Exam rules. The exam is open book and open notes— you can use any printed or handwritten material. However, no electronic devices are allowed. Anything with an on-off switch must be turned off. There is one exception: you can use a music player as long as nobody else can hear it and you just put it on shuffle and do not use its controls during the exam.

About the problems. These problems are similar to your homework problems. The same conventions about alphabets, etc. that were used on the homework will apply here.

Scoring. Ten problems, 10 points each, total 100 points.

1. Give a regular expression matching exactly those strings accepted by the following NFA:

   ![NFA Diagram](image)

   Answer: $a \mid b \mid aa \mid aa^*ba$. There are also other correct answers, of course.
2. Convert the following NFA to an equivalent DFA:

```
Answer:
```

![Diagram of NFA](image)

![Diagram of DFA](image)
3. Which of the following languages are regular? Circle the letter of each regular language (more than one is regular). In each example, “valid” just means, “has the right form for”, e.g. a valid U.S. phone number has a certain form involving digits, parentheses, and dashes, regardless of whether it actually connects to a working telephone; a valid credit card number satisfies certain relations between the digits but does not have to belong to an individual with good credit.

(a) the set of syntactically correct (compilable) Java programs. No, because this involves checking matching parentheses (as well as other constructions that require something to match)

(b) the set of valid IP addresses. Yes, these follow a simple pattern with a fixed number of characters.

(c) the set of valid credit card numbers. Yes, for two reasons. First, the set of valid credit card numbers is finite and any finite set is regular. That answer, of course, is of no practical value. More usefully, credit card numbers have a fixed number of digits, so we can remember the digits so far seen in a state of the machine, and perform any computation we want with those digits.

(d) the set of palindromes (strings equal to their reverse) No, we used the pumping lemma to show this set is not regular.

(e) the set of valid social security numbers. Yes, same reasons as (c).

4. Convert the following grammar to an equivalent PDA:

\[ S \rightarrow aSbb \mid SS \mid \epsilon \]

Answer produced by the textbook algorithm: The starting state is 0, the only accepting state is 1, and the rules are as follows.

\[
\begin{align*}
0, \epsilon, \epsilon & \rightarrow 1, S \\
1, a, a & \rightarrow 1, \epsilon \\
1, b, b & \rightarrow 1, \epsilon \\
1, \epsilon, S & \rightarrow 1, aSbb \\
1, \epsilon, S & \rightarrow 1, SS \\
1, \epsilon, S & \rightarrow 1, \epsilon 
\end{align*}
\]
5. A recipe has two parts: a list of ingredients and a list of instructions. Each line of the ingredient list (i.e. each part of the ingredient list between two newlines, or between the start and the first newline) has the form number, followed by unit, followed by name, followed optionally by information in parentheses, as illustrated by this ingredient list:

2 C milk
1 C egg whites (beaten)
1 lb butter (melted)
1/2 C sugar

In this problem you are asked to give a grammar that describes the format of a recipe ingredient list. (We will ignore the instruction list part of a recipe for now.) The alphabet should be the ASCII alphabet. The grammar should not try to generate English words, but allow any sequence of letters as an ingredient, i.e. if we replace butter by szyefj in the above example, it is OK if that recipe list is generated. The grammar just has to enforce the rules mentioned above. Note that the recipe list is really just one string (that happens to contain some newline characters). The empty ingredient list is not OK. The possible units, however, are not arbitrary strings—you cannot replace “1 C” by “1 Z”. You cannot even write “1 stick butter” instead of “1/4 lb butter”. (This would be necessary if you want to write a program to scale up a recipe to any number of servings.)

\[
S \rightarrow L \text{ Newline } S \mid \epsilon \\
L \rightarrow \text{ Number Unit String Optional} \\
Optional \rightarrow (\text{ String }) \mid \epsilon \\
\text{ Number } \rightarrow \text{ Digit } \mid \text{ Number } \mid \text{ Fraction } \mid \epsilon \\
\text{ Fraction } \rightarrow \text{ Digit/Digit} \\
\text{ Unit } \rightarrow \text{ C } \mid \text{ Tbsp } \mid \text{ tsp } \mid \text{ T } \mid \text{ t } \mid \text{ oz } \mid \text{ lb } \mid \text{ g } \mid \text{ qt } \mid \text{ gal } \mid \text{ l} \\
\text{ String } \rightarrow \text{ Letter More} \\
More \rightarrow \text{ Letter More } \mid \epsilon \\
\text{ Letter } \rightarrow \text{ | a } \mid \text{ b } \mid \text{ c... } \mid \text{ z } \mid \text{ A } \mid \text{ ... } \mid \text{ Z}
\]
6. Please use the pumping lemma to prove that the set $L$ of $\alpha\alpha$ such that $\alpha$ is any string of $a$’s and $b$’s is not regular. For example, $abaaba$ is in $L$ but $abbaab$ is not.

