CS3342 – Assignment 2
due Feb. 16, 2023
2-day no-penalty extension until: Feb. 18, 11:55pm

1. (30pt) Consider postfix expressions, which are arithmetic expressions in which the operator comes after the operands: `a + b` is written as `a b +`. Assume `const` operands and `$` end marker.

   (a) (10pt) Write an SLR(1) grammar, G_r, for postfix expressions, that is not LL(1). Give the FIRST and FOLLOW sets for all nonterminals. Show that G_r is not LL(1). Draw the LR parser as a graph; the states contain the LR-items, the transitions are labelled by tokens, reduce states are double circled. Include also (as jflap does) the trivial states, containing a single LR-item with the dot at the end. Build its LR parse table (as done by jflap) to prove it is SLR(1).

   [(1+2) - (3×4)] / 5 + (6×7) .

   (b) (5pt) Draw the parse tree (in G_r) for the string `1 2 3 4 + * - 5 / 6 7 * + $`.

   (c) (10pt) Write an LL(1) grammar, G_ℓ, for postfix expressions. Build its LL parse table (as done by jflap) to prove it is LL(1). Give also the FIRST and FOLLOW sets for all nonterminals.

   (d) (5pt) Draw the parse tree (in G_ℓ) for the string `1 2 3 4 + * - 5 / 6 7 * + $`.

2. (20pt) Scientific notation is expressing a number as $m \times 10^n$, where m, the *mantissa*, is a decimal number and n, the *exponent*, is an integer. Scientific e notation is the same thing written in a single line as m e n. We will assume, for simplicity, that the mantissa must have digits before and after the decimal point and that the exponent must have at least one digit. Examples: `40.5e3`, `8.0e0`, `-23.11e20`, `+0.234e-9`.

   (a) (15pt) Construct an attribute grammar that uses only one attribute, *val*; the value of *val* for the root will store the value corresponding to the number represented by the scientific e notation given by the yield of the tree.

   (b) (5pt) Draw an annotated parse tree for the string `-12.345e-10`. Show the attribute flow (arrows and values).

3. (50pt) Write a Python program, `balance.py`, which computes the minimum number of edit operations to balance a string. The *edit distance* between two string is the minimum number of operations – insertion, deletion, replacement – necessary to transform one string into another. A string consisting only of parentheses, `(` and `)` is called *balanced* if its parentheses can be properly matched: each open parenthesis, `(`, with a following closed parenthesis, `)`.

   The program, on an input string consisting only of parentheses, outputs:

   - the smallest edit distance, d, between the input string and a balanced string and
   - a balanced string at edit distance d from the input.

   For example:

   ```
   $ python balance.py "(()())()" outputs: d = 0, balanced string: "(()())()"
   $ python balance.py ")())" outputs: d = 1, balanced string: "(())"
   $ python balance.py "))())" outputs: d = 2, balanced string: "()()()"
   $ python balance.py ")))((" outputs: d = 3, balanced string: "()()"
   ```

   Note that there can be more than one closest balanced string, and the program is allowed to output any such solution. Also, optimality proof is not required.

fop: 2   countL   ) balanced) balanced ((

countR ++

return opLeft+()+opRight.

```
def optStr(str):
    if hasBalance:
        Left:  Right:
        return optStr(left) + ( ) + optStr(Right).
    else
```

)) ( ) ( ) (( ))) (.

( )

cur   countR

))) (((

=> 4 ( ) ( ) ( )

))) ( ) (((

=> ( ) (( )) ( ) .

(( ) ( ) ( ) ( )

(: count L += 1

+= 1 . . . until)

if count L

((((( ) ( ) ( ) .

((( ) ( )) ((( )) .

)) ( )) (((

)) (( ) ( ) ( )

( ) ( ) ( ) ( )))) (

))   ( )
((( )
( )   ( )

count R = ~~3~~ 2.

))) (

mn

)))) (

-2

count R = 4.

(( ) ( )) ( ) .

L 1 2 1 2 1
R
Str (( ) ( )

balanced? yes - getnext
no
rmBalance.

Simplist Str Balance
unbalance:) at beginning.
too many

return

))
((
)(

))) ((odd '('
)))(((odd ')'

AA~~A~~BB => rm 'A'
=> 'A' => 'B'   'B' => 'A'.

even num =>
BB~~B~~AAA
AB AB AB.

:7   count 'B'/2 + count 'A'/2 + numOf OddGroup.

BBBBAAAA
ABABABAB.
count 'B'/2 + count 'A'/2

READ ME! Submit your answers as a *single pdf file* in OWL. Solutions should be typed; readable (by others!) hand-written solutions are also acceptable. Source code, if required, is submitted as separate files.

JFLAP: You are allowed to use JFLAP to help you solve the assignment. Make sure you understand what it does; JFLAP will not be available during in-person exams!

LaTeX: For those interested, the best (the only!) program for scientific writing is LaTeX. It is free and you can start using it in minutes: `https://tobi.oetiker.ch/lshort/lshort.pdf`

1. (30pt) Consider postfix expressions, which are arithmetic expressions in which the operator comes after the operands: `a + b` is written as `a b +`. Assume `const` operands and `$` end marker.

   (a) (10pt) Write an SLR(1) grammar, G_r, for postfix expressions, that is not LL(1). (1) Give the FIRST and FOLLOW sets for all nonterminals. (2) Show that G_r is not LL(1). (3) Draw the LR parser as a graph; the states contain the LR-items, the transitions are labelled by tokens, reduce states are double circled. (4) Include also (as jflap does) the trivial states, containing a single LR-item with the dot at the end. Build its LR parse table (as done by jflap) to prove it is SLR(1).

   [(1+2) - (3×4)] / 5 + (6×7).

   (b) (5pt) Draw the parse tree (in G_r) for the string `1 2 3 4 + * - 5 / 6 7 * + $`.

   (c) (10pt) Write an LL(1) grammar, G_ℓ, for postfix expressions. Build its LL parse table (as done by jflap) to prove it is LL(1). Give also the FIRST and FOLLOW sets for all nonterminals.

   (d) (5pt) Draw the parse tree (in G_ℓ) for the string `1 2 3 4 + * - 5 / 6 7 * + $`.

1A A T B.S

(a) S' -> S

S -> T S

S -> $

T -> A A T B.

S -> num   S -> num sign.

Sign -> Sign num.sign

Sign -> num num sign

Sign -> sign sign sign.

Sign -> $

1 2 3 4 + * - 5 / 6 7 * + $

SL(4)

LL/LR(1)

S' -> S

S -> num

S -> num num T sign

T -> num num T sign.

P num sign -> P

$A \to B\,C$ ⊆ CEG

$A \to BD$.

← SLL

2212

$P \to P$ sign

$T \to$ num

$T \to$ num num T sign

$P \to N_2 S$

$P \to N_2 P S S$

$N_2 \to$ num num

$S \to$ sign

$P \to P N_1 S$

$N_1 \to$ num

$P \to \$$

$N = 1 \quad S = 0$

$N = 2 \quad S = 1$

$N = 3 \quad S = 2$

$N = 4 \quad S = 3$.