15-354: Midterm

October 17, 2017

Name:	Alexander Chi
Andrew ID:	achi

Instructions

- Fill in the box above with your name and your Andrew ID. **Do it, now!**
- Clearly mark your answers in the allocated space. If need be, use the back of a page for scratch space. If you have made a mess, cross out the invalid parts of your solution, and circle the ones that should be graded.
- Scan the test first to make sure that none of the 14 pages are missing. Some parts may be on the back of the page.
- The problems are of varying difficulty and are not necessarily sorted in order of increasing difficulty. You might wish to pick off the easy ones first.
- One sub-question is an open problem in complexity theory. Mark it as such and don't waste any time on it.
- You have 80 minutes. Good luck.

1	20	0
2	20	30
3	20	10
4	20	20
5	20	20
Total	100	70

Problem 1: Wurzelbrunft Functions (20 pts.)

Someone has given Wurzelbrunft a recursion theory book for his birthday. He finds large function values confusing, so he decides to study k-W functions: any computable function $f : \mathbb{N} \to \mathbb{N}$ such that $f(x) < k$ for all x.

Wurzelbrunft thinks that, for sufficiently small k, all k-W functions are primitive recursive. Help him come up with an actual theorem and a proof. The stronger the theorem, the better.

A. **Claim:**

any computable function f w/ k-W for finite k is primitive recursive

B. **Proof:**

subclaim: any decidable $f : \mathbb{N} \to \{0,1\}$ is P.R.

in particular, since we know that $\exists$ TM T deciding f, and it must be finite, we also know we have a finite runtime on the TM, because it should halt

then we can use the finite TM as a "lookup table" to simulate itself in a P.R. manner
namely. we can pass a sequence as our "tape"

□ subclaim

then we can see that if

$f : \mathbb{N} \to [k]$ that $f = f_1, f_2, \ldots f_k$ for f_i decision problems.

since finite addition is PR we have the desired result

Problem 2: Computing Transients and Periods (20 pts.)

For the following, assume we are given a C program that computes a function $f : A \to A$ where $A = \{0, 1, \ldots, 10^{15} - 1\}$. Assume that the computation of $f(a)$ is constant time but expensive.

Write t for the transient of a point in A, and p for the period. We can compute t and p in a memoryless way using Floyd's algorithm, but sometimes there are better solutions. If you write pseudo-code to describe your algorithms below, make sure to provide ample comments; no credit otherwise.

A. Suppose that the transients of all points under f are at most 3 (three). Give a fast and memory efficient algorithm to compute the transient and period of a point $a \in A$. State the running time of your algorithm in terms of t and p.

$2p+2t = p+t$

put pebbles on the first three iterations

have another pebble go until it hits a pebble

then whichever iteration that static pebble is

is t and the other pebble's distance $-t$ is p

this would have running time.

$t+p$

B. Now suppose that the periods of all points under f are at most 3 (three). Give a fast and memory efficient algorithm to compute the transient and period of a point $a \in A$. State the running time of your algorithm in terms of t and p.

have three pebbles a,b,c. b two iterations ahead,
a,3 and c,0 keep going until two collide,
if it is the

a&b → p=1 t= distance travelled by b
b&c → p=2 t= distance travelled by c
a&c → p=3 t= "

~~assuming each pebble~~

with time
t+p+1

Problem 3: Counting and Decidability (20 pts.)

Suppose $A \subseteq \mathbb{N}$. It is often important to understand the behavior of the corresponding counting function $f_A : \mathbb{N} \to \mathbb{N}$ defined by

$$f_A(n) = \text{ cardinality of } (A \cap \{0, 1, \ldots, n-1\})$$

For example, when A is the set of primes, the corresponding prime-counting function has been studied in great detail in number theory. It behaves roughly like $n/\ln n$, but the details are very messy.

A. Show that f_A is computable whenever A is decidable.

if A decideable $\exists g(x)$ s.t. $g(x) = yes$ iff $x \in A$

thus we could compute.

```
f_A(n) =
    int acc=0
    for int i:=0, i<n, i++ {
        if (g(i)) {
            acc++; }
    }
```

this could be expensive but it would always be computable and correct

thus by church turing

f_A should be computable

B. Show that A must be decidable when f_A is computable.

if f_A computable then

$x \in A$ iff $|A \cap \{0 \ldots x\}| - |A \cap \{0 \ldots x-1\}| = |(\text{or } A \cap \{0\} \text{ if } x=0)$

this should be correct since

$x \in A$ iff $(A \cap \{0 \ldots x\}) - (A \cap \{0 \ldots x-1\}) = \{x\}$ since

$(A \cap \{0 \ldots x\}) - (A \cap \{0 \ldots x-1\}) \subseteq A \cap \{x\} \subseteq \{x\}$

memory $x \in A$ iff $A \cap \{x\} = \{x\} \iff x \in A$

C. Now assume that A is semidecidable and f_A is computable. Give another decision algorithm for membership in A (different from the one in part (B)). *Comment:* This is not frivolous; it is important in Kolmogorov complexity.

open ?.

X

not a complexity theory question ...

