204433 การแปลภาษาโปรแกรม
การบ้านที่ 1

ให้นิสิตตอบคำถามและเขียนโปรแกรมต่อไปนี้

1. จาก grammar ของ expression ทางคณิตศาสตร์ที่เราได้กล่าวถึงในห้องเรียน

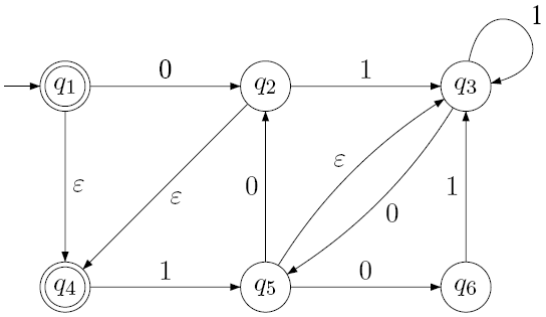
```
expression = ["+" | "-"] , term , {("+" | "-") , term};
term       = factor , {("*" | "/" | "%") , factor};
factor     = constant | "(" , expression , ")";
constant   = digit , {digit};
digit      = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
```

ให้นิสิตเขียน grammar ใหม่เพื่อให้รองรับ operator "^" ที่แทนการยกกำลัง และให้ว่า operator นี่มี precedence สูงที่สุด แต่ไม่สูงไปกว่าการใส่วงเล็บ

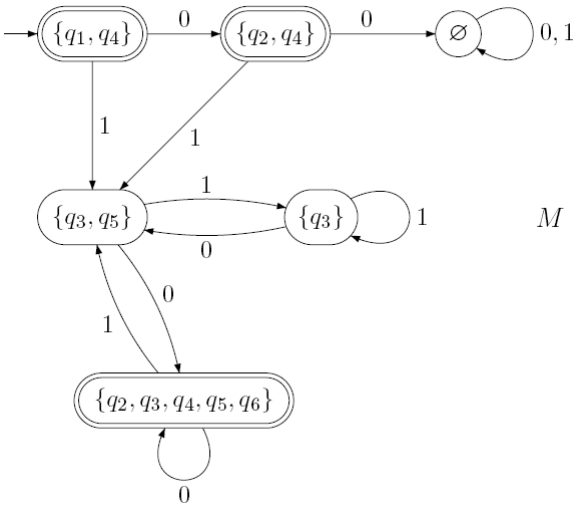
```
expression = ["+" | "-"] , term , {("+" | "-") , term};
term       = exp , {("*" | "/" | "%") , exp};
exp  = factor , {“^”, factor}
factor     = constant | "(" , expression , ")";
constant   = digit , {digit};
digit      = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | “9”;
```

การบ้านที่ 2

1. แปลง NFA ต่อไปนี้ให้เป็น DFA

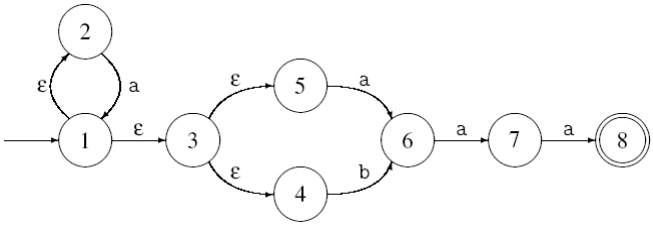


ได้ DFA ดังแสดงต่อไปนี้



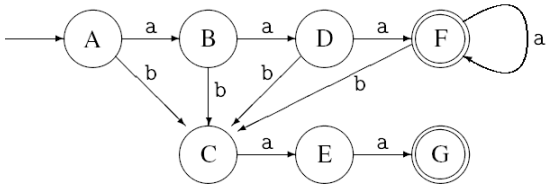
2. ให้ regular expression ต่อไปนี้ a*(a | b)aa จงหา
   - NFA

ได้ NFA ดังต่อไปนี้

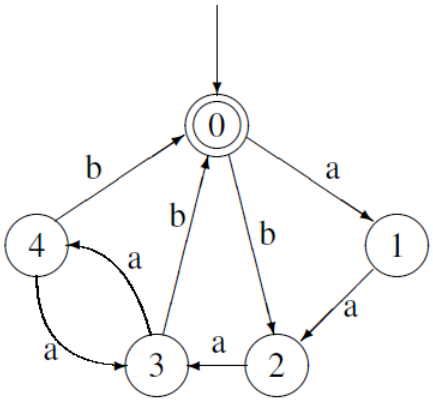


   - เปลี่ยน NFA ให้เป็น DFA

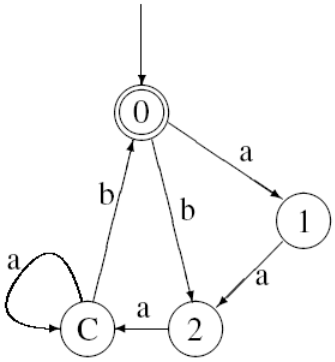
ได้ DFA ดังต่อไปนี้



3. ลดรูป (minimize) DFA ต่อไปนี้โดยใช้อัลกอริทึมของ Hopcroft ที่เราได้คุยกันในชั้นเรียน



