Name:

1 a	20	
b	20	
c	20	
d	20	
e	20	
f	20	
g	20	
2	10	
3	30	
4	12	
5	15	
Total	207	

CS 341
Practice Final Exam

1. Please write neatly. You will lose points if we cannot figure out what you are saying.

2. Whenever you answer a question with a machine or a grammar, add comments to help us figure out what you are trying to do.

3. DO NOT WRITE JUNK and hope for partial credit. If you are trying to prove something and you know that you haven't got it right, say so. That may get you partial credit. But if you give the impression that you think that junk is nonjunk, you won't get points.

(1) For each of the following languages:

(i.) **(4 points)** Determine the appropriate category, as listed below.

(ii.) **(8 points)** Prove that the language is in the category you have specified, except as noted below, in which case these points will be assigned to part iii. You can do this part by writing an appropriate grammar (or regular expression), exhibiting an appropriate recognizing machine, using closure properties, or by showing that the language is finite. What I mean by "exhibit a recognizing machine" is:

- For an FSM: Draw it.
- For a PDA: Draw it, but if it is very complicated, also provide a description of how it works. We will give partial credit for the description even if the details of the machine are missing or wrong.
- For a Turing machine: Describe an algorithm in clear English. It need not be specifically a Turing Machine program. Any clear algorithm is acceptable.

(iii.) **(8 points)** Prove that the language is not in the next most restricted category, except as noted below, in which case these points will be assigned to part ii.

Assume the following language categories:

A. L is regular. (In this case, you can skip part iii.)
B. L is not regular but is context free
C. L is not context free but is decidable (In this case, if L is a set of strings that include descriptions of machines (FSMs, PDAs or TMs), you do not need to prove that it is not context free.)
D. L is not in D but is in SD.
E. L is not in SD. (In this case, you can of course skip part ii.)

You may take as theorems that the following languages are not decidable:

$H = \{\langle M, w\rangle : \text{TM } M \text{ halts on input string } w\}$
$H_\varepsilon = \{\langle M\rangle : \text{TM } M \text{ halts on the empty tape}\}$
$H_{ANY} = \{\langle M\rangle : \text{there is any string on which TM } M \text{ halts}\}$
The corresponding languages A, A_ε, and A_{ANY}.

You may take as theorems that the following languages are not semidecidable:

$\neg H = \{\langle M, w\rangle : \text{TM } M \text{ does not halt on input string } w\}$
$H_{\neg ANY} = \{\langle M\rangle : \text{there does not exist any string on which } M \text{ halts}\}$
$H_{ALL} = \{\langle M\rangle : \text{TM } M \text{ halts on all inputs}\}$
$EqTMs = \{\langle M_a, M_b\rangle : L(M_a) = L(M_b)\}$
The corresponding languages $\neg A$, $A_{\neg ANY}$ and A_{ALL}.

You may not use Rice's Theorem for these problems.

To get full credit for any reduction proof, you must justify, in detail, why the reduction is correct. So purely copying and pasting a standard reduction will not get full credit even if it happens to work.

Remember that:

- If S is a set, $|S|$ is the cardinality of S.
- If w is a string, $|w|$ is the length of w.
- $\#_{\mathtt{a}}(w)$ is the number of `a`'s in the string w.

(a) $L = \{\langle M, w\rangle : M \text{ moves right exactly twice while operating on } w\}$

D. If it can move right only twice, then *M* can read only the first two input characters. More generally, at any point in its computation, the only things that can affect its future behavior are:

- The contents of the two tape squares to the right of the read/write head, which may contain any element of Γ.
- The contents of the square under the read/right head, which may also contain any element of Γ.
- The contents of the two tape squares to the left of the read/write head, which may contain any element of Γ.
- The rest of the tape to the left, but every one of these squares must contain a blank, so there's no information here.
- The state of *M*.

So the number of distinct configurations of *M* is $maxconfigs = |\Gamma|^5 \cdot |K|$. If *M* ever enters a configuration a second time, it will do the same thing that it did the first time. So the following algorithm decides *L*:

1. Run *M* on *w* for *maxconfigs* + 1 steps or until it halts or moves right three times. Keep track of how many times it moves right. Note that if it didn't naturally halt, it is in an infinite loop.
2. If it halted and moved right exactly twice, accept.
3. If it halted and moved right some other number of times, reject.
4. If it moved right three times, reject.
5. If it didn't move right at all, reject. It is in a loop and will never move right.
6. If it moved right exactly once, reject. Either that one move is outside the loop, in which case there won't be any more, or it's inside the loop and there will be an infinite number of additional ones.
7. If it moved right twice, we need to find out whether those moves were inside the loop or preceded entry into the loop. So continue running *M* for another *maxconfigs* steps.
8. If the total number of rightward moves is still two, accept.
9. Otherwise reject.

