Homework 2

Michael Anderson

April 12, 2011

CS517

Prof. Cull

1

	0	1	2	3	...
f_0	0	0	0	0	...
f_1	1	1	1	1	...
f_2	2	2	2	2	...
f_2	3	3	3	3	...
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	

The diagonal function $d(0) = 0$, $d(1) = 1$, $d(2) = 2$, ... is simply the non-constant function $d(n) = n$.

2

Given that the $g(x)$'s are actually different constant functions, then $d(n)$ maps to different values on each of its input, which by definition makes it non-constant.

As for the BIG question, if the constant functions $g_0, g_1, g_2, \ldots$ are listed in no particular order (I assume that this is what the question is asking, or else it is trivial), then we cannot predict the growth rate of a diagonal function. Because the constant functions could be ordered randomly, there is not necessarily any relationship between n and $d(n)$, $d(n)$ could be increasing over some intervals of n, and decreasing in others. Therefore there is no way to say that when n is greater than some constant, it is in some Θ or Ω class.

3

Let $F(i) = p_i(x)$ be the computable function that enumerates the polynomials. Let a diagonal function d be defined as $d(n) = Perturb(p_n(n))$, where $Perturb(x)$ is some computable function such that $\forall x$, $Perturb(x) \neq x$.

d is computable, because it is representable as the composition of two computable functions: $d(n) = Perturb(F(n)(n))$. d is not a polynomial because all of the polynomials are enumerated in the table, and given some arbitrary polynomial p_n, $d(n) \neq p_n(n)$ by the definition of $Perturb$.

4

All of the argument in (3) generalizes to arbitrary classes of functions. Again suppose there is a computable $F(i) = f_i$ that enumerates all of the members of the class. Let $d(n) = Perturb(p_n(n))$ where $Perturb$ is a function that does not map any of its possible inputs to themselves.

d is computable by the computation $Perturb(F(n)(n))$, and d does not belong to the class enumerated by $F(n)$, because $d(n) \neq f_n(n)$ for any n.

5

Can prove the claim by exhaustion.

Since $x^4 > 6$ if $|x| > 1$, for integer valued x, we need only consider $x \in \{-1, 0, 1\}$.

Since $6 - 0 = 6$ does not have an integer root, in other words there is no integer y that satisfies $y^2 = 6$, $x \neq 0$. Similarly, Neither does $6 - 1 = 5$, so $x \neq 1$ and $x \neq -1$. Therefore the equation has no integer solutions.

Since the set of integers is a superset of the set of natural numbers, there are also no natural number solutions.

6

From the notes, *Recursive* is the set of sets with recognizers, *RE* is the set of sets with acceptors, and *coRE* is the set of sets with rejectors.

Assume that there is a set S that has both an acceptor and a rejector. Then imagine a program that runs the acceptor for S and the rejector for S in two threads, and returns YES if the acceptor thread halts and returns YES, and returns NO if the rejector thread halts and returns NO. Such a program would be a recognizer for S. So if S has an acceptor and a rejector, it also has a recognizer, and therefore $RE \cap coRE \subseteq Recursive$.

Assume that there is a set S that has a recognizer. Since by definition a recognizer halts and returns YES for inputs in S it is an acceptor, and since it halts and returns NO for inputs not in S it is a rejector. So if S has a recognizer, it also has an acceptor and a rejector, and therefore $Recursive \subseteq RE \cap coRE$.

$$RE \cap coRE \subseteq Recursive \quad \text{and} \quad Recursive \subseteq RE \cap coRE \quad \Longrightarrow$$

$$Recursive = RE \cap coRE$$

7

8

(a) If for some input f_1, f_2 to IN-EQ-PRIM the correct answer is YES, in other words that $\exists v, f_1(v) \neq f_2(v)$, then an acceptor can be built that will return YES, and therefore IN-EQ-PRIM is in RE.

Such an acceptor could simply iterate through all possible values of v, compute each $f_1(v)$ and $f_2(v)$, and halt and return YES when a v is found such that $f_1(v) \neq f_2(v)$. This is doable in finite time because if the correct answer is YES then such a v actually exists and will eventually be found by the acceptor, and because $f_1 \in PRIM$ and $f_2 \in PRIM$ so they have finite runtime for all inputs.

(b) Let $R_H(s)$ be a hypothetical recognizer that returns YES if s is in the halting set, and NO if s is not in the halting set. Let $R_I(f_1, f_2)$ be a hypothetical recognizer for the IN-EQ-PRIM problem. Let $f_p(x)$ be a program that runs the first x steps or instructions of a program p. $f_p(x)$ should return HALT if p finished executing in x steps or less, and it should return DID NOT HALT if p did not finish after x steps. Let $SR(x)$ be a program that simply returns DID NOT HALT regardless of the input.

Now we can define a recognizer for the halt set as $R_H(s) = R_I(f_s(x), SR(x))$. Since R_I is a recognizer for IN-EQ-PRIM it can see if $f_s(x) = SR(x)$ for all values of x, in which case it correctly returns NO... s will never halt regardless of how big x gets. If s halts at some point, $f_s(x)$ returns HALT for some values of x, and $f_s(x) \neq SR(x)$, and R_I correctly returns YES.

(c) Because we have shown through a diagonal argument that no recognizer can exist for the HALT set, and because by (b) if a recognizer exists for IN-EQ-PRIM then it could be used to build a recognizer for the halt set, there can be no recognizer for IN-EQ-PRIM.