ปฏิบัติการที่ 11
ไวยากรณ์และพาร์สทรี

รหัสนักศึกษา 6204062616081 ชื่อและนามสกุล พิทยา ศรีหล้า ตอน 1

1. กำหนดไวยากรณ์ CFG มาให้

S -> NP VP,
NP -> Det N,
PP -> P NP,
VP -> 'slept',
VP -> 'saw' NP,
VP -> 'walked' PP,
Det -> 'the',
Det -> 'a',
N -> 'man',
N -> 'park',
N -> 'dog',
P -> 'in',
P -> 'with'

จงแสดงการทำ Rightmost derivation ในการสร้างประโยค the dog walked in the park (ทำมือ)

<u>ตอบ</u>

S => NP **VP**
=> NP ‘walked’ **PP**
=> MP ‘walked’ P **NP**
=> NP ‘walked’ P Det **N**
=> NP ‘walked’ P **Det** ‘park’
=> NP ‘walked’ **P** ‘the’ ‘park’
=> **NP** ‘walked’ ‘in’ ‘the’ ‘park’
=> Det **N** ‘walked’ ‘in’ ‘the’ ‘park’
=> **Det** ‘dog’ ‘walked’ ‘in’ ‘the’ ‘park’
=> ‘the’ ‘dog’ ‘walked’ ‘in’ ‘the’ ‘park’

2. กำหนดไวยากรณ์ Backus-Naur form (BNF) มาให้ดังต่อไปนี้

<assign> → <var> = <expr>

<expr> → <var> - <expr> | <var>+<expr> | <var>+<var> | <var>

<var>→ a | b | c | d

2.1 จงนำไปสร้างไวยากรณ์ด้วย NLTK (CFG format)

```
grammar = CFG.fromstring("""
assign -> var '=' expr
expr -> var '-' expr | var '+' expr | var '+' var | var
var -> 'a' | 'b' | 'c' | 'd'
""")
```

2.2 เขียนคำสั่งเพื่อพิมพ์แสดงจำนวน Production แสดงคำสั่งและผลการรัน

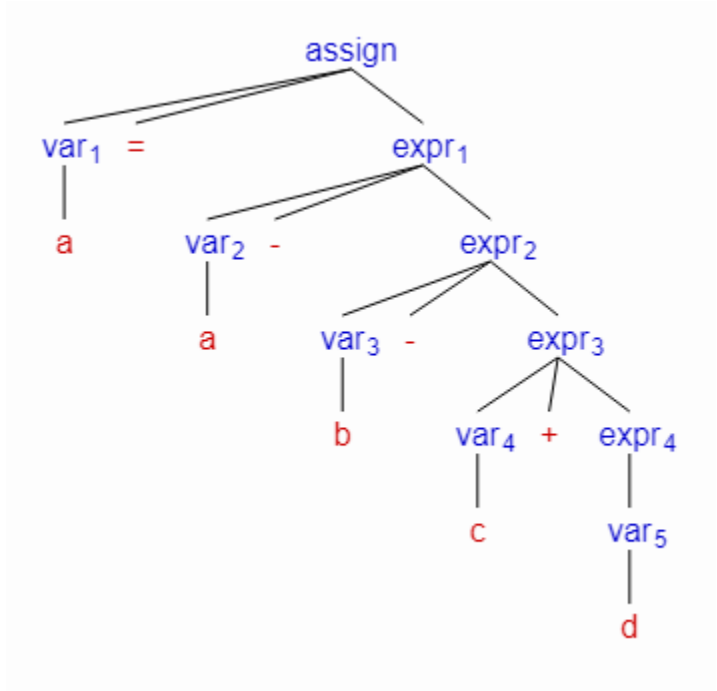
```
print("Number of Productions:", len(grammar.productions()))
```

```
Number of Productions: 9
```

2.3 เขียนคำสั่งเพื่อแสดงประโยคโดยกำหนด depth = 6 และ n = 20 แสดงคำสั่งและผลการรัน

```
a = a - a - a + a
a = a - a - a + b
a = a - a - a + c
a = a - a - a + d
a = a - a - b + a
a = a - a - b + b
a = a - a - b + c
a = a - a - b + d
a = a - a - c + a
a = a - a - c + b
a = a - a - c + c
a = a - a - c + d
a = a - a - d + a
a = a - a - d + b
a = a - a - d + c
a = a - a - d + d
a = a - a - a
a = a - a - b
a = a - a - c
a = a - a - d
```

