 2 ถึง 9 จะเป็นการพิสูจน์แบบย่อ โดยเฉลยจะเป็นแนวทางในการพิสูจน์แบบเต็ม

$$2. L = \{a^n b^{3n} a^n \mid n \geq 1\}$$

พิสูจน์โดยใช้ปั๊มปิงเลนมาสำหรับภาษาตอนเทิร์กฟรี

ยกตัวอย่างสตริง $a^m b^{3m} a^m$ ซึ่งจะถูกแบ่งออกเป็น 5 กรณีหลัก ๆ ได้ดังนี้

$$(1) vxy = aa\dots a$$

$$(2) vxy = aa\dots abb\dots b$$

กรณีนี้จะถูกแบ่งออกเป็น 3 กรณีย่อยดังนี้

- v เป็น a อย่างเดียว และ y เป็น b อย่างเดียว

- v เป็น a อย่างเดียว และ y เป็น $aa\dots abb\dots b$

- v เป็น $aa\dots abb\dots b$ และ y เป็น b อย่างเดียว

$$(3) vxy = bb\dots b$$

$$(4) vxy = bb\dots baa\dots a$$

กรณีนี้จะถูกแบ่งออกเป็น 3 กรณีย่อยในทำนองเดียวกันกับกรณี (2)

$$(5) vxy = aa\dots a$$

ในทุก ๆ กรณี เราสามารถทำขั้นตอนนี้ได้ คือ เท่ากับเท่าใดก็ได้ (ยกเว้น 1) เพื่อทำให้สตริง $uv^i xy^i z$ ไม่เป็นสมาชิกอยู่ภายใต้ภาษา L

$$3. L = \{w \in \{a, b, c\}^* \mid n_a(w) = \min(n_b(w), n_c(w))\}$$

พิสูจน์โดยใช้ปั๊มปิงเลนมาสำหรับภาษาตอนเทิกซ์พรี

ยกตัวอย่างสตริง $a^m b^m c^{m+1}$ ซึ่งจะถูกแบ่งออกเป็น 5 กรณีหลัก ๆ ได้ดังนี้

(1) $vxy = aa\dots a$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(2) $vxy = aa\dots abb\dots b$

กรณีนี้จะถูกแบ่งออกเป็น 3 กรณีย่อย คือ

- v เป็น a อย่างเดียว และ y เป็น b อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 3$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

- v เป็น a อย่างเดียว และ y เป็น $aa\dots abb\dots b$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

- v เป็น $aa\dots abb\dots b$ และ y เป็น b อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(3) $vxy = bb\dots b$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(4) $vxy = bb\dots bcc\dots c$

กรณีนี้จะถูกแบ่งออกเป็น 3 กรณีย่อยในหนอนเดียวกันกับกรณี (2) โดยทั้ง 3 กรณีย่อยนี้ เราสามารถทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(5) $vxy = cc\dots c$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 0$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

$$4. L = \{a^i b^j c^k \mid i > j > k\}$$

พิสูจน์โดยใช้ปั๊มปิงเลนมาสำหรับภาษาตอนเทิกซ์พรี

ยกตัวอย่างสตริง $a^{m+2} b^m c^m$ ซึ่งจะถูกแบ่งออกเป็น 5 กรณีหลัก ๆ ได้ดังนี้

(1) $vxy = aa\dots a$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 0$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(2) $vxy = aa\dots abb\dots b$

กรณีนี้จะถูกแบ่งออกเป็น 3 กรณีย่อย คือ

- v เป็น a อย่างเดียว และ y เป็น b อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 0$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

- v เป็น a อย่างเดียว และ y เป็น $aa\dots abb\dots b$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

- v เป็น $aa\dots abb\dots b$ และ y เป็น b อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(3) $vxy = bb\dots b$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 0$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(4) $vxy = bb\dots bcc\dots c$

กรณีนี้จะถูกแบ่งออกเป็น 3 กรณีย่อย คือ

- v เป็น b อย่างเดียว และ y เป็น c อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 3$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

