CS143 Midterm
Spring 2016

- Please read all instructions (including these) carefully.
- There are 5 questions on the exam, all 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	20	
3	10	
4	20	
5	20	
TOTAL	80	

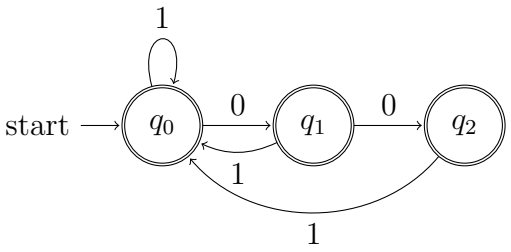
1. **Regular Languages and DFAs** (10 points)

(a) Write a regular expression for the set of strings over $\{0, 1\}$ that do not contain a sequence of 3 (or more) zeros consecutively, anywhere in the string. Examples:

- In the set of strings: 0, 100111111101, 0100100
- Not in the set of strings: 000, 1101000001

$(0?\,0?\,1)^*\,0?\,0?$

(b) Give a DFA that recognizes the same set of strings. Your DFA need not be complete (it need not have transitions for all inputs in all states).



2. **Top-Down Parsing** (20 points)

Consider the following grammar (note each production is numbered):

$$
\begin{array}{rcll}
S & \rightarrow & \text{IF } (E) \text{ THEN } (E)\ X & (1) \\
 & | & E & (2) \\
X & \rightarrow & \epsilon & (3) \\
 & | & \text{ELSE } (E) & (4) \\
E & \rightarrow & \text{TRUE } Y & (5) \\
 & | & \text{FALSE } Y & (6) \\
Y & \rightarrow & \epsilon & (7) \\
 & | & \text{AND } (E) & (8)
\end{array}
$$

S is the start symbol. The terminals are IF, THEN, ELSE, AND, TRUE, FALSE, (and). Now, answer the following:

(a) Compute the first sets for all non-terminals and follow sets for all symbols of the grammar. Write your answers below.

$$
\begin{array}{rcl@{\qquad}rcl}
\text{FIRST}(S) & = & \text{If, True, False} & \text{FOLLOW}(S) & = & \$ \\
\text{FIRST}(X) & = & \epsilon, \text{ Else} & \text{FOLLOW}(X) & = & \$ \\
\text{FIRST}(E) & = & \text{True, False} & \text{FOLLOW}(E) & = &),\$ \\
\text{FIRST}(Y) & = & \epsilon, \text{ And} & \text{FOLLOW}(Y) & = &),\$ \\
\text{FOLLOW(IF)} & = & (& \text{FOLLOW(THEN)} & = & (\\
\text{FOLLOW(ELSE)} & = & (& \text{FOLLOW(AND)} & = & (\\
\text{FOLLOW(TRUE)} & = & \text{And,), \$} & \text{FOLLOW(FALSE)} & = & \text{And,), \$} \\
\text{FOLLOW}('(') & = & \text{True, False} & \text{FOLLOW}(')') & = & \text{Then, Else,), \$}
\end{array}
$$

(b) Fill in the LL(1) parsing table for the grammar. Put only the production number in the table, not the right-hand side.

	(	)	IF	THEN	ELSE	TRUE	FALSE	AND	$
S			1			2	2		
X					4				3
E						5	6		
Y		7						8	7

3. **Grammars** (10 points)

(a) Give a context-free grammar for the set of all strings over the alphabet $\{a, b, c\}$ that are palindromes. (A *palindrome* is a string that reads the same forwards as backwards: the empty string, *aa* and *abccba* are all examples of palindromes.)

$$
\begin{array}{rcl}
S & \rightarrow & \texttt{a}\ S\ \texttt{a} \\
 & | & \texttt{b}\ S\ \texttt{b} \\
 & | & \texttt{c}\ S\ \texttt{c} \\
 & | & \texttt{a} \\
 & | & \texttt{b} \\
 & | & \texttt{c} \\
 & | & \epsilon
\end{array}
$$

(b) Consider the following grammar:

$$
\begin{array}{rcl}
S & \rightarrow & S\ \texttt{and}\ S \\
 & | & S\ \texttt{or}\ S \\
 & | & T \\
 & | & \texttt{a} \\
T & \rightarrow & \texttt{a}
\end{array}
$$

In this grammar S and T are the non-terminals and S is the start symbol; `and`, `or`, and `a` are terminal symbols. How many parse trees are there for the string: `a and a or a`? Justify your answer.

There are 16 possible parse trees given the above grammar.

First, either `and` can have higher precedence than `or` or `or` can have higher precedence than `and`. This gives two possible parse trees to produce the `and` and `or`.

What remains is the three S non-terminals. Each S non-terminal produce the terminal `a` via the production $S \rightarrow$ `a` or the production $S \rightarrow T \rightarrow$ `a`. Each S non-terminal can choose a production independently from the others so there are $2^3 = 8$ parses.

In total, we have $2 * 2^3 = 16$ possible parse trees.

