CSE340 Fall 2019 - Homework 2

Due: Wednesday October 2 2019 by 11:59 PM on Canvas

All submissions should be typed, no exceptions.

1. Consider the grammar.

$S \rightarrow A\ B\ g\ C$	(1)
$A \rightarrow a\ A\ f$	(2)
$A \rightarrow (\ C\ \ B\ )$	(3)
$A \rightarrow\ D$	(4)
$B \rightarrow b\ B\ c$	(5)
$B \rightarrow \varepsilon$	(6)
$C \rightarrow c\ C$	(7)
$C \rightarrow \varepsilon$	(8)
$D \rightarrow d\ D$	(9)
$D \rightarrow \varepsilon$	(10)

where S, A, B, C and D are the non-terminals, S is the start symbol, and a, b , c , d , f , g , (and) are the terminals.

1.1. Calculate the FIRST for all grammar symbols as follows

1.1.1. Do an initialization pass by applying FIRST sets rules I and II.
1.1.2. Do successive passes, on the grammar rules in the order they are listed and apply to each grammar rule FIRST set rules III, then FIRST set rule IV then FIRST set rule V

In your answer, you should show the FIRST sets after each pass. You do not have to use the notation I used in class to indicate the order in which the elements are added to the sets.

1.2. Calculate the FOLLOW sets for all non-terminals as follows

1.2.1. Do an initialization pass by applying FOLLOW sets rule I
1.2.2. Do one pass on all grammar rules, in the order they are listed, and apply to each grammar rule FOLLOW set rules IV and V.
1.2.3. Do successive passes on all grammar rules in the order they are listed and apply to each grammar rule FOLLOW set rules II and III until there is no change.

In your answer, you should show the FOLLOW sets after each pass. You do not have to use the notation I used in class to indicate the order in which the elements are added to the sets.

1.3. Show that the grammar has a predictive recursive descent parser. You should show that the conditions of predictive parsing apply for every non-terminal.

1.4. Write parse_S(), parse_A(), parse_B(), parse_C() and parse_D(). Your parser should follow the general model of predictive parser that we saw in class. In particular, for non-terminals that can generate ε, the parser should check the FOLLOW set before choosing to parse the righthand side that generates ε.

1.5. Give a full execution trace for your parser from part (a) above on input a f g c c d

2. Consider the following operator grammar

   $E \rightarrow E \,\&\, E$
   $E \rightarrow \sim E$
   $E \rightarrow (E) \mid id$

   & is a right associative binary operator. ~ is a unary operator and has lower precedence than &

   2.1. Draw the precedence table for this grammar
   2.2. Show step by step how id ~ id & id & id is parsed

3. **(Lambda calculus binding)**. For each of the following determine for each variable x the λx. it is bound to. I have numbered the variables and the abstractions. The variables are numbered using Arabic numerals and the abstractions are numbered using roman numerals. If variables 4 and 7 are bound to abstraction I, your answer should be of the form I → 4 7 to indicate that abstraction I has variables 4, and 7 bound by it.

```
Example    ( λx.  x  x  λx.  x  ) x
              I   1  2  II   3    4

Answer     I → 1  2
           II → 3
```

   Your answered need not be colored

```
3.1.  ( λx.  x  x  λx.  x  ( λx.  x  x  λx.  x  ) x ) x
         I   1  2  II   3     III 4  5  IV   6    7   8

3.2.  ( ( λx.  x  x  λx.  x  ) x ( λx.  x  x  λx.  x  ) x ) x
           I   1  2  II   3    4   III  5  6  IV   7    8   9

3.3.   λx.  ( λx. λx.  x  x  λx.  x  ( λx.  x  x ) λx.  x  ) x  x
        I      II  III 1  2  IV   3     V   4  5   VI   6    7  8
```

4. (**Beta Reduction**) For each of the following, give the resulting expression after executing one beta reduction. If there is more than one redex in the given expression, you can choose which redex you want to reduce. If there is no reducible expression, you should say so in your answer. If the reduction requires renaming, you should do renaming first. You should show the expression after alpha renaming and then after beta reduction. You should only do renaming as needed no more than needed.

```
4.1.  ( λx.  x  (λx.  x) x )  ( x x ) ( λx.  x  x  )
4.2.   x  ( λx.  λx.  x    x ) x
4.3.  ( λx.  ( λx.  ( λx.  x )  y ) z ) w
4.4.   λz.  ( λx.  ( λy.  z  y ) )  y
4.5.   ( λx.  ( λy.  ( λz.  x  y ) x ) ) y z
4.6.  ( λx.  ( λy.  ( λz.  x  y ) x )  ( y  z )
```