- v เป็น b อย่างเดียว และ y เป็น $bb\dots bcc\dots c$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$
- v เป็น $bb\dots bcc\dots c$ และ y เป็น c อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(5) $vxy = cc\dots c$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

$$5. L = \{a^n b^m c^{n+m} \mid m, n \geq 1\}$$

พิสูจน์โดยใช้ปั๊มปิงเล่มมาสำหรับภาษาค่อนเทิกซ์พรี

ยกตัวอย่างสตริง $a^m b^m c^{2m}$ ซึ่งจะถูกแบ่งออกเป็น 5 กรณีหลัก ๆ ได้ดังนี้

- (1) $vxy = aa\dots a$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$
- (2) $vxy = aa\dots abb\dots b$

กรณีนี้จะถูกแบ่งออกเป็น 3 กรณีย่อย คือ

- v เป็น a อย่างเดียว และ y เป็น b อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$
- v เป็น a อย่างเดียว และ y เป็น $aa\dots abb\dots b$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$
- v เป็น $aa\dots abb\dots b$ และ y เป็น b อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(3) $vxy = bb\dots b$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(4) $vxy = bb\dots bcc\dots c$

กรณีนี้จะถูกแบ่งออกเป็น 3 กรณีย่อยคือ

- v เป็น b อย่างเดียว และ y เป็น c อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$
- v เป็น b อย่างเดียว และ y เป็น $bb\dots bcc\dots c$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$
- v เป็น $bb\dots bcc\dots c$ และ y เป็น c อย่างเดียว ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

(5) $vxy = cc\dots c$ ทำช้าสตริง $uv^i xy^j z$ ด้วยค่า $i = 2$ เพื่อทำให้สตริง $uv^i xy^j z \notin L$

$$6. L = \{a^n b^m a^n b^m \mid m, n \geq 1\}$$

ภาษานี้ไม่เป็นภาษาค่อนเทิกซ์พรี เนื่องจากอันที่จริงแล้ว ภาษานี้เป็นเซตย่อของภาษา $\{ww : w \in \{a, b\}^*\}$ ซึ่งเราเคยทราบจากตัวอย่างของบทนี้แล้วว่าไม่ใช่ภาษาค่อนเทิกซ์พรี ดังนั้นวิธีการพิสูจน์โดยบีบีปิงเล่มมาจึงมีความคล้ายคลึงกัน

$$7. \ L = \{wcw \mid w \in \{a\}^*\}$$

ภาษาที่เป็นภาษาค่อนเท็กซ์ฟรี เนื่องจากเป็นภาษาเดียวกันกับ $\{a^nca^n : n \geq 0\}$ ซึ่งเราสามารถสร้างพุชดาวน์อัตโนมายาชื่นmarginรับได้ โดยจะนำตัวอักษร a ส่วนแรกใส่ลงไปในสแต็ก จากนั้นเมื่อพบจุดกึ่งกลางสตริง(ตัวอักษร c) ให้นำสตริงส่วนหลังตรวจสอบว่าตัวอักษรที่กำลังอ่านอยู่เป็นตัวอักษรเดียวกับ a ซึ่งอยู่ส่วนบนสุดของสแต็กหรือไม่ ถ้าเหมือนกันก็นำตัวอักษร a บนสแต็กออก จนกว่าจะไม่พบจุดกึ่งกลางสตริงที่อ่านเข้ามา

$$8. \ L = \{w \in \{a, b\}^* \mid n_a(w) = 3 n_b(w)\}$$

ภาษาที่เป็นภาษาค่อนเท็กซ์ฟรี เนื่องจากความสามารถสร้างพุชดาวน์อัตโนมายาชื่นmarginรับได้ดังนี้

- ทุกครั้งที่พบตัวอักษร a และสัญลักษณ์บนสุดของสแต็กไม่ใช่ b ให้ใส่ตัวอักษร a ลงไปบนสแต็ก
- ทุกครั้งที่พบตัวอักษร b และสัญลักษณ์บนสุดของสแต็กไม่ใช่ a ให้ใส่ตัวอักษร b ลงไปบนสแต็ก
- ทุกครั้งที่พบตัวอักษร a และสัญลักษณ์บนสุดของสแต็กเป็น b ให้นำ b นั้นออกจากสแต็ก
- ทุกครั้งที่พบตัวอักษร b และสัญลักษณ์บนสุดของสแต็กเป็น a ให้นำ a นั้นออกจากสแต็ก

$$9. \ \text{คอมพลีเมนต์ของภาษา } L \text{ โดยที่ } \bar{L} = \{a^i b^j c^k \mid i \geq j \text{ or } j \geq k\}$$