(b) $L = \{wx : |w| = 2 \cdot |x| \wedge w \in \mathtt{a}^+\mathtt{b}^+ \wedge x \in \mathtt{a}^+\mathtt{b}^+\}$

Context-free, not regular. *L* can be accepted by a PDA *M* that pushes one character for each `a` and `b` in *w* and then pops two characters for each `a` and `b` in *x*.

L is not regular, which we show by pumping. Note that the boundary between the *w* region and the *x* region is fixed; it's immediately after the last `b` in the first group. Choose to pump the string $w = \mathtt{a}^{2k}\mathtt{b}^{2k}\mathtt{a}^{k}\mathtt{b}^{k}$. $y = \mathtt{a}^{p}$, for some nonzero *p*. Pump in or out. The length of *w* changes but the length of *x* does not. So the resulting string is not in *L*.

(c) $L = \{<M_a, M_b> : \text{if } M_a \text{ does not accept } \varepsilon \text{ then } M_b \text{ rejects } \varepsilon\}$

SD/D: Note that L is equivalent to $\{<M_a, M_b> : M_a \text{ accepts } \varepsilon \text{ or } M_b \text{ rejects } \varepsilon\}$.

The following algorithm semidecides L:

Run both machines on ε in parallel. If either M_a accepts or M_b rejects, halt and accept.

Proof not in D: R is a reduction from H = {<M, w> : TM M halts on w} to L, defined as follows:

R(<M, w>) =

1. Construct the description of $M\#(x)$ that, on any input, immediately accepts.
2. Construct the description of $M\#\#(x)$ that operates as follows:
   2.1 Erase the tape.
   2.2 Write w on the tape.
   2.3 Run M on w.
   2.4 Accept.
3. Return <$M\#\#$, $M\#$>.

If *Oracle* exists and decides L, then C = *Oracle*(R(<M, w>)) decides H. R can be implemented as a Turing machine. And C is correct. Note that $M\#$ rejects nothing. $M\#\#$ accepts everything or nothing, depending on whether M halts on w. So:

- <M, w> ∈ H: M halts on w so $M\#\#$ accepts everything, including ε. So *Oracle* accepts.
- <M, w> ∉ H: M doesn't halt on w so $M\#\#$ accepts nothing, including ε. $M\#$ rejects nothing. So *Oracle* rejects.

But no machine to decide H can exist, so neither does *Oracle*.

(d) $L = \{<M> : M \text{ rejects at least two even length strings}\}$

SD/D: The following algorithm semidecides L:

Run M on the even length strings in Σ* in lexicographic order, interleaving the computations. As soon as two such computations have rejected, halt.

Proof not in D: R is a reduction from H = {<M, w> : TM M halts on w} to L, defined as follows:

R(<M, w>) =

1. Construct the description of $M\#(x)$ that, on input x, operates as follows:
   1.1. Erase the tape.
   1.2. Write w on the tape.
   1.3. Run M.
   1.4. Reject.
2. Return <$M\#$>.

If *Oracle* exists and decides L, then C = *Oracle*(R(<M, w>)) decides H:

- <M, w> ∈ H: M halts on w so $M\#$ rejects everything and thus rejects at least two even length strings, so *Oracle* accepts.
- <M, w> ∉ H: M doesn't halt on w so $M\#$ doesn't halt and thus rejects nothing and so does not reject at least even length two strings. *Oracle* rejects.

But no machine to decide H can exist, so neither does *Oracle*.

(e) $L = \{w = xyx^R : x \in \{0, 1\}^+, y \in \{0, 1\}^*\}$

Regular. There is no reason to let x be more than one character. So all that is required is that the string have at least two characters and the first and last must be the same. $L = (0\ (0 \cup 1)^*\ 0) \cup (1\ (0 \cup 1)^*\ 1)$.

(f) $L = \{<M> : L(M) \text{ is not regular}\}$

¬SD: R is a reduction from ¬H = {<M, w> : TM M does not halt on w} to L, defined as follows:
R(<M, w>) =
1. Construct the description of $M\#(x)$ that operates as follows:
   1.1. Save its input x on a second tape.
   1.2. Erase the tape.
   1.3. Write w.
   1.4. Run M on w for $|x|$ steps or until it halts.
   1.5. If M would have halted, then loop.
   1.6. Else: if $x \in a^n b^n$, accept. Otherwise reject.
2. Return <$M\#$>.