ลดรูปได้ดังต่อไปนี้



4. ทำความเข้าใจและอธิบายการทำงานของ Maximum Munch Scanner นิสิตที่สนใจเรื่องนี้เป็นพิเศษ อาจารย์แนะนำให้อ่านบทความต่อไปนี้ที่อาจารย์ได้ให้ไว้พร้อมกับเลคเชอร์ที่ 6

   Thomas Reps, "`Maximal munch' tokenization in linear time", ACM TOPLAS, 20(2), March 1998, pp 259-273.

   (ดูโค๊ดในเลคเชอร์ที่ 3 ประกอบคำอธิบายต่อไปนี้ด้วย)
   โค๊ดที่เพิ่มเติมเข้ามาใน Maximum Munch Scanner เพื่อกำจัดการ rollback ที่มากจนเกินเหตุนั้นมีดังต่อไปนี้
   - **เพิ่ม global counter InputPos เพื่อทำการบันทึกตำแหน่งของ input stream**

- **เพิ่ม bit array 2 มิติ Failed เพื่อบันทึก transition ที่เป็น dead-end กล่าวคือเป็น transition ที่ไม่นำเข้าหา accepted states แถวของ Failed มีไว้สำหรับ state แต่ละ state และคอลัมน์ของ Failed มีไว้สำหรับตำแหน่งของ input stream**

เมื่อเรารู้ว่าคู่ของ <state, ตำแหน่งของ input stream> ใดที่จะนำไปสู่ dead-end แล้ว เราก็จะ break ออกจาก while loop แรกทันที การบันทึกว่าคู่ <state, ตำแหน่งของ input stream> ใดๆ จะนำไปสู่ dead-end นั้น กระทำใน while loop ที่สองขณะที่คู่ลำดับ <state, ตำแหน่งของ input stream> แต่ละอันถูก pop ออกจาก stack
ไฮไลท์ในส่วนที่เพิ่มเติมต่อไปนี้

```
// recognize words
NextWord() {
    state ¬ s0
    lexeme ¬ empty string
    clear stack
    push (bad,bad)
    while (state ¹ se) do
      char ¬ NextChar( )
      InputPos ¬ InputPos + 1
      lexeme ¬ lexeme + char
      if Failed[state,InputPos]
         then break;
      if state ∈ SA
        then clear stack
      push (state,InputPos)
      state ¬ d(state,char)
    end

    // clean up final state
    while (state ∉ SA and state ≠ bad) do
       Failed[state,InputPos) ¬ true
      〈state,InputPos〉 ← pop()
       truncate lexeme
       roll back the input one character
    end
    // report the results
    if (state ∈ SA)
       then return lexeme
       else  return invalid
}
```

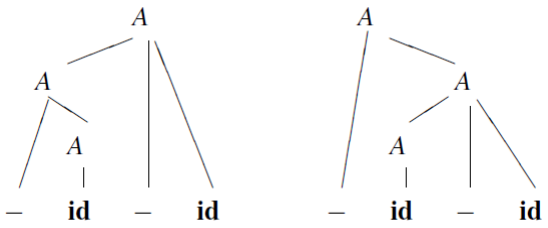
5. เขียน CFG ที่ accept สตริงที่มีจำนวนตัวอักษร a เท่ากับจำนวนตัวอักษร b

S -> Ɛ
S -> aSbS
S -> bSaS

6. จงแสดงว่า grammar ต่อไปนี้
A -> - A
A -> A - id
A -> id

มี ambiguity โดยการหาสตริงหนึ่งตัวที่มี parse tree ได้มากกว่าหนึ่งรูปแบบ แสดงสตริงที่ได้และ parse tree ทั้งสอง จากนั้นให้เขียน grammar ใหม่โดยกำจัด ambiguity ทิ้ง และแสดง parse tree ที่ได้จากการ derive สตริงที่เราหามาในตอนแรกเพื่อพิสูจน์ว่า grammar มี ambiguity

พิจารณาสตริง: - 5 - 4
มี parse tree ได้สองรูปแบบ



ด้านซ้ายให้ผลลัพธ์ - 5 - 4 = - 9 ส่วนด้านขวาให้ผลลัพธ์ - (5 - 4) = 1

แก้ grammar เพื่อกำจัด ambiguity โดยให้ unary - มี precedence มากที่สุด ได้ grammar ดังต่อไปนี้
A -> A - B
A -> B
B -> -B
B -> id

