Harvard University
Computer Science 121

Problem Set 8

Due November 17, 2015 11:59pm

Problem set by Kevin Zhang

Collaboration Statement: I collaborated with Tomoya Hasegawa and Jessica Zhu, got help from no one else, and referred to no other resources.

PART A (Graded by Zhengyu and Geoff)

PROBLEM 1 (2+2+2+2+2+2 points, suggested length of 2/3 page)

For each part (A) through (F), answer TRUE or FALSE and briefly justify your answers.

(A) $n^6 - n^4 = \Theta(n^6 - 5n^3)$.

(B) $\sqrt{n} = \Omega(n^{\sin n})$.

(C) $n! = O(2^n)$.

(D) $n^{100001} = o(1.00001^n)$. (Hint: L'Hôpital's rule)

(E) $3^n = 5^{O(n)}$.

(F) Suppose that $f(n) = g(O(n))$. Give a counterexample to prove that this does not necessarily imply that $f(n) = O(g(n))$, and show why your counterexample works.

Solution.

(A) True; both are polynomial and have highest power 6, so their asymptotic complexities are exactly the same.
(B) False; the given expression would mean that $n^{\sin n} \leq c\sqrt{n}$; for large enough values of n where $\sin n = 1$, this becomes obviously false.
(C) False; the left side grows by increasing factors, while the right only grows by a factor of 2 for each increment in n.
(D) True; the given statement is equivalent to saying $\lim_{n\to\infty} \frac{n^{100001}}{1.00001^n} = 0$; L'Hôpital's rule applied exactly 100002 times gives $\lim_{n\to\infty} \frac{0}{1.00001^n * \ln(1.00001)^{100002}} = 0$.
(E) True; $3^n \leq 5^{\log_5 3 * n}$.
(F) Consider $f(n) = 3n^{3n}$; this is equal to $g(O(n))$ where $g(n) = n^n$ and $c = 3$ in $O(n) = cn$, but $O(g(n)) = O(n^n)$; the equation $3n^{3n} = O(n^n)$ is obviously not true since n^{3n} grows much faster than n^n.

PROBLEM 2 (6 points, suggested length of 1/3 page)

Two languages H and K are *recursively separable* if there exists some recursive language R such that $H \subseteq R$ and $K \subseteq \overline{R}$. Prove that the languages $L_1 = \{\langle M \rangle : M \text{ accepts } \langle M \rangle\}$ and $L_2 = \{\langle M \rangle : M \text{ rejects } \langle M \rangle\}$ are not recursively separable.

Solution.
Suppose that there is a decidable language R such that $L_1 \subset \overline{R}$ and $L_2 \subset R$; this means that $R \cap L_1 = \emptyset$ (i.e. there are no machines in R that accept their own descriptions). R has a Turing machine that accepts exactly R; let's call it T. It's obvious that $\langle T \rangle$ is either in L_1 or L_2; suppose it's in L_1. Then since $\langle T \rangle$ is in L_1, T accepts $\langle T \rangle$. But this means that $\langle T \rangle$ is in R, since T accepts $\langle T \rangle$, which is a contradiction since the intersection of R and L_1 is the empty set. Hence, $\langle T \rangle$ cannot be in L_1. Now suppose it's in L_2; that is, T does not accept $\langle T \rangle$. But since $L_2 \subset R$ and $\langle T \rangle \in L_2$, $\langle T \rangle \in R$; i.e. T does accept $\langle T \rangle$, a contradiction. Hence, there exists no such R.

PROBLEM 3 (6 points, suggested length of 1/3 page)

A function $f : \Sigma^* \to \Sigma^*$ is *computable* if there exists a Turing Machine M that, when given $s \in \Sigma^*$ as input, halts with $f(s)$ written on the tape. The *range* of f is $\{f(x) : x \in \Sigma^*\}$. Prove that a nonempty language is r.e. if and only if it is the range of a computable function.

Solution.
Suppose we have a nonempty recognizable language L. Since L is recognizable, there is an enumerator that prints out the strings in L one at a time. Moreover, we can construct a bijective function $g : \mathbb{N} \to \Sigma^*$ since Σ^* is countable. We construct f using the enumerator that prints out the strings in L as follows: for $s \in \Sigma^*$, $s = g(k)$ for some k. Run the enumerator that prints out strings in L; $f(s)$ is equal to the kth string printed by the enumerator. Thus, the output of f is always some string in L, and we have constructed a function that prints out all strings in L. We can construct a TM that halts with $f(s)$ on the tape as follows: given a string $s \in \Sigma^*$, calculate k such that $g(k) = s$ (this is possible because g is bijective and so can be inverted). Then, run the enumerator for L and count the strings that are printed. When s is printed out, take the current count and pass it into g, then erase the input tape and write the output of g on the tape. This process is guaranteed to halt since any particular string in L must be printed out at some point.

