Autumn 1997–98, Prof. Boneh
Solution Set
CS 154, Problem Set 6

6.1 (5 points)

PROBLEM: **Determine which of the following problems are decidable and which aren't. (a)** DEAD CODE ELIMINATION: **Given a program, does it have a line that is never reached? (b)** SECURITY BREACH DETECTION: **Given a program, does it ever read** /etc/passwd**? (c)** TIME SINK: **Given a program, will it run for more than a million steps on any input?**

(a) Undecidable. I'll give two different solutions, both of which also solve part **(b)**.

Method 1 The first method ignores my hint about treating programs as TMs and actually writes a C program that serves as a reduction from L_{HALT}. Here's the pseudo-code, which given M and w writes a Universal TM with an extra step at the end.

```
set tape = input w, put head on 1st square, put state in q_0.
do {
   change tape square, move head, and change state based on delta(M).
} until ( state = q_a or state = q_r or head moves off left end of tape )
read /etc/passwd
```

Obviously, every line of this C program is read at least once, except maybe the last line which is not read if M runs forever on w (we treat falling off the left end of the tape as immediately halting and rejecting, following Prof. Boneh's definition). Answering if there's an unreached state in the computer program answers the halting question.

Method 2 Here's the correct way to use my hint, which says to treat every line of code as a state, so L is "is any state never reached?" We reduce from $L_{\text{NOT-EMPTY}}$ by taking the input machine M for NOT-EMPTY and adding a new letter #. If the input is # then we cycle through all the states except q_a and q_r, and then go to q_r. Thus, every state is reachable except maybe q_a. Since our new machine M' acts exactly the same as M on M's alphabet, M' reaches q_a on some input iff M accepts some word. Thus, if we hand M' to the DEAD STATE oracle, if it says "yes" then M is not in $L_{\text{NOT-EMPTY}}$, otherwise it is.

(b) Undecidable. The two methods rom part **(a)** solve this problem directly, because they are constructed so that all lines/states are reachable except one. We just make that line/state the one that reads `/etc/passwd`. Thus, the oracle for SECURITY BREACH will act exactly the same as the one for DEAD CODE and the problem is undecidable for the same reason.

Another solution, due to Zheng Huang, is one of the few that took our "give an outline" at its word and did it properly. The argument is this: Think of each line as a state. Suppose state q_i is the state which reads `/etc/passwd`. Swap states q_i and q_a. Then the problem is "given a program, does it ever reach q_a?" This is just $L_{\text{NOT-EMPTY}}$.

(c) Decidable. The reason is that we only need to consider (relatively) short strings.

The algorithm works as follows: construct all strings of length at most 1 million (there are a finite number). Run M on each string for at most 1 million steps (a finite number of steps). Accept if some string didn't halt by the millionth step, reject otherwise.

The reason this works is that the TM can only look at the first million characters of any input: to read later characters would require more than a million steps! Thus, if a string halts within a million steps, it does so with the machine looking at only the first million characters of it.

Another way to think of this is that strings with the same first million characters are in an equivalence class with respect to this problem. Problems that require us to consider all inputs are decidable only if the inputs fall into a fininte number of equivalence classes. Another example of a decidable problem is this: does a TM M halt after only one step on any input?

6.2 (5 points)

PROBLEM: **Which of the following statements are true? (a)** $n^n = O(2^n)$ **(b)** $\log(n^5) = O(\log n)$ **(c)** $\log^5 n = \Omega(\log n)$ **(d)** $2^n + n^{1000} = \theta(2^n)$ **(e) if** $f(n) = o(\log n)$ **then** $2^{f(n)} = O(n)$**.**

(a) False. If it were true, then there would be a c such that $\frac{n^n}{2^n} \leq c$ for all $n \geq N_0$. But $\frac{n^n}{2^n} = (n/2)^n$ which is clearly larger than c when $n = \max\{1, N_0, 2c\}$.

(b) True. Try $c = 5$, $N_0 = 1$. Since $\log(n^5) = 5 \log n$, $\frac{\log n^5}{\log n} = 5 \leq c$ for all n.

(c) True. Here, the bound holds if there exists a c and N_0 such that for all $n > N_0$, $\frac{\log^5 n}{\log n} \geq c$. In this case, the ratio equals $\log^4 n$ which is $\geq c$ if $c = 1$ and $N_0 = 10$, say. (Or d, if d is the base of the logarithm.)

Note: Some people weren't sure of the terminology. $\log^i n = (\log n)^i$ (just like $\sin^2 x$, etc). If you want to say $\log \log \log \ldots \log n$,k you write it $\log^{(i)} n$. $\log^{(5)} n$ is not in $\Omega(\log n)$.

(d) True. For θ, the bound holds if there exist c_1 and c_2 such that, for $n \geq N_0$,

$$c_1 \times 2^n \leq 2^n + n^{1000} \leq c_2 \times 2^n$$

$c_1 = 1$ obviously works for all n. $c_2 = 2$ works for $n > 10000$, because for such large n $2^n > n^{1000}$ (you can check this yourself; I recommend a particularly powerful calculator.)

(e) True. For little-o, the condition is that for *all* c there exists an N_0 such that for $n \geq N_0$, $f(n) \leq c \log n$. Thus, since raising to a power is monotone, we have $2^{f(n)} \leq 2^{c \log n}$. I'll pretend the base of the log is 2 (but see below) to get $2^{f(n)} \leq n^c$. Since this holds for *all* c, in particular it holds for $c = 1$. (If the base of the log had been b instead of 2, we would have picked $c = \log_2 b$.) Thus, for all $n \geq N_0$, $2^{f(n)} \leq n$, so $2^{f(n)}$ is in $O(n)$.

Note: $2^{f(n)}$ is *not* in $o(n)$, because $2^{f(n)}$ may be $> cn$ for $c < 1$.