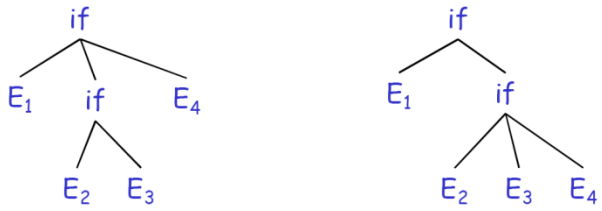
7. พิจารณา grammar ของประโยค if-then-else ต่อไปนี้

S -> if E then S | if E then S else S | OTHER

โดย S แทน non-terminal ที่เป็น statement ของโปรแกรม E แทน non-terminal ที่เป็น expression ของโปรแกรม และ OTHER เป็น non-terminal หรือ terminal ที่ไม่เกี่ยวข้องกับประโยค if-then-else ที่เรากำลังพิจารณา

- อธิบายว่าทำไม grammar นี้ ambiguous (คำใบ้: ลองนึกถึงที่เราได้คุยกันในชั้นเรียนว่า ambiguous grammar คือ grammar ที่เราสามารถสร้าง parse tree ได้มากกว่าหนึ่งแบบเวลาที่เราทำ derivation เพื่อตรวจจับสตริง)

  ประโยคต่อไปนี้ if E1 then E2 then E3 else E4 มี parse tree ได้มากกว่าหนึ่งรูปแบบดังต่อไปนี้



8. กำจัด left recursion ใน grammar ต่อไปนี้ให้หมดสิ้น

$$
\begin{aligned}
Exp &\rightarrow Exp + Exp \\
Exp &\rightarrow Exp - Exp \\
Exp &\rightarrow Exp * Exp \\
Exp &\rightarrow Exp / Exp \\
Exp &\rightarrow \mathbf{num} \\
Exp &\rightarrow (Exp)
\end{aligned}
$$

เขียน grammar ใหม่ที่กำจัด left recursion ออกได้ทั้งหมดดังต่อไปนี้

$$
\begin{aligned}
Exp &\rightarrow \mathbf{num}\, Exp_1 \\
Exp &\rightarrow (\, Exp\,)\, Exp_1 \\
Exp_1 &\rightarrow + Exp\, Exp_1 \\
Exp_1 &\rightarrow - Exp\, Exp_1 \\
Exp_1 &\rightarrow * Exp\, Exp_1 \\
Exp_1 &\rightarrow / Exp\, Exp_1 \\
Exp_1 &\rightarrow \varepsilon
\end{aligned}
$$

9. กำจัด left recursion ใน grammar ต่อไปนี้ และนำผลลัพธ์ของ grammar ที่ได้มาทำ left factoring
   E -> E E+
   E -> E E *
   E -> num

กำจัด left-recursion ได้โดยการเขียน grammar ด้านบนใหม่ดังต่อไปนี้

$$
\begin{aligned}
E &\rightarrow \mathbf{num}\, E' \\
E' &\rightarrow E + E' \\
E' &\rightarrow E * E' \\
E' &\rightarrow \varepsilon
\end{aligned}
$$

ทำการ left-factor หลังจากกำจัด left-recursion ได้โดยการเขียน grammar ใหม่ดังต่อไปนี้

$$
\begin{aligned}
E &\rightarrow \mathbf{num}\, E' \\
E' &\rightarrow E\, Aux \\
E' &\rightarrow \varepsilon \\
Aux &\rightarrow + E' \\
Aux &\rightarrow * E'
\end{aligned}
$$

10. คำนวณหา FIRST และ FOLLOW ของ non-terminal และ terminal ใน grammar ต่อไปนี้
    Z -> d
    Z -> XYZ
    Y -> ε
    Y -> c
    X -> Y
    X -> a

    จากนั้นสร้างตาราง predictive parsing และบอกว่า grammar นี้เป็น LL(1) หรือไม่ โดยให้ X Y และ Z เป็น non-terminal ส่วน a c และ d เป็น terminal

FIRST(X) = {a, c, ε}
FIRST(Y) = {c, ε}
FIRST(Z) = {a, c, , d}
FIRST(a) = {a}
FIRST(c) = {c}
FIRST(d) = {d}

FOLLOW(X) = {a, c, d}
FOLLOW(Y) = {a, c, d}
FOLLOW(Z) = {}
FOLLOW(a) = {}
FOLLOW(c) = {}
FOLLOW(d) = {}

ตาราง predictive parsing เป็นดังต่อไปนี้

	a	c	d
X	$X \to a$ $X \to Y$	$X \to Y$	$X \to Y$
Y	$Y \to \varepsilon$	$Y \to \varepsilon$ $Y \to c$	$Y \to \varepsilon$
Z	$Z \to XYZ$	$Z \to XYZ$	$Z \to d$ $Z \to XYZ$

และจะได้ว่า grammar นี้ไม่ใช่ LL(1) เนื่องจากในตาราง parsing ในบางช่องของคู่ non-terminal และ terminal เชน X กับ a ที่มี production ที่เป็นไปได้มากกว่าหนึ่งอยู่