CS 340, Fall 2015 — Dec 17th/19th — Final Name: _____

Note: in all questions, the special symbol ϵ (epsilon) is used to indicate the empty string.

Question 1. [5 points] Consider the following regular expression:

ab(cc|bc)*a(cc)*

For each of the following strings, circle the string if it is in the language generated by the regular expression, or cross it out if it is not in the language.

- ϵ
- **ab**
- **aba**
- **abccacc**
- **abbca**
- **abccbc**
- **abbccc**
- **abacc**
- **abcacc**
- **abbccca**

Question 2. [5 points] Write a regular expression that generates the language of all strings containing an even number of **a**'s and an odd number of **b**'s.

Examples of strings in the language:

- b
- bbb
- aab
- aba
- ababbaa

Examples of strings not in the language:

- ϵ
- ab
- aa
- bbab
- baab

Question 3. [5 points] Write a regular expression that generates the language of all strings containing symbols from the alphabet {**a**, **b**, **c**} not containing the substring **bb**.

Examples of strings in the language:

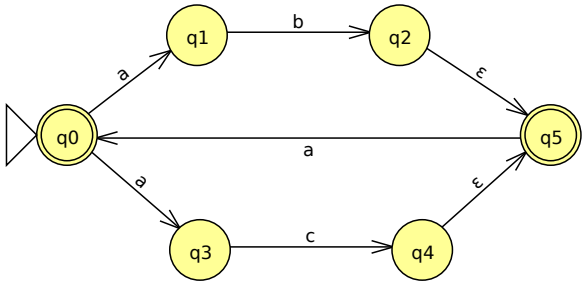
- ϵ
- a
- b
- ab
- cbabaaccb

Examples of strings not in the language:

- bb
- bbc
- abbbac
- bba
- acabba

Question 4. [5 points] Specify a *deterministic* finite automaton (DFA) that recognizes the language described in Question 3. Make sure that you clearly indicate the start state and final state(s), and the direction and symbol for each transition.

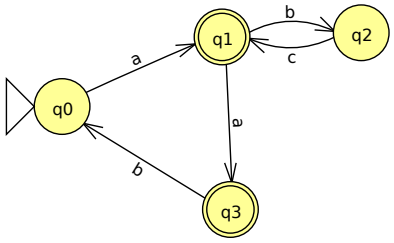
Question 5. [10 points] Consider the following nondeterministic finite automaton (NFA):



(a) What feature or features make this automaton nondeterministic?

(b) Convert this NFA to a *deterministic* finite automaton (DFA). Show the table mapping each reachable set of NFA states to the corresponding DFA state.

Question 6. [5 points] Consider the following finite automaton:



Show how to modify this finite automaton so that there is a single final state, without changing the language that the automaton recognizes. Hint: the modified automaton does not need to be deterministic.

Question 7. [5 points] Specify a context-free language (CFG) that generates the language of all strings containing symbols from the alphabet {**a**, **b**, **c**} containing the same number of **a**'s and **b**'s.

Examples of strings in the language:

- ϵ
- c
- ab
- ba
- abcab

Examples of strings not in the language:

- bba
- bc
- aba
- acaba

In your CFG, use capital letters to represent nonterminal symbols. Make sure that each production has a single nonterminal symbol on the left-hand side. Indicate which nonterminal symbol is the start symbol.

Question 8. [10 points] Consider the following context-free grammar (CFG), which uses nonterminal symbols V L M and terminal symbols **( ) a b**, where V is the start symbol:

V → **(** L **)**	L → M
V → **a**	L → ϵ
V → **b**	M → V M
	M → V

(a) Show a derivation for the string **(a (b a))**. Continue on the right-hand side if necessary.

String	Production	String	Production
V			

(b) Draw the parse tree for the derivation you found in part (a). Make sure that each node represents a single nonterminal or terminal symbol.

Programming Questions

First things first: the following web page specifies which resources you may use during the exam, and links to the permitted resources:

`http://ycpcs.github.io/cs340-fall2015/assign/final.html`

Start by downloading the exam zipfile using the command

`wget` *zipfileURL*

where *zipfileURL* is the URL of the exam zipfile.

Import the zipfile as an Eclipse project. The project will be called `cs340-final`. You will be editing the file `src/cs340_final/core.clj`.

There are four functions to complete: **frobnicate**, **make-doubler**, **combine-pairs**, and **weighted-average**. Each has a detailed comment explaining the requirements, expected behavior, and a point value.

Make sure you meet all of the requirements of each function: e.g., if the function is required to be tail recursive, make sure your function is tail recursive.

Some general hints:

- `(empty? s)` tests whether sequence `s` is empty
- `(first s)` gets the first element in sequence `s`
- `(second s)` gets the second element in sequence `s`
- `(rest s)` gets a sequence with all but the first element of sequence `s`
- `[]` is an empty vector
- `(conj v elt)` creates a vector which results from appending `elt` onto the vector `v`
- `(conj v elt1 elt2)` returns a vector which results from appending `elt1` and `elt2` onto the vector `v`
- `(concat s1 s2)` returns a sequence containing all of the elements of sequence `s1` followed by all of the elements of sequence `s2`

You can run the command `lein test` in a terminal window to run unit tests.

When you are done, submit your code using the blue up arrow icon (Submit project) in Eclipse, or run the command `make submit`. Type your Marmoset username and password when prompted.

Most importantly:

Have fun!