2.4 เขียนคำสั่งเพื่อสร้างประโยค a = a – b – c + d และแสดงพาร์สทรีพร้อมรูปต้นไม้



2.5 แสดงการทำ Derivation แบบ Leftmost เพื่อสร้างประโยคในข้อ 2.4

<u>ตอบ</u>

<assign> → **<var>** = <expr>

=> a = **<expr>**
=> a = **<var>** - <expr>
=> a = a - **<expr>**
=> a = a - **<var>** - <expr>
=> a = a - b - **<expr>**
=> a = a - b - **<var>** + <expr>
=> a = a - b - c + **<expr>**
=> a = a - b - c + **<var>**
=> a = a - b - c + d

3. กำหนดไวยากรณ์มาให้

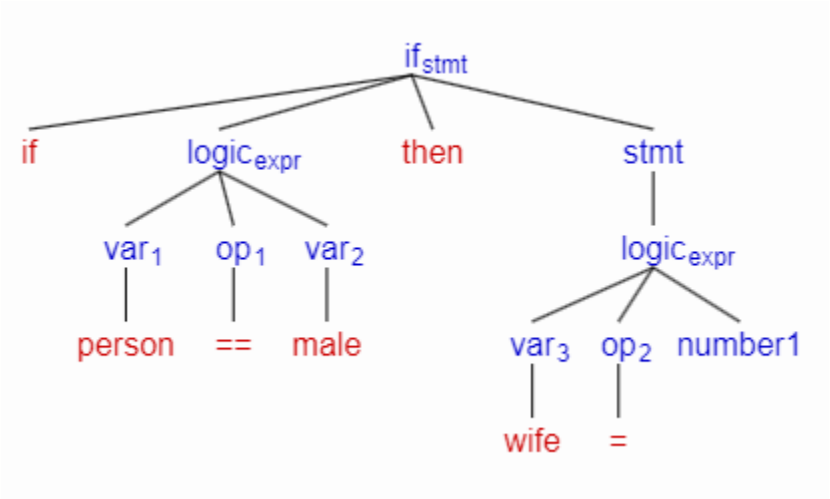
```
grammar = CFG.fromstring("""
if_stmt -> 'if' logic_expr 'then' stmt 'else' stmt | 'if' logic_expr 'then' stmt
logic_expr -> var op var
stmt -> if_stmt | logic_expr
op -> '==' | '='
var -> 'person' | 'male' | 'kid' | 'wife' | '0' | 'status' | 'married' | '1'
""")
```

3.1 จงเขียนคำสั่งแสดงประโยคด้วยการกำหนดคำสั่ง depth=5, n=10

```
depth = 5
n = 10
print("Sentence generated by the NLTK parser")
for sentence in generate(grammar,depth=5,n=10):
   print(' '.join(sentence))
```

```
Sentence generated by the NLTK parser
if person == person then person == person else person == person
if person == person then person == person else person == male
if person == person then person == person else person == kid
if person == person then person == person else person == wife
if person == person then person == person else person == 0
if person == person then person == person else person == status
if person == person then person == person else person == married
if person == person then person == person else person == 1
if person == person then person == person else person = person
if person == person then person == person else person = male
```

3.2 จงสร้างประโยค if Person == male then Wife = 1 พร้อมทั้งแสดงพาร์สทรีที่ได้และรูปภาพของต้นไม้ (แสดงคำสั่งและผลลัพธ์)



3.3 แสดงการทำ Derivation แบบ Rightmost เพื่อแสดงประโยคข้อ 3.2

<u>ตอบ</u>

if_stmt => if logic_expr then **stmt**

=> if logic_expr then **logic_expr**
=> if logic_expr then var op **var**
=> if logic_expr then var **op** 1
=> if logic_expr then **var** = 1
=> if **logic_expr** then wife = 1
=> if var op **var** then wife = 1
=> if var **op** male then wife = 1
=> if **var** == male then wife = 1
=> if parson == male then wife = 1

4. กำหนดไวยากรณ์ในรูปแบบ Backus-Naur form (BNF)

