CS 473 Spring 2020 Midterm 1

Name: 
NetID: 

- You have **50 minutes** to complete this exam.
- This is a **closed-book, closed-notes** exam.
- Do not share anything with other students. Do not talk to other students. Do not look other students’ exams. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- If you believe there is an error or an ambiguous question, you may seek clarification from the instructor. Please speak quietly or write your question out.
- Including this cover sheet, there are 10 pages to the exam, including one blank page for workspace. Please verify that you have all 10 pages.
- Please write your name and NetID in the spaces above.
- Show your work. Partial credit will be given for incomplete answers.
- If you finish with time remaining, check your work!
<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1. (20 points)
Write regular expressions for each of the following languages, or explain why no such regular expression exists.

(a) (5 points) Strings over the alphabet \{a, b\} that start with \(a\) and have odd length.

Solution: \(a((a|b)(a|b))^*\)

(b) (5 points) Strings over the alphabet \{0, 1\} with at least two consecutive 0’s.

Solution: \((0|1)^*00(0|1)^*\)

(c) (5 points) Strings over the alphabet \{a, b, c\} in which every \(a\) is immediately followed by at least two \(b\)’s.

Solution: \((b|c|abb)^*\)

(d) (5 points) Strings over the alphabet \{0, 1\} that start and end with different symbols.

Solution: \((0(0|1)^*1)|(1(0|1)^*0)\)

Problem 2. (20 points)
Write context-free grammars for each of the following languages.

(a) (5 points) Strings containing zero or more \(b\)’s followed by one or more \(a\)’s.

Solution:

\[
S \rightarrow bS \mid A \\
A \rightarrow aA \mid a
\]
(c) (5 points) Arithmetic expressions with + and * operators such that * has higher precedence (i.e., binds tighter) than + and both + and * are left-associative. You can use the symbol number to represent numbers.

Solution:

\[ S \rightarrow S + A \mid A \\
A \rightarrow A * B \mid B \\
B \rightarrow \text{number} \]

(d) (5 points) Arithmetic expressions in prefix notation, where an expression consists of either a number, an operator (one of +, -, *, and /) followed by two more expressions, or an expression in parentheses.

Solution:

\[ S \rightarrow \text{number} \mid O \ S \ S \mid ( \ S \ ) \\
O \rightarrow + \mid - \mid * \mid / \]
Problem 3. (25 points)
Consider the following grammar, with terminal symbols let, in, ident, :=, and $:

\[ T \rightarrow S$ \\
\[ S \rightarrow D \text{ in } E \\
\[ D \rightarrow V D \mid \varepsilon \\
\[ V \rightarrow \text{ident} := E \mid \text{let ident} := E \\
\[ E \rightarrow \text{ident} \mid \varepsilon \\

(a) (10 points) Write the FIRST and FOLLOW sets for the grammar.

<table>
<thead>
<tr>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T \rightarrow S$</td>
<td>ident, let, in</td>
</tr>
<tr>
<td>$S \rightarrow D \text{ in } E$</td>
<td>ident, let, in</td>
</tr>
<tr>
<td>$D \rightarrow V D$</td>
<td>ident, let</td>
</tr>
<tr>
<td>$V \rightarrow \text{ident} := E$</td>
<td>ident</td>
</tr>
<tr>
<td>$V \rightarrow \text{let ident} := E$</td>
<td>let</td>
</tr>
<tr>
<td>$E \rightarrow \text{ident}$</td>
<td>ident</td>
</tr>
<tr>
<td>$E \rightarrow \varepsilon$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

or

<table>
<thead>
<tr>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>ident, let, in</td>
</tr>
<tr>
<td>$S$</td>
<td>ident, let, in</td>
</tr>
<tr>
<td>$D$</td>
<td>ident, let</td>
</tr>
<tr>
<td>$V$</td>
<td>ident, let</td>
</tr>
<tr>
<td>$E$</td>
<td>ident</td>
</tr>
</tbody>
</table>

Ident, let, in, $\varepsilon$
(b) (15 points) Fill in the LL(1) parse table for the grammar.

Solution:

<table>
<thead>
<tr>
<th></th>
<th>let</th>
<th>in</th>
<th>ident</th>
<th>:=</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T → S$</td>
<td>T → S$</td>
<td>T → S$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>S → D in E</td>
<td>S → D in E</td>
<td>S → D in E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>D → V D</td>
<td>D → ε</td>
<td>D → V D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>V → let ident := E</td>
<td>V → ident := E</td>
<td>V → ident := E</td>
<td>V → ε</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E → ε</td>
<td>E → ε</td>
<td>E → ident,</td>
<td>E → ε</td>
<td>E → ε</td>
</tr>
</tbody>
</table>
Problem 4. (20 points)
Consider the following grammar, with terminal symbols [, ], ident, and number:

\[
T \rightarrow E$
\]
\[
E \rightarrow E \ [ \ E \ ] \ | \ ident \ | \ number
\]

Write the start state of the LR(0) parser for this grammar, including all its items, its transition edges, and the items in each of its successor states.

Solution:

\[
\begin{array}{c}
T \rightarrow E$
\end{array}
\]
\[
\begin{array}{c}
E \rightarrow E \ [ \ E \ ]
\end{array}
\]
\[
\begin{array}{c}
E \rightarrow ident.
\end{array}
\]
\[
\begin{array}{c}
E \rightarrow number.
\end{array}
\]
Problem 5. (15 points)
Suppose you were writing a type checker for a language with the following grammar:

\[ E \rightarrow \text{number} \mid \text{ident} \mid E + E \]

The type of ASTs for the language is defined as:

```c
typedef struct A_exp{
    enum {E_num, E_id, E_plus} kind;
    union {int ival;
            char *name;
            struct {struct A_exp *lhs; struct A_exp *rhs;} operands;} u;
} *A_exp;
```

Fill in the code for the `E_plus` case of the type checker, where addition and subtraction take two operands of type `int` and return a result of type `int`. If the assignment has a type error, return the value `Ty_err`.

```c
Ty typecheck(A_exp e){
    switch(e->kind){
        case E_num:
            return Ty_int;
        case E_true:
            return Ty_bool;
        case E_false:
            return Ty_bool;
        case E_plus:
            break;
    }

    return Ty_err;
}
```

Solution:

```c
Ty typecheck(A_exp e){
    switch(e->kind){
        case E_num:
            return Ty_int;
        case E_true:
            return Ty_bool;
        case E_false:
            return Ty_bool;
        case E_plus:
            break;
    }

    return Ty_err;
}
```
Ty ty1 = typecheck(e->u.operands.lhs);
Ty ty2 = typecheck(e->u.operands.rhs);
if(ty1 == ty2 && ty2 == Ty_int) return Ty_int;
else return Ty_err;
}
1 Scratch Space