CS 6410: Compilers

Fall 2019

HW 3 – LR Construction and LL Grammars

Assigned: Monday, October 4, 2023, Due: Saturday, November 4, 2023
Instructor: Tamara Bonaci
College of Computer and Information Science
Northeastern University – Seattle

Submission Guidelines

- Please submit your homework as a single .pdf file through Canvas.
- You do not have to type in your submission - hand-written and then scanned, or photographed documents are fine, as long as the total size of your document is not too big, and your document is readable.
- This assignment is meant to be worked on individually, and you should submit it by **11:59pm on Saturday, November 4, 2023**.

Problem 1 (Cooper and Torczon, Problem 3.4)

The following grammar is not suitable for a top-down predictive parser. Identify the problem, and correct it by rewriting the grammar. Please show that the new grammar satisfies the LL(1) condition.

```
L ::= R a | Q ba
R ::= aba | caba | R bc
Q ::= bbc | bc
```

Problem 2 (Cooper and Torczon, Problem 3.5)

Consider the following grammar:

```
A ::= B a
B ::= dab | C b
C ::= c B | A c
```

Does this grammar satisfy the LL(1) condition? Please justify your answer. If it does not, please rewrite is as an LL(1) grammar for the same language.

Problem 3 (Cooper and Torczon, Problem 3.7)

Suppose that an elevator is controlled by two commands: ↑ to move the elevator up one floor, and ↓ to move the elevator down one floor. Assume that the building is arbitrarily tall, and that the elevator starts at floor x.
Please write an LL(1) grammar that generates arbitrary command sequence that:

1. Never cause the elevator to go below floor x
2. Always returns the elevator to floor x at the end of the sequence.

For example, ↑↑↓↓ and ↑↓↑↓ are valid command sequences, but ↑↓↓↑ and ↑↓↓ are not. For convenience, you may consider a null sequence as valid. Prove that your grammar is LL(1).

Problem 4

Please write a grammar that generates the straight-line code language given below, but that is suitable for LL(1) parsing. That is, eliminate the ambiguity, eliminate the left recursion, and (if necessary) left-factor.

```
S ::= S ; S
S ::= id := E
S ::= print( L )
E ::= id
E ::= num
E ::= E + E
E ::= ( S, E )
L ::= E
L ::= L , E
```