Name: Daniel Schlatter

Date: 6/28/19

Class: CSCI 3415

Program 1 Report

Abstract

This report contains the problem statement for Program 1, my approach to coding the program, my test results, and conclusions about what I learned while coding it.

Problem Statement

Create a Python program that reads in a grammar from an input file and then uses that grammar to print leftmost derivations (if they exist) of sentences read from standard input. Your Python program should accept a command-line argument specifying the grammar file. Your program must be able to parse the example grammar files on Canvas. 3 After parsing the grammar file and printing the parsed grammar, your program will accept sentences from standard input and then print the leftmost derivation for each, if they exist. Your code should follow the Python coding standards as defined in PEP 8 – Style Guide for Python Code.

Approach

The method to parsing the grammar file is simple, separate each line into a tuple of the left side and right side, then the right side may be separated into a list depending on the number of lexemes in it. Print the resulting grammar list going line by line, separating tuple elements with a -> symbol.

Parsing the user entered sentence is done recursively. The derivation method is called with the starting lexeme of the grammar file as a form argument, the grammar file, and the user entered sentence. The sentence is then checked to see if it matches the current form, and if it doesn't, it won't return anything and this iteration will be thrown out, otherwise, itll return an empty list if it's a full match or a list of the matching form with the recursive call fo the function if it's a form match but not a full match yet. Each recursive call is done on a list of applicable rules from the grammar file. After the full match has been found, the list is built from the bottom up, and then passed to the print function.

Results

Results with sentences from Problem Set 3 – 6:

```
In [3]: %run derive.py example_3.2.txt
Reading grammar from example_3.2.txt
<assign> -> <id> = <expr>
<id> -> A
<id> -> B
<id> -> C
<expr> -> <id> + <expr>
<expr> -> <id> * <expr>
<expr> -> ( <expr> )
<expr> -> <id>
---
Enter a sentence:

A = A * ( B + ( C * A ) )
```

```
Sentence:

A = A * ( B + ( C * A ) )

Derivation:

0: <assign> -> <id> = <expr>

1: -> A = <expr>

2: -> A = <id> * <expr>

3: -> A = A * <expr>

4: -> A = A * ( <expr> )

5: -> A = A * ( <id> + <expr> )

6: -> A = A * ( B + <expr> )

7: -> A = A * ( B + ( <expr> ) )

8: -> A = A * ( B + ( <id> * <expr> ) )

9: -> A = A * ( B + ( C * <expr> ) )

10: -> A = A * ( B + ( C * <id> ) )

11: -> A = A * ( B + ( C * A ) )

---

Enter a sentence:


B = C * ( A * C + B )

Sentence:

B = C * ( A * C + B )

Derivation:

0: <assign> -> <id> = <expr>

1: -> B = <expr>

2: -> B = <id> * <expr>

3: -> B = C * <expr>

4: -> B = C * ( <expr> )
```

```
5: -> B = C * ( <id> * <expr> )

6: -> B = C * ( A * <expr> )

7: -> B = C * ( A * <id> + <expr> )

8: -> B = C * ( A * C + <expr> )

9: -> B = C * ( A * C + <id> )

10: -> B = C * ( A * C + B )

---

Enter a sentence:


A = B * ( C * ( A + B ) )

Sentence:

A = B * ( C * ( A + B ) )

Derivation:

0: <assign> -> <id> = <expr>

1: -> A = <expr>

2: -> A = <id> * <expr>

3: -> A = B * <expr>

4: -> A = B * ( <expr> )

5: -> A = B * ( <id> * <expr> )

6: -> A = B * ( C * <expr> )

7: -> A = B * ( C * ( <expr> ) )

8: -> A = B * ( C * ( <id> + <expr> ) )

9: -> A = B * ( C * ( A + <expr> ) )

10: -> A = B * ( C * ( A + <id> ) )

11: -> A = B * ( C * ( A + B ) )

---

Enter a sentence:
```