4. **Bottom-Up Parsing** (20 points)

Consider the following grammar:

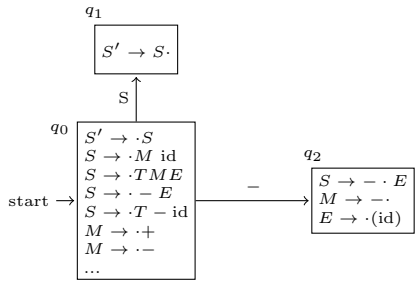
$$\begin{array}{rcl} S & \rightarrow & M \text{ id} \mid TME \mid -E \mid T - \text{id} \\ T & \rightarrow & E \times T \mid E \\ E & \rightarrow & (\text{ id }) \\ M & \rightarrow & + \mid - \end{array}$$

Note that the terminals are $\{\text{id}, +, -, \times, (,)\}$. For your convenience, here are the first and follow sets of the non-terminals.

$$\begin{array}{rclrcl} \text{FIRST}(S) & = & \{+, -, (\} & \text{FOLLOW}(S) & = & \{\$\} \\ \text{FIRST}(T) & = & \{(\} & \text{FOLLOW}(T) & = & \{+, -\} \\ \text{FIRST}(E) & = & \{(\} & \text{FOLLOW}(E) & = & \{\$, \times, +, -\} \\ \text{FIRST}(M) & = & \{+, -\} & \text{FOLLOW}(M) & = & \{\text{id}, (\} \end{array}$$

Is this grammar SLR(1)? Justify your answer.

The grammar is not SLR(1). We can show this by highlighting a shift-reduce conflict, which can be found while trying to construct the LR(0) automaton for this grammar. We add the extra production $S' \rightarrow S$ and give the following partial LR(0) automaton:



State q_2 has a potential shift-reduce conflict, between shifting '(' due to item $E \rightarrow \cdot(\text{id})$ and reducing $M \rightarrow -\cdot$. This is an actual conflict iff '(' is in the FOLLOW set of M. We can see above that this is indeed the case, which means we have found a shift-reduce conflict according to SLR(1) rules.

(More space for question 4.)

5. **Semantic Actions** (20 points)

In Python, reading from a variable before it has been assigned is an error. In this problem you will construct an undefined-variable checker for a small Python-like language using semantic actions. For full credit, your semantic actions should do all of the following (you may find it helpful to look at the grammar on the next page):

- If a statement `S` contains an expression `E`, and `E` references a variable that may be undefined before reaching `S`, print the error message "`A variable may be undefined`". You need not print out which variable (or variables) is undefined.
- If `v` is defined before a statement `S`, then `v` is defined after `S`. This rule always applies in addition to subsequent rules below.
- Variable `v` is defined after the statement `v = E`.
- A variable defined inside an `if` is defined after the `if` when it is defined in both branches.
- In a statement sequence $\texttt{S}_1$; $\texttt{S}_2$, variables defined after $\texttt{S}_1$ are defined before $\texttt{S}_2$.

Your solution should include the following attributes:

- `var.name` is a string containing the variable's name (defined by the lexer).
- `expr.refd` is the set of variables referenced inside the expression. This is an attribute you must compute.
- `stmt.in_defs` is the set of variables defined at the beginning of the statement. This is an attribute you must compute.
- `stmt.out_defs` is the set of variables defined at the end of the statement. This is an attribute you must compute.

Fill in the semantic actions in the alloted space. You should operate on the set attributes using standard set operations (e.g., union, intersection, membership, etc.). Do not be concerned about the order of attribute evaluation—assume the parser can figure out a correct order in which to evaluate your assignments to attributes.

Use bison syntax for the attributes: `$i.[attr]` refers to an attribute of the i-th symbol of the production, `$$.[attr]` refers to an attribute of the production's result. Otherwise use any reasonable and clear programming notation.

```
stmt : var '=' expr
       {
           if ($3.refd ⊈ $$.in_defs)
               print("A variable may be undefined");
           $$.out_defs = $$.in_defs ∪ {$1.name};
       }

     | stmt ';' stmt
       {
           $1.in_defs = $$.in_defs;
           $3.in_defs = $1.out_defs;
           $$.out_defs = $3.out_defs;
       }

     | 'if' expr 'then' stmt 'else' stmt 'fi'
       {
           if ($2.refd ⊈ $$.in_defs)
               print("A variable may be undefined");
           $4.in_defs = $$.in_defs;
           $6.in_defs = $$.in_defs;
           $$.out_defs = $4.out_defs ∩ $6.out_defs;
       }

expr : expr '+' expr
       {
           $$.refd = $1.refd ∪ $3.refd;
       }

     | expr '<' expr
       {
           $$.refd = $1.refd ∪ $3.refd;
       }

     | var
       {
           $$.refd = {$1.name};
       }

     | int_const
       {
           $$.refd = ∅;
       }
```

This scratch page intentionally left blank!