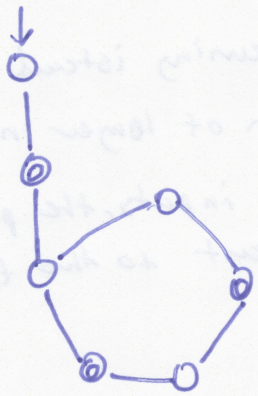
Problem 4: Tally Languages (20 pts.)

Languages $L \subseteq \{a\}^\star$ are sometimes referred to as tally languages. Let $c(n) = |L \cap \{\varepsilon, a, \ldots, a^{n-1}\}|$ and define the density of L to be $\Delta(L) = \lim_{n\to\infty} c(n)/n$. For example, $\Delta((aa)^\star) = 1/2$.

Regular tally languages are fairly easy to describe.

A. Characterize the minimal DFAs associated with regular tally languages.

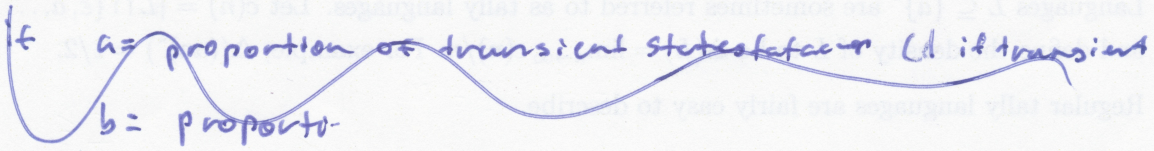
they would be like the single letter languages of our HW, which we could characterize as them as cycles with transients, i.e. something like this,



1. transient can't merge with cycle.
2. cycle states unique.

this we can just by by saying that for any language on $\{a\}^*$ that is regular, that by pidgeonhole thm. there are minimal n, and some m s.t. $\delta^m = \delta^n$ meaning that after n, we just loop this means the transient is m states and the loop is $\frac{1}{2}$

B. Explain how to compute the density $\Delta(L)$ for any regular tally language L.



the proportion of cycle states that accept

we could justify this by saying that past a certain finite length, the "transient" states stop mattering, meaning instead that it depends on the fraction of longer inputs and since within those inputs, the proportion that accept are equivalent to the fraction of accepting cycle states

e.g for $(aa)^*$

$\frac{1}{2}$ of states accepting

$\Delta = \frac{1}{2}$

Problem 5: Word Shuffle (20 pts.)

The word shuffle operation, in symbols $\|$, is a map from $\Sigma^\star \times \Sigma^\star$ to $\mathfrak{P}(\Sigma^\star)$ defined by

$$\varepsilon \,\|\, y = y \,\|\, \varepsilon \;=\; \{y\}$$
$$xa \,\|\, yb = (x \,\|\, yb)\, a \cup (xa \,\|\, y)\, b.$$

As usual, we can extend the operation to languages by

$$K \,\|\, L = \bigcup \{\, x \,\|\, y \mid x \in K, y \in L \,\}$$

For example, $aa \,\|\, bbb = \{aabbb, ababb, abbab, abbba, baabb, babab, babba, bbaab, bbaba, bbbaa\}$.
Also, $a^\star \,\|\, b^\star = \{a, b\}^\star$.

A. Let $\mathcal{A}_1 = \langle\, Q_1, \Sigma, \tau_1; I_1, F_1 \,\rangle$ and $\mathcal{A}_2 = \langle\, Q_2, \Sigma, \tau_2; I_2, F_2 \,\rangle$ be two NFAs accepting regular languages K and L, respectively. Show how to construct a finite state machine $\mathcal{A} = \langle\, Q, \Sigma, \tau; I, F \,\rangle$ that accepts the shuffle language $K \,\|\, L$. Explain what you are doing.

$Q =$ $Q_1 \times Q_2$ or a set isomorphic to it — this is to represent the state one subsequence is in and which state the other is in

$\tau =$ for $T(x, (S_1, S_2))$

$T = (T_1(x), S_2) \cup (S_1, T_2(x))$

$I =$ $I_1 \times I_2$

Justification:
This essentially is equivalent to

$ax \,\|\, by = a(x \,\|\, by) \cup b(ax \,\|\, y)$
meaning that we would say that $x \in K \,\|\, L$
iff two disjoint subsequences
a, b in x
$a \in K$ $b \in L$ meaning we
cause on which char is in which subsequence

$F =$ $F_1 \times F_2$

B. Now suppose that $\mathcal{A}_1$ and $\mathcal{A}_2$ are in fact DFAs. Give a good upper bound on the size of a DFA that accepts $K \parallel L$ in terms of n_1 and n_2, the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. No lower bound is required (it's messy), but you must explain your upper bound.

$\leq 2^{n_1 \cdot n_2}$

this is first derivable from the determinization of the NFA from before we could treat the DFA as an NFA.

then create the NFA used in part a)

and assuming all states are reachable, our only restriction is on expansion but since we cannot assume anything about it, we can only remove states, not containing at least 1 start thus giving

so $\leq 2^{n_1 n_2} - 2^{(n_1-1)(n_2-1)}$

blank

blank