Answer: Assume, for proof by contradiction, that $L$ is regular. Let $N$ be as in the pumping lemma. Choose $w = a^N b^N a^N b^N$. Then $w$ is in $L$. According to the pumping lemma we have $w = xyz$ with $|xy| \leq N$ and $|y| \geq 1$ and $xy^j z \in L$ for $j = 0, 1, 2, \ldots$. Since $|xy| \leq N$ and the first $N$ characters of $w$ are $a$, we must have $y = a^k$, and $k > 0$ since $|y| \geq 1$. Then $xy^2 z = a^{N+k} b^N a^N b^N$, which must belong to $L$ by the pumping lemma, but which does not belong to $L$ since it is not of the form $\alpha\alpha$, since its first half matches $a^* b^*$ but the second half starts with $b$, then has some $a$’s, and then more $b$’s. That completes the proof.
7. Please write a Turing machine (program) that erases the last half of its input. If the input has an odd number of characters, it erases the middle character as well as those following. For example, if started with \textit{aaaaabbbb}, it terminates with \textit{aaaa} on the tape, and if started with \textit{aaaabaaaa} it terminates with \textit{aaaa} on the tape. Please comment your code so it won’t take me all day to figure it out. Assume the input alphabet is \{a, b\}. Answer:

\begin{verbatim}
0,a->c,1,R // replace a by c
0,b->d,1,R // and b by d
1,a->a,1,R
1,b->b,1,R // and move right
1,_->_,2,L // back up when you hit blank
2,a->3,_,L // erase the last character
2,b->3,_,L
2,c->3,_,L // even if it was the middle character
2,d->3,_,L
3,a->3,a,L // and move left
3,b->3,b,L // without modifying the tape
3,c->0,c,R // until you hit c or d
3,d->0,d,R // and loop
0,->4,_,L // done erasing,
4,c->4,a,L // move left replacing c by a
4,d->4,b,L // and d by b
\end{verbatim}

Halt when you get to the left end of the tape, when you will read the triangle marker. (No instruction is necessary for that.) You could hit the triangle in state 3 if there was only one character of input, but that’s OK.
8. True or false:

(a) It’s impossible to write a Java program that takes Java programs \(e\) as inputs, and produces the same output as program \(e\) would produce. (Here the “output” of a Java program is what, if anything, it prints to the console.)

\[
\begin{array}{c}
\text{T} \\
\text{F}
\end{array}
\]

(b) It’s impossible to write a Java program that takes Turing machines \(e\) as inputs, and prints out 1 if Turing machine \(e\) halts (with no input) and 0 if it does not.

\[
\begin{array}{c}
\text{T} \\
\text{F}
\end{array}
\]

(c) It’s impossible to write a Java program that takes Java programs \(e\) as inputs, and prints out 1 if program \(e\) would eventually print something to the console, and 0 if not.

\[
\begin{array}{c}
\text{T} \\
\text{F}
\end{array}
\]

(d) It’s impossible to write a Turing machine that would take Java programs \(e\) as inputs, and produce the same output as program \(e\) would produce.

\[
\begin{array}{c}
\text{T} \\
\text{F}
\end{array}
\]

(e) It’s impossible to write a Java program that takes as input a Java program \(e\) and a number \(k\), and outputs 1 if the \(e\) outputs \(k\) (before printing anything else, although it may go on outputting other characters after printing the digits of \(k\)), and 0 otherwise.

\[
\begin{array}{c}
\text{T} \\
\text{F}
\end{array}
\]

9. Please classify the following sets as either recursive or r.e. (recursively enumerable), as in the homework. You do not have to justify or explain your answers.

(a) The set of palindromes (strings equal to their reverses, such as \(racecar\)). \textit{Recursive—we can write a program to test whether a string is a palindrome.}

(b) The set of strings accepted by a particular finite automaton. \textit{Recursive—just feed the input to the machine and see if it is in an accepting state when the input is gone. There is no question of waiting for it to finish, it takes exactly as many steps as the input has characters.}

(c) The set of strings accepted by a particular Turing machine (assume here that I have listed a machine with several thousand instructions that I do not expect you to read and analyze). \textit{r.e.; without analyzing the particular machine, for all we know it is a universal machine, which would recognize an r.e. but not recursive set.}

(d) The set of \(n\) such that, somewhere in the decimal expansion of \(\pi\), there occurs a string of \(n\) consecutive 7’s. \textit{r.e., because if there is a string of \(n\) sevens, we will find it eventually, but if there is not, we don’t know when to stop looking, in the present state of mathematical knowledge.}
(e) The set of Java programs that produce at least five characters of output to the console. r.e., because if five characters are eventually produced, we can accept that input string; but we can’t say “recursive”, because we have no program to tell us whether five characters will be eventually produced, and indeed no such program exists, because we can reduce the halting problem to this set: let $F(e)$ be the program that prints 1234 and then runs $e$ at input $e$; if it halts it prints 5. Then $e$ belongs to the halting set if and only if $F(e)$ prints at least five characters of output. Hence the set in this problem is not recursive.

10. Consider the Subset Product Problem. The input is a finite set $A$ of positive integers and a positive integer $K$. The problem is to determine if $A$ has a subset $B$ such that the product of the elements of $B$ is equal to $K$. For example, if $A = \{1, 3, 5, 7, 11, 17\}$, and $K = 165$, we could take $B = \{3, 5, 11\}$, but if $K = 8$ then there is no subset $B$ of this particular $A$ meeting the requirement. Note, numbers are represented in ordinary decimal notation for purposes of this problem.

Is this problem in $P$, or is it in $NP$, or is it in Co-$NP$? Briefly explain your answer.

Answer: it is in $NP$, because a non-deterministic Turing machine can first generate a subset $B$ of its input $A$, and then check (in polynomial time) whether $B$ satisfies the required condition. We cannot say that this problem is in $P$, because there is no (apparent) algorithm to generate the required set $B$ (and generating all of the subsets would take exponential time). In fact this problem is known to be $NP$-complete, so it is in $P$ if and only if all $NP$ problems are in $P$. 