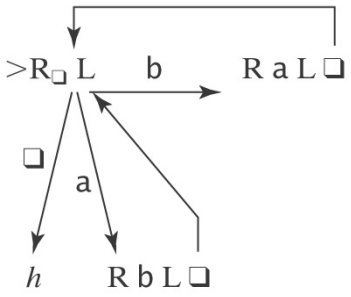
If *Oracle* exists and semidecides L, then C = *Oracle*(R(<M, w>)) semidecides ¬H:
- <M, w> ∈ ¬H: M does not halt on w, so $M\#$ always gets to step 1.6. So $L(M\#) = a^n b^n$, which isn't regular. *Oracle* accepts.
- <M, w> ∉ ¬H: M halts on w. Suppose it does so in k steps. Then, for all strings of length k or more, $M\#$ loops at step 1.5. So $L(M\#)$ is finite and thus regular. So *Oracle* does not accept.

But no machine to semidecide ¬H can exist, so neither does *Oracle*.

(g) $L = \{wxw : |w| = 2 \cdot |x|, w \in \{a, b\}^*, x \in \{c\}^*\}$

Not context-free. We prove it using the Pumping Theorem. Let $w = a^{2k} c^k a^{2k}$.

(2) Consider the following Turing Machine M, described in our macro language:



Give a short English description of what M does. Don't describe how it operates. Describe its result.

Shift the input string one character to the right and replace each b with an a and each a with a b.

(3) Give an example that proves each of the following:

(a) It is possible, given two languages L_1 and L_2, both in SD/D, that $L_1 \cap L_2$ is in D.

PCP $\cap$ H = $\varnothing$, which is regular and thus in D.

(b) It is possible that, if L_1 and L_2 are context free, $L_1 \cap L_2$ is not context free.

Let L_1 be $\{a^n b^n c^m : n, m \geq 0\}$, which is context-free. Let L_2 be $\{a^m b^n c^n : n, m \geq 0\}$, which is also context-free. $L_1 \cap L_2 = A^n B^n C^n$, which is not context-free.

(c) The SD languages are not closed under complement.

H is in SD. But $\neg$H is not.

(d) Let L_A be the set of all languages defined over some alphabet A. Then let md: $L_A \rightarrow L_A$ be a function that maps from one language to another, defined as follows:

$$md(L) = \{x : \exists w \in L\ (w = sxt, s \in \Sigma^*, x \in \Sigma^*, t \in \Sigma^*)\}$$

If L is context free but not regular, $md(L)$ may be regular.

Let: $L = A^n B^n$, which is context-free but not regular.
Then: $md(L) = \{a^i b^j : i, j \geq 0\}$, which is regular.

(e) It is possible that, if $L_1 \leq L_2$ and $L_2 \in$ SD then $L_1 \in$ D.

Let $L_1 = \{a\}$ and let L_2 = H. $\{a\} \leq$ H. But $\{a\}$ is in D.

(f) There exists a language L that is Turing enumerable but not lexicographically Turing enumerable.

Any language that is in SD/D will serve. So let L = H.

(4) Let M be an arbitrary Turing machine. Suppose that $timereq(M) = 3n^3(n+5)(n-4)$. Circle all of the following statements that are true:

a) $timereq(M) \in \mathcal{O}(n)$. False
b) $timereq(M) \in \mathcal{O}(n^6)$. True
c) $timereq(M) \in \mathcal{O}(n^5/50)$. True
d) $timereq(M) \in \Theta(n^6)$. False

(5) Show that SUBSET-SUM = {<*S*, *k*> : *S* is a multiset (i.e., duplicates are allowed) of integers, *k* is an integer, and there exists some subset of *S* whose elements sum to *k*} is in NP.

A deterministic, polynomial-time verifier *V* operates as follows:

1. It checks the length of a proposed certificate *c*. If it is longer than |*S*|, it rejects. Otherwise it continues.
2. For each element *e* of *c*, it walks through *S* looking for an unused element that matches *e*. If it finds one, it marks it as used. If it fails to find one, it immediately rejects.
3. It sums the elements of *c*. If the sum is equal to *k* it accepts, otherwise it rejects.

V runs in polynomial time because:

1. Step 1, checking, the length of *c*, takes $\mathcal{O}(|S|)$ time.
2. Step 2, checking that all the elements in *c* are in *S*, can be done in $\mathcal{O}(|S|^2)$ time.
3. Step 3, checking the sum: *V* can sum the elements of *c* in $\mathcal{O}(|c|)$ time (assuming constant cost per addition). And it can compare that sum to *k* in $\mathcal{O}(k)$ time.

So timereq(V) $\in \mathcal{O}(|\langle S, k\rangle|^2)$.