คอมพลีเมนต์ของภาษา L ไม่เป็นภาษาค่อนเท็กซ์ฟรี โดยก่อนอื่นจะพิสูจน์ให้ด้วกว่าภาษา L เป็นภาษาค่อนเท็กซ์ฟรีโดยการสร้างไวยกรณ์ค่อนเท็กซ์ฟรีที่ร่องรับภาษา L ดังนี้

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow AC$$

$$A \rightarrow aAb \mid B \mid \epsilon$$

$$B \rightarrow aB \mid a$$

$$C \rightarrow cC \mid \epsilon$$

$$S_2 \rightarrow DE$$

$$D \rightarrow aD \mid \epsilon$$

$$E \rightarrow bEc \mid F \mid \epsilon$$

$$F \rightarrow cF \mid c$$

เนื่องจากภาษาค่อนเท็กซ์ฟรี ไม่มีคุณสมบัติปิดภายใต้การคอมพลีเมนต์

$$\bar{L} = \{a, b, c\}^* - \{a^i b^j c^k \mid i \geq j \text{ or } j \geq k\}$$

ดังนั้นคอมพลีเมนต์ของภาษา L จึงไม่ใช่ภาษาค่อนเท็กซ์ฟรี

อันที่จริง ภาษาหนึ่งที่เป็นเซตย่อยของภาษา \bar{L} คือ $\{a^i b^j c^k \mid i < j < k\}$ ซึ่งเราไม่สามารถสร้างพุชดาวน์อัตโนมายารองรับได้อยู่แล้ว ดังนั้น \bar{L} จึงไม่ใช่ภาษาค่อนเท็กซ์ฟรี

10. บีบีปิงเลมมาสำหรับภาษาค่อนเท็กซ์ฟรี มีความแตกต่างจากบีบีปิงเลมมาสำหรับภาษาเรกูลาร์ ดังนี้

- การแบ่งสตริงจะถูกแบ่งออกเป็น 5 ส่วน uvxyz โดยทุก ๆ กรณีจะต้องได้รับการพิสูจน์
- การทำซ้ำสตริงจะทำซ้ำ 2 ส่วน คือ v และ y