Results with Problem set 3-7:

```
In [1]: %run derive.py example_3.4.txt

Reading grammar from example_3.4.txt

<assign> -> <id> = <expr>

<id> -> A

<id> -> B

<id> -> C

<expr> -> <expr> + <term>

<expr> -> <term>

<term> -> <term> * <factor>

<term> -> <factor>

<factor> -> ( <expr> )

<factor> -> <id>

---

Enter a sentence:


A = ( A + B ) * C

Sentence:

A = ( A + B ) * C

Derivation:

0: <assign> -> <id> = <expr>

1: -> A = <expr>

2: -> A = <term>

3: -> A = <term> * <factor>

4: -> A = <factor> * <factor>
```

```
5: -> A = ( <expr> ) * <factor>

6: -> A = ( <expr> + <term> ) * <factor>

7: -> A = ( <term> + <term> ) * <factor>

8: -> A = ( <factor> + <term> ) * <factor>

9: -> A = ( <id> + <term> ) * <factor>

10: -> A = ( A + <term> ) * <factor>

11: -> A = ( A + <factor> ) * <factor>

12: -> A = ( A + <id> ) * <factor>

13: -> A = ( A + B ) * <factor>

14: -> A = ( A + B ) * <id>

15: -> A = ( A + B ) * C

---

Enter a sentence:


A = B + C + A

Sentence:

A = B + C + A

Derivation:

0: <assign> -> <id> = <expr>

1: -> A = <expr>

2: -> A = <expr> + <term>

3: -> A = <expr> + <term> + <term>

4: -> A = <term> + <term> + <term>

5: -> A = <factor> + <term> + <term>

6: -> A = <id> + <term> + <term>

7: -> A = B + <term> + <term>

8: -> A = B + <factor> + <term>
```

```
9: -> A = B + <id> + <term>

10: -> A = B + C + <term>

11: -> A = B + C + <factor>

12: -> A = B + C + <id>

13: -> A = B + C + A

---

Enter a sentence:


A = A * ( B + C )

Sentence:

A = A * ( B + C )

Derivation:

0: <assign> -> <id> = <expr>

1: -> A = <expr>

2: -> A = <term>

3: -> A = <term> * <factor>

4: -> A = <factor> * <factor>

5: -> A = <id> * <factor>

6: -> A = A * <factor>

7: -> A = A * ( <expr> )

8: -> A = A * ( <expr> + <term> )

9: -> A = A * ( <term> + <term> )

10: -> A = A * ( <factor> + <term> )

11: -> A = A * ( <id> + <term> )

12: -> A = A * ( B + <term> )

13: -> A = A * ( B + <factor> )

14: -> A = A * ( B + <id> )
```

```
15: -> A = A * ( B + C )

---

Enter a sentence:


A = B * ( C * ( A + B ) )

Sentence:

A = B * ( C * ( A + B ) )

Derivation:

0: <assign> -> <id> = <expr>

1: -> A = <expr>

2: -> A = <term>

3: -> A = <term> * <factor>

4: -> A = <factor> * <factor>

5: -> A = <id> * <factor>

6: -> A = B * <factor>

7: -> A = B * ( <expr> )

8: -> A = B * ( <term> )

9: -> A = B * ( <term> * <factor> )

10: -> A = B * ( <factor> * <factor> )

11: -> A = B * ( <id> * <factor> )

12: -> A = B * ( C * <factor> )

13: -> A = B * ( C * ( <expr> ) )

14: -> A = B * ( C * ( <expr> + <term> ) )

15: -> A = B * ( C * ( <term> + <term> ) )

16: -> A = B * ( C * ( <factor> + <term> ) )

17: -> A = B * ( C * ( <id> + <term> ) )

18: -> A = B * ( C * ( A + <term> ) )
```

```
19: -> A = B * ( C * ( A + <factor> ) )

20: -> A = B * ( C * ( A + <id> ) )

21: -> A = B * ( C * ( A + B ) )

---

Enter a sentence:
```

Pylint results:

```
-------------------------------------------------------------------
---


Your code has been rated at 10.00/10 (previous run: 9.89/10,
+0.11)
```

Conclusions

Throughout the coding process, I learned many things about Python syntax (such as sub-functions, conditionals within arguments, how spacing/tabs matter), conventions (some of the stuff pylint gave me errors on such as snake case and docstrings, etc.) ,commonly used basic library functions such as format, enumerate, etc. I also learned about parsers, specifically left-most derivation parsing.