CECS 419-519, Writing Assignment 2, Due 8:00 am, February 9th, 2024, Dr. Ebert

Directions

Make sure name is on all pages. Order pages (front and back) so that solutions are presented in their original numerical order. **Please no staples or folding of corners (your papers won't get lost). A paper clip is OK** Show all necessary work and substantiate all claims. Avoid plagiarism.

Problems

1. An instance of the `Zero` decision problem is a Gödel number x and the problem is to decide if P_x outputs 0 on every input. Let $d_{\text{Zero}}(x)$ be the decision function for `Zero` and consider the following antagonist function $g(x)$ which diagonalizes against all computable functions in an attempt to contradict the assumption that $d_{\text{Zero}}(x)$ is total computable.

$$g(x) = \begin{cases} 1 & \text{if } d_{\text{Zero}}(x) = 1 \\ 0 & \text{otherwise} \end{cases}$$

   Has the antagonist succeeded? In other words, based on g's definition, may we conclude that $d_{\text{Zero}}(x)$ cannot be total computable? Explain. (20 pts)

   Solution. Let e be the Gödel number for a program P_e that computes $g(x)$ and consider $g(e)$. Case 1: $g(e) = 1$. This would mean $d_{\text{Zero}}(e) = 1$ which is false since $g(e) \neq 0$ for some inputs (since there do in fact exist programs that always output 0). Now suppose $g(e) = 0$. This means that $g(x)$ does not always output 0, which is correct! Therefore, g's definition does not result in a contradiction. □

2. An instance of the problem `Identity` decision problem is a Gödel number x, and the problem is to decide if $P_x(i) = i$ for all $i \geq 0$. Prove that the `Zero` decision problem is Turing reducible to the `Identity` decision problem. Do this by writing a program that decides the `Zero` decision problem and is able to make calls to the function $\text{query}_{\text{ID}}(x)$ which returns 1 iff x is a positive instance of `Identity`. Conclude that the `Identity` decision problem is undecidable (since the `Zero` problem was proved undecidable in lecture). (20 pts)

   Solution. Consider the following function $f(x, y)$. On inputs x and y, simulate $P_x(y)$. If $P_x(y) = 0$, then output y. Otherwise, loop forever. By the Church-Turing thesis we know that $f(x, y)$ is URM computable. Moreover, by the SMN-Theorem, there is a total computable function $h(x)$ for which $\phi_{h(x)}(y) = f(x, y)$. Moreover, notice that $\phi_{h(x)}(y)$ is the identity function iff $d_{\text{Zero}}(x) = 1$. Therefore, `Zero` is Turing reducible to `ID` since $d_{\text{Zero}}(x) = 1$ iff $d_{\text{ID}}(h(x)) = 1$. This is actually a mapping reduction from `Zero` to `ID` via the function $h(x)$. □

3. Consider the following proposal for solving the equation $\phi_e(y) = f(y, e)$. In other words, the goal is to write a URM-program P that computes the unary function $f(y)$ and where P is able

to make references to its own Gödel number e. To do this, the programmer assumes that at run time e will be placed in register R_2. Therefore, the programmer treats R_2 as a "read only" register and reads from R_2 whenever a computation involving e is desired. Prove or disprove: when using the above method, the equation $\phi_e(y) = f(y, e)$ does in fact hold for all inputs y and all Gödel numbers e so long as P_e does not make use of the instructions: $Z(2)$, $S(2)$, and $T(i, 2)$ for any $i \neq 2$. (20 pts)

Solution. Consider the function $f(y, x) = x$ which is computed by the program $P = T(2, 1)$ which has Gödel number

$$2^{\beta(T(2,1))} - 1 = 2^6 - 1 = 63.$$

Then $P_{63}(y) = 0$ for all y. However, $f(y, 63) = 63$, and so $\phi_e(y) \neq f(y, e)$ for all y. The problem is that $e = 63$ works very differently as a unary function than it does as a binary function. As a unary function, the value of register 2 is always initialized to 0. So, at run time the unary function being computed is *not* $\phi_{63}(y)$ which returns 0 rather than the desired 63.