บทที่ 8

- ทั่วเรียงแมชีนมีความแตกต่างจากพุชดาวน์อ็อโตมาตาดังนี้
 - ทั่วเรียงแมชีนใช้อินพุตเทปเป็นหน่วยความจำ แต่พุชดาวน์อ็อโตมาตาใช้สแต็ก
 - อินพุตเทปของทั่วเรียงแมชีนสามารถอ่านและเขียนได้
 - หัวอ่านของทั่วเรียงแมชีนสามารถเคลื่อนที่ซ้าย-ขวาได้ แต่หัวอ่านของพุชดาวน์อ็อโตมาตาเคลื่อนที่ไปทางขวาได้เท่านั้น
 - $L = \{a^n b^m \mid n \geq m \text{ และ } n, m \neq 0\}$
- แนวคิด : (1) ขับคู่ระหว่าง a และ b โดย
- แทน a ตัวซ้ายมือสุดด้วย x
 - แทน b ตัวซ้ายมือสุดด้วย y
- (2) ถ้ายังเหลือ a ที่ไม่สามารถจับคู่กับ b ได้ ให้ยอมรับสตริงนั้น
- $L = \{ww^Rw \mid w \in \{a, b\}^+\}$
- แนวคิด : สำหรับภาษา w ที่เป็นต้องใช้ทั่วเรียงแมชีนแบบคาดเดาไม่ได้ โดยใช้เทคนิคการเดารอยต่อระหว่าง สตริงอย่าง w, w^R, w เมื่ອនกันกับตัวอย่างที่เคยศึกษาแล้วในบทเรียน คือ $\{ww \mid w \in \{a, b\}^*\}$
- $L = \{w \in \{0, 1, 2\}^+ \mid n_0(w) = n_1(w) = n_2(w)\}$
- แนวคิด : ภาษานี้มีวิธีสร้างทั่วเรียงแมชีนคล้ายกันกับภาษาในตัวอย่าง $\{w \in \{a, b\}^+ \mid n_a(w) = n_b(w)\}$ แต่ทั่วเรียงแมชีนของภาษานี้จะมีขนาดใหญ่กว่าค่อนข้างมาก ดังนั้นผู้อ่านเพียงทำความเข้าใจในหลักการก็เพียงพอแล้ว
- $L = \text{เซตของสตริงปีกกาในภาษา } C \text{ ซึ่งมีจำนวนของปีกกาเปิดและปีกกาปิดเท่ากัน เช่น } \{\}, \{\{\}\}, \{\{\{\}\}\} \text{ เป็นต้น}$
- แนวคิด : (1) อ่านอินพุตจากซ้ายไปขวาจนพบ } และเปลี่ยน } ให้เป็น y จากนั้นเคลื่อนหัวอ่านไปทางซ้ายจนกระทั่งพบ { และเปลี่ยน { ให้เป็น x เพื่อจับคู่กับ } ก่อนหน้านี้
- (2) ทำซ้ำขั้นตอนแรก แต่ให้เริ่มหัวอ่านจากตำแหน่งซ้ายมือสุดของลักษณะบนเทป จนกระทั่ง { และ } ถูกจับคู่ทั้งหมด
- $L = \{a^n b^{2n} c^{3n} \mid n \geq 1\}$
- แนวคิด : วิธีสร้างทั่วเรียงแมชีนสำหรับภาษานี้จะคล้ายคลึงกับภาษา $\{a^n b^n c^n\}$ แต่แตกต่างกันตรงที่เมื่อพบตัวอักษร a หนึ่งตัว จะต้องมาจับคู่กับตัวอักษร b สองตัว และจับคู่กับตัวอักษร c สามตัว
- $f(n) = 3n$
- แนวคิด : วิธีสร้างทั่วเรียงแมชีนสำหรับภาษานี้มีความคล้ายคลึงกับภาษา $f(w) = 2w$ จากตัวอย่างในบทเรียน แต่แตกต่างกันตรงที่หลังจากที่เปลี่ยนเลข 1 ในบูนารี n ให้เป็น x หมวดแล้ว ให้เคลื่อนหัวอ่านไปจนพบซองว่าง แล้วแทรก 11 ลงบนเทป (แทนที่จะแทรกเพียง 1 ตัวเดียว)
- $f(n) = n + 3$
- แนวคิด : ทั่วเรียงแมชีนสำหรับภาษานี้สามารถสร้างได้อย่างง่าย ๆ โดยให้อ่านตัวเลขบูนารีบนเทปจากซ้ายไปขวาจนกระทั่งพบซองว่าง หลังจากนั้นก็ให้แทรก 111 ลงไปบนเทป

$$9. \quad f(n) = \begin{cases} 1 & \text{ถ้า } n \text{ เป็นจำนวนเท่าของ } 3 \text{ เช่น } 0, 3, 6, 9, \dots \\ 0 & \text{ถ้า } n \text{ ไม่เป็นจำนวนเท่าของ } 3 \text{ เช่น } 1, 2, 4, 5, 7, \dots \end{cases}$$

แนวคิด : แทน n ด้วยตัวเลขฐานารีและจับคู่เลข 1 ที่ลับสามตัว (111) ถ้าสามารถจับคู่ได้หมด ให้ลงเทป แล้วเขียน 1 ลงบนเทป มิฉะนั้นให้ลงเทปและเขียน 0 แทน

$$10. \quad f(n) = \max (f_1(n), f_2(n))$$

แนวคิด : ทำการคำนวณหาผลลัพธ์ของ $f_1(n)$ และ $f_2(n)$ โดยใช้ 0 คืนระหว่างผลลัพธ์ทั้งสอง จากนั้นตรวจสอบว่าผลลัพธ์ของจำนวนใดที่มีจำนวนเลข 1 มากกว่ากัน แล้วลงผลลัพธ์ของฟังก์ชัน ที่น้อยกว่าด้วยช่องว่าง