**CSCI 465**

**Exam I**

**Due October 25, 2020**

**Take Home**

**(Total Point: 120)**

# First Name\_\_\_\_Derek\_\_\_\_\_\_\_\_\_\_\_

Last Name\_\_\_\_\_Trom\_\_\_\_\_\_\_\_\_\_\_

1. (10 points) Regular expressions are closed under many operations--- that is, if we apply these operations to an RE or a collection of REs, the result is an RE.

1. What are these operations?
   1. Union, Concatenation, Kleene Closure, Positive Closure
2. What are the main incentives to use these operations?
   1. The main incentives to use these operations is that we are able to create sets of rules for names etc. But also through the use of these operations we are able to combine and use the smaller sets to make larger sets of rules. These rules can be combined to describe a full languages rules.
3. Can you provide one example for each operation?
   1. Union(∪)
      1. The union of Letters and Digits in a language as an example would be the set of all 62 letters and digits 0-9 and a-zA-Z.
   2. Concatenation
      1. (0 + 1) 0
      2. This would concatenate to {00,10}
   3. Kleen Closure or Star Closure(\*)
      1. If we have say a\*
      2. This corresponds to any string of a’s
      3. {∈, a, aa, aaa, aaaa, aaaaa….an}
   4. Positive Closure(+)
      1. If we have ab+
      2. The set would be {ab, abb, abbb, abbbb,…abn>0}
4. (20 points) Transform the following (recursive) grammar into Iteration (i.e., {} or \*)

(a) E→ T | E addop T

addop→+ | -

R = T({+|-}T)\*

(b) RepeatStmt → **Repeat** Stmt Rest **Until** expr

Rest → ; Stmt Rest | ∈

expr→boolean

**Repeat** Stmt (;Stmt)\* **Until** (T+F)

1. (30 points) Consider the following grammar

S→A d

A→c |a A B

B→ b B |∈

1. Find FIRST(B) and Follow(B)
2. FIRST(B)
   1. = FIRST(b)FIRST(B) | ∈
   2. ={b, ∈)
3. Follow(B)
   1. Find B’s on RHS = 2
   2. Add symbols to right of B = {}
   3. Find follow B and follow A
      1. B={}
      2. FOLLOW(A) = {d}∪FIRST(b)=
   4. Follow(B) = { b, ∈,d)
4. Why is it harder to compute FOLLOW sets?
   1. It is harder to compute follow sets because the follow sets also depend on other follow sets being calculated if they are non-terminals to the left of the Follow symbol. The follow sets also may have to depend on the first sets of some terminal and non-terminal symbols in order to find the follow sets of the original symbol.
5. Why do we need to Compute FIRST and FOLLOW sets?
   1. We need to compute these sets in order to select the appropriate production rules for recursive parsing as well as to fill in entries of a parsing table for non-recursive predictive parsing table.
6. (10 points) Remove any issue from the following grammar so it can work with LL(1) parser.

expr → expr addop term| term

term→ ID | ID ‘(‘ expr ‘)’

addop→ + |-

* + 1. Eliminate immediate left recursion
       1. Expr → term expr’
       2. Expr’→ addop term expr’ | ε
       3. term→ ID | ID ‘(‘ expr ‘)’
       4. addop→ + |-
    2. Left factor
       1. Expr → term expr’
       2. Expr’→ addop term expr’ | ε
       3. term→ ID A
       4. A → ‘(‘ expr ‘)’| ε
       5. addop→ + |-

1. (20 points) Consider the following grammar

B→0B1

B→∈

(a). Transform the above grammar to PDA

* + 1. T(q,∈, B) = (q, 0B1) < apply rule 2 for nonterminal>
    2. T(q,∈, B) = (q, ∈) <apply rule 2 for nonterminal>
    3. T(q, 0, 0) = (q, ∈) < apply rule 1 for terminal>
    4. T(q, 1, 1) = (q, ∈) < apply rule 1 for terminal>

(b). Parse the input **00110** with the initial configuration **(q, 00110, B)**

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Input | Stack | Transition |
| 1 | .00110 | B | No Move |
| 2 | .00110 | 0B1 | 1 |
| 3 | 0.0110 | B1 | 3 |
| 4 | 0.0110 | 0B11 | 1 |
| 5 | 00.110 | B11 | 3 |
| 6 | 00.110 | 11 | 2 |
| 7 | 001.10 | 1 | 4 |
| 8 | 0011.0 | EMPTY | 4 |
| 9 | Not | Accepted |  |

1. (30 points) consider the following **syntax-directed definition** which can be useful in converting from binary to decimal

(**Note**: L and frac represent strings of binary digits):

|  |  |
| --- | --- |
| Production | Semantic Rules |
| S → L. frac | S.val = L.val + frac.val |
| L → L1 **0** | L.val = 2\*L1.val |
| L → L1 **1** | L.val = 2\*L1.val + 1 |
| L →∈ | L.val = 0 |
| frac→ ∈ | frac.val = 0 |
| frac→ **0** frac1 | frac.val = frac1.val /2 |
| frac→ **1** frac1 | frac.val = 0.5 + (frac1.val/2) |

DRAW annotated parse tree based on the above syntax directed definition table. Using annotated tree with synthesized value **val**, show how to convert each of the following binary numbers (i.e., input) to the corresponding decimal number (i.e., output).

1. 1010.2 = 1010

1. 1.0112= 1.37510