4. An instance of the decision problem `One` is a Gödel number x, and the problem is to decide if function ϕ_x equals the `one` function, i.e. the function that outputs 1 on every input. Consider the decider function

$$d_{\text{one}}(x) = \begin{cases} 1 & \text{if } \phi_x(y) = 1 \text{ for all inputs y} \\ 0 & \text{otherwise} \end{cases}$$

   (a) Evaluate $d_{\text{one}}(x)$ for each of the following Gödel number's x. Explain your reasoning. (3 pts each)

      i. $x = e_1$, where e_1 is the Gödel number of the program $P = S(2), T(2, 1), J(1, 2, 1)$

      ii. $x = e_2$, where e_2 is the Gödel number of the program $P = Z(1), S(1), S(2), J(1, 2, 1)$.

      iii. $x = e_3$, where e_3 is the Gödel number of the program that computes $d_{\text{one}}(x)$ (assuming it is URM computable).

   Solution. $d_{\text{one}}(e_1) = 0$ since P_{e_1} loops forever, but $d_{\text{one}}(e_2) = 1$, since P_{e_2} always halts with 1 in R_1. $d_{\text{one}}(e_3) = 0$ since $d_{\text{one}}(x) = P_{e_3}(x)$ sometimes outputs 0.

   (b) Prove that $d_{\text{one}}(x)$ is not URM computable. In other words, there is no URM program that, on input x, always halts and either outputs 1 or 0, depending on whether or not ϕ_x equals the `one` function. Do this by writing a program P that uses $d_{\text{one}}(x)$ and makes use of the `self` programming concept. Then explain why P creates a contradiction. (15 pts)

   Solution. Consider a program P which, on input y, computes $d_{\text{one}}(\texttt{self})$. If $d_{\text{one}}(\texttt{self}) = 1$, then P returns 0, contradicting the result of $d_{\text{one}}(\texttt{self})$ (since $0 \neq 1$). Otherwise, in case $d_{\text{one}}(\texttt{self}) = 0$, then P returns 1, again contradicting the result of $d_{\text{one}}(\texttt{self})$ since in this case P would compute the constant function that always outputs 1.

5. Consider a model of computation $\mathcal{M}$ that is similar to the URM model, but has more instructions and hence allows for more time-efficient computations. For example, there exists a univeral $\mathcal{M}$-program P_U which, for any program P_e, is capable of simulating a single step of $P_e(x)$ in at most $c_1 \log(x)$ steps, for x sufficiently large (in other words, the bound is guaranteed only for $x \geq k$, for some constant $k \geq 1$. Moreover, an instance of the `Bounded Halting` problem is a pair (e, n) where e is the Gödel number of an $\mathcal{M}$-program, and n is some nonnegative integer. The problem is to decide if there is some $x \in \{2^n, 2^n + 1, \ldots, 2^{n+1} - 1\}$ for which P_e halts on input x in less than or equal to n^4 steps. Prove that there is no function $g(e, n)$ that can

decide the `Bounded Halting` problem within $c_2 n^2$ steps, for n sufficiently large. Hint: use the `self` programming concept. This problem shows that Kleene's 2nd Recursion Theorem also has applications to computational complexity theory! (25 pts)

Solution. Consider the following program P. On input x, compute the n for which $x \in \{2^n, 2^n+1, \ldots, 2^{n+1}-1\}$. Simulate g on inputs $e = \texttt{self}$ and n. Note: this simulation requires $S(n) = (c_2 n^2)(c_1(\log n + c'))$ steps, where c' is the size of P (since we are simulating a program G that computes g, the program has two inputs $\gamma(P)$ and n and so we must consider the size of both inputs when computing the cost of simulating each step of G). Notice that, for sufficiently large n, $S(n) = o(n^4)$, and so, after simulating $g(e, n)$, P may still execute additional steps. So, assuming n is sufficiently large, suppose $g(\texttt{self}, n) = 0$. In this case P immediately returns 1 which contradicts g, since P halts within n^4 steps on *all* inputs in the set $\{2^n, 2^n+1, \ldots, 2^{n+1}-1\}$. On the other hand, if $g(\texttt{self}, n) = 1$, then P loops forever and thus will never halt within n^4 steps for any $x \in \{2^n, 2^n+1, \ldots, 2^{n+1}-1\}$. This again is a contradiction. Note that this proof will continue to work so long as G's running time stays sufficiently below $n^4/\log(n + c')$.