<E> → <E> + <E> | <E> * <E> | (<E>) | - <E> | <literal_value>

<literal_value> -> 10 | 20 | 30

4.1 จงสร้างไวยากรณ์ใน NLTK

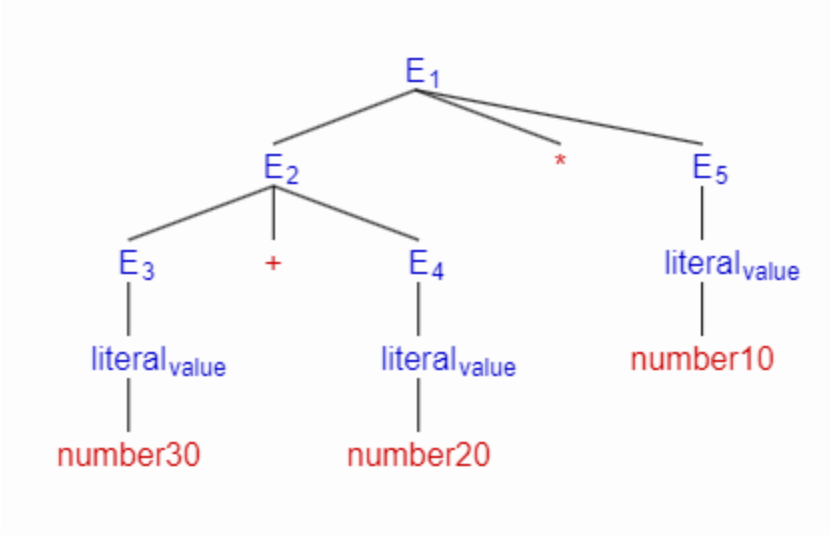
```
grammar = CFG.fromstring("""
E -> E '+' E | E '*' E | '('E')' | '-' E | literal_value
literal_value -> '10' | '20' | '30'
""")
```

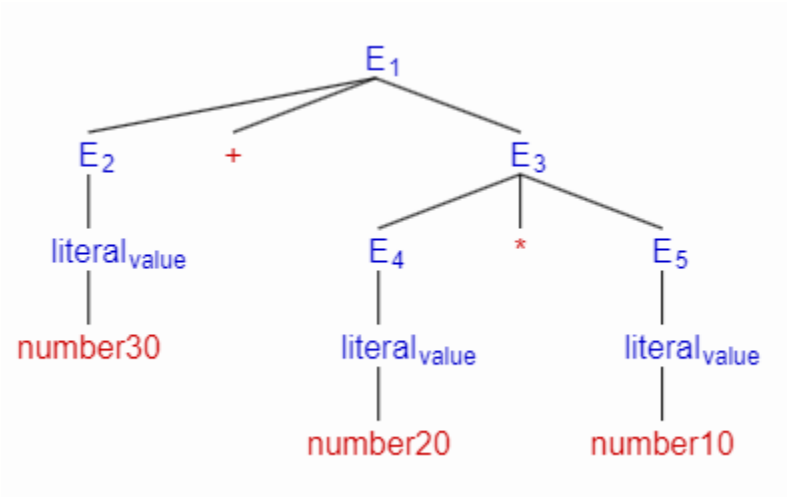
4.2 จงเขียนคำสั่งแสดงประโยคด้วยการกำหนดคำสั่ง depth=5,n=20

```
print("Sentence generated by the NLTK parser")
for sentence in generate(grammar,depth=5,n=20):
   print(' '.join(sentence))
```

```
Sentence generated by the NLTK parser
10 + 10 + 10 + 10
10 + 10 + 10 + 20
10 + 10 + 10 + 30
10 + 10 + 20 + 10
10 + 10 + 20 + 20
10 + 10 + 20 + 30
10 + 10 + 30 + 10
10 + 10 + 30 + 20
10 + 10 + 30 + 30
10 + 10 + 10 * 10
10 + 10 + 10 * 20
10 + 10 + 10 * 30
10 + 10 + 20 * 10
10 + 10 + 20 * 20
10 + 10 + 20 * 30
10 + 10 + 30 * 10
10 + 10 + 30 * 20
10 + 10 + 30 * 30
10 + 10 + ( 10 )
10 + 10 + ( 20 )
(E
```

4.3 จงสร้างประโยค '30 + 20 * 10' แล้วแสดงพาร์สทรี และวาดต้นไม้ทั้งหมดที่ได้





4.4 ผลลัพธ์ที่ได้จากต้นไม้ทั้งสองเป็นอย่างไร ? ไวยากรณ์ดังกล่าวมีความกำกวมหรือไม่กำกวม?

ตอบ

ไวยากรณ์นี้มีความกำกวม เพราะ เนื่องจากไม่มีการระบุลำดับของตัวดำเนินการอย่างชัดเจนจึงทำให้สามารถตีความประโยค ``30 + 20 * 10'' ได้หลายแบบ