CS143 Midterm
Spring 2018

- Please read all instructions (including these) carefully.
- There are 6 questions on the exam, some with multiple parts. You have 80 minutes to work on the exam.
- The exam is open note. You may use laptops, phones and e-readers to read electronic notes, but not for computation or access to the internet for any reason.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. Do not write on the back of exam pages or other pages.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

NAME: _____

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: _____

Problem	Max points	Points
1	10	
2	15	
3	10	
4	10	
5	20	
6	15	
TOTAL	80	

1. **Regular Expressions and Context Free Grammars** (10 points)
We discussed in class that regular expressions are the "weakest" of the formal languages we deal with. One way of making this concrete is by saying that they have the least expressive power—all regular languages can be described by a context free grammar, but the converse is not true. In class we also gave a systematic way for building an NFA from a regular expression by defining a translation for 5 core regular expressions (ϵ, character 'c', concatenation, union, Kleene star).

   Give a similar translation from any regular expression over an alphabet Σ to a corresponding CFG. That is, for any regular expression R, your translation should construct a CFG G such that the languages of R and G are equal. Specify your translation by filling in the right-hand sides of the productions below; the subscript on each non-terminal indicates the kind of regular expression for that production. If you need to write multiple productions for a non-terminal use the vertical bar "|" notation to separate multiple right-hand sides.

$$
\begin{aligned}
S_{\epsilon} &\rightarrow \\
S_{c \in \Sigma} &\rightarrow \\
S_{AB} &\rightarrow \\
S_{A+B} &\rightarrow \\
S_{A^*} &\rightarrow
\end{aligned}
$$

$$
\begin{aligned}
S_{\epsilon} &\rightarrow \epsilon \\
S_{c \in \Sigma} &\rightarrow c \\
S_{AB} &\rightarrow S_A S_B \\
S_{A+B} &\rightarrow S_A \mid S_B \\
S_{A^*} &\rightarrow S_{A^*} S_A \mid \epsilon
\end{aligned}
$$

2. **Top-Down Parsing** (15 points)
Consider the following grammar, where A is the start symbol:

$$\begin{aligned} A &\rightarrow BwA \\ A &\rightarrow \epsilon \\ B &\rightarrow CxB \\ B &\rightarrow yC \\ C &\rightarrow z \end{aligned}$$

In this grammar A, B and C are non-terminals and w, x, y, and z are terminals. Prove or disprove: This grammar is LL(1).

There is nothing obvious that would disqualify the grammar from being LL(1) immediately: it isn't left recursive, it is left factored already, and it is not obviously ambiguous. So the only way to definitively answer the question is to build the LL(1) parsing table and check for conflicts. Below we show the grammar is LL(1), as the LL(1) parsing table has at most one production per entry.

```
First(C) = {z}
First(B) = First(C) + y = {y,z}
First(A) = First(B) = {y,z} + \epsilon = {y,z,\epsilon}

Follow(A) = { $ }
Follow(B) = { w }
Follow(C) = { x,w }
```

	w	x	y	z	\$
A			BwA	BwA	ϵ
B			yC	CxB	
C				z	

3. **Regular Languages and Finite Automata** (10 points)

Let L be a language over the alphabet $\Sigma = \{2, 7, +\}$ consisting of strings representing valid arithmetic expressions in base-10 that evaluate to an odd number. Numbers can have multiple digits.

Examples of strings in L:

$$27 \qquad 22 + 77 + 72 \qquad 2727 + 772$$

Examples of strings **not** in L:

$$77 + 727 \qquad 22 \qquad 727 + 22 + +72$$

(a) Give an intuitive explanation as to why L is a regular language. As a hint, when reading in a valid arithmetic expression s, what do we need to keep track of when determining whether s is in L?

We need only track parity (that is, whether the number is odd or even) of the running sum seen so far. This tracking can be done by applying rules of arithmetic as we scan from left to right (e.g., an even number plus an even number is an even number). Since we use a constant amount of memory independent of the expression length, intuitively L should be regular.