Now suppose we have a computable function f; we want to show that the range of f is recognizable. To do this, we need to construct a TM that accepts a string iff it is in the range of f. We do this using g above; we define the following TM:

- Do the following starting from 0 and counting upwards:
  - Compute $f(g(i))$, where i is the current index. If the result is equal to the input string, halt and accept. Otherwise, continue to the next index.

It should be clear that this process halts and accepts iff the input string is an output of the function f, and otherwise runs forever (i.e. rejects the input string).

PROBLEM 4 (CHALLENGE! (3 extra credit points) points, suggested length of 1/2 page)

In this problem you will define a function that grows faster than any computable function. Thus you will prove the existence of an uncomputable function directly, without relying on Turing's diagonalization argument.

The **busy-beaver function** $\beta(n)$ is the largest number of a's that can be printed by any n-state, two-symbol Turing machine that eventually halts when started from the empty tape.

Show that β is not computable.

Solution.

PART B (Graded by Madhu and Erin)

PROBLEM 5 (6+6 points, suggested length of 1/2+2/3 pages)

Prove that the class $\mathcal{P}$ is closed under:
(A) Concatenation.
(B) Kleene star. (*Hint:* Look at the algorithm we gave in class for recognizing context-free languages via Chomsky Normal Form.)

Solution.

(A) Suppose we have languages A and B in $\mathcal{P}$. To test if a string s is in $A \circ B$, we only need to check every possible pairing of strings that could make s; to be exact, we just split s into two strings at every possible place and check if the first string is in A and the second string is in B. Since A and B are decidable languages in $\mathcal{P}$, they have Turing machines with polynomial runtimes; let's call them T_1 and T_2, with corresponding polynomial runtimes $O(n^p)$ and $O(n^q)$. We construct the following Turing machine for some input string s:

- Do the following for all possible pairs of strings that can be concatenated to make s:
  - Pass the first string in the pair to T_1. If T_1 accepts, move on; if it rejects, reject.
  - Pass the second string in the pair to T_2. If T_2 accepts, accept s; if it rejects, reject.

This only accepts if $s \in A \circ B$, and has polynomial run-time $O(n(O(n^p)+O(n^q))) = O(n^{\text{MAX}(p,q)+1})$ since the loop is run at most n times and each runthrough takes $O(n^p) + O(n^q)$ runtime.

(B) Suppose we have a language $L \in \mathcal{P}$, and a corresponding TM T with runtime $p(n)$. We construct the following TM T':

- For an input string s of length n, do the following for each of $i = 1$ through $i = n$:
  - For each of $j = 1$ through $j = n - i + 1$, do the following:
    - Pass the substring of s starting at index j of length i to T. T must either accept or reject because its language is in $\mathcal{P}$. If T accepts, keep this substring in memory. If T rejects, do the following for $k = j$ through $k = j + i - 1$:
      - If the string starting at j and ending at index k AND the string starting at index $k + 1$ and ending at index $i + j - 1$ are both saved (the empty string counts as saved), save this substring as well.

- At the end of the above loops, if the entire string has been saved, accept. Otherwise, reject.

The above process works by checking every possible substring of the input string and building upwards from there to check if the total string is a composite of accepted strings. For example, if the input string is *abcd*, it would check *a*, *b*, *c*, and *d* in turn, then check *ab*, *bc*, *cd*, then check *abc* and *bcd*, and finally *abcd*. The outermost loop of this algorithm is run n times; the next loop is run approximately n times per outer loop, and the inner loop takes $p(n) + O(n)$ at most. Hence, the runtime of this algorithm is $O(n) * O(n) * (p(n) + O(n))$, or $O(n^{2+\text{MAX}(k,1)})$, where $p(n) = O(n^k)$.

PROBLEM 6 (8 points, suggested length of .5 pages)

Prove that $\text{ALL}_{\text{DFA}} = \{\langle D \rangle : D \text{ is a DFA and } L(D) = \Sigma^*\}$ is in $\mathcal{P}$.

Solution.
This is simply a matter of checking all possible paths in a DFA D to make sure that they end in an accept state. In particular, every state along any path must be an accept state, or the string that led to that path is not accepted by the DFA. We write a TM T that decides ALL_{DFA} as follows:

- Check that an input string s encodes a valid DFA D; if no, reject. Otherwise, decode into a DFA $D = (Q, \Sigma, q_0, \delta, F)$.
- Mark the start state of D (which is q_0).
- Mark any states that can be transitioned to from any already-marked states. Repeat this step while this step successfully marks a state.
- If any states encountered in steps 2 and 3 are not accept states, reject. Otherwise, continue.
- If step 3 terminates without rejecting, accept because all reachable states in D are accept states and therefore D accepts all strings.

This machine clearly halts, and decides whether an input DFA D accepts all strings or not. The polynomial runtime of this machine depends on both the length of the input string and the number of states in the DFA; step 1 takes $O(n)$, where n is the length of the input string, and steps 2 and 3 take at MOST $O(q)$ time, where q is the number of states in D. Thus, the runtime of this algorithm is $O(\text{MAX}(n, q))$.