(b) One way to write a regular expression for L is:

$$(E+)^*O((+E)^* + O(+E)^* + O)^*(+E)^*$$

where E is a regular expression for the set of all strings over Σ that are even numbers and O is a regular expression for the set of all strings over Σ that are odd numbers. Give definitions for E and O (i.e., give regular expressions for each of them).

$$E = (2|7)^*2$$
$$O = (2|7)^*7$$

4. **Lexical Analysis** (10 points)
We are writing a `flex` lexer for a restricted class of C-style comments:

- Comments begin with `/*` and end with `*/`. No nesting is allowed; the first `*/` after a `/*` closes the comment.
- Comments are all on one line—they never contain newlines. There is no need to keep track of the current line.

Below are two different, correct implementations for the lexer. Answer the questions below each one by circling "yes" or "no". THE SOLUTION IS THE CAPITALIZED OPTION.

(a) Implementation #1

```
1.      /\*       { BEGIN(A) }
2.       <A>\*/   { BEGIN(INITIAL) }
3.       <A>.     {  }
```

Will the lexer work properly if...

...we swap lines 2 and 3? YES no

(b) Implementation #2

```
1.       /\*          { BEGIN(A) }
2.       <A>\*        { BEGIN(B) }
3.       <B>/         { BEGIN(INITIAL) }
4.       <B>.         { BEGIN(A) }
5.       <A>[^\*]     { }
```

Will the lexer work properly if...

...we swap lines 2 and 3? YES no
...we swap lines 2 and 4? yes NO
...we swap lines 2 and 5? YES no

5. **Bottom Up Parsing** (20 points)

(a) Give a grammar over the alphabet $\{a\}$ that has one shift-reduce conflict and no other conflicts under SLR(1) parsing rules. There are short answers to this question, with a small number of productions, terminals, and short right-hand sides.

Show the deterministic parsing automaton for your grammar, including the extra production that is introduced as part of the automaton's construction.

```
S -> aSa | a

Automaton start state:
S' -> .S
S  -> .aSa
S  -> .a
Transition to 1 on S
Transition to 2 on a

State 1:
S' -> S.

State 2:
S -> a.Sa
S -> a.    <- Shift reduce conflict as "a" is in Follow(S)
S -> .aSa
S -> .a
Transition to 2 on a
Transition to 3 on S

State 3
S -> aS.a
Transition to 4 on a

State 4
S -> aSa.
```

(b) Give a grammar over the alphabet $\{a\}$ that has one reduce-reduce conflict and no other conflicts under SLR(1) parsing rules. There are short answers to this question, with a small number of productions, terminals, and short right-hand sides.

Show the deterministic parsing automaton for your grammar, including the extra production that is introduced as part of the automaton's construction.

```
S -> a
S -> A
A -> a


Automaton start state:
S' -> .S
S  -> .a
S  -> .A
A  -> .a
Transition to 1 on S
Transition to 2 on a
Transition to 3 on A

State 1:
S' -> S.

State 2:
S -> a.
A -> a.  --> Reduce/Reduce conflict as $ is in the Follow of A and S

State 3:
S -> A.
```

6. **Syntax-Directed Translation** (15 points)

Consider the binary numbers over $\{0, 1\}$. Give a syntax directed translation (i.e., a context free grammar and associated actions) that assigns an attribute of the root of the parse tree the value of the binary number converted to base 10. For example, for the string 1001 attribute of the root of the parse tree should be assigned the value 9.

The following solution uses bison's action notation, but this was not required for full credit; any attribute notation we could understand was accepted.

```
S -> 1      { $$ = 1 }

S -> 0      { $$ = 0 }

S -> S1     { $$ = 2 * $1 + 1 }

S -> S0     { $$ = 2 * $1 }
```