CS 5000 Homework 11

Sally Devitry (A01980316)

Problem 1

Solution

There is an infinite amount of programs, that for any input, they have the same output. Take the following two programs, for example, that output the same thing.

$P_1 =$

1	$[A1]$	$X1 < -X1 + 1$

$P_2 =$

1	$[A1]$	$X1 < -X1 + 1$
2		$X1 < -X1$

P_1 and P_2 have the exact same output, but will have different program numbers. We can add an infinite amount of no ops or increments/decrements that reverse each other, and the program output will not change, but the program number will.

Problem 2

Solution

The function, $H(x)$ returns 1 if the program completes on itself. Otherwise, it will not complete on it's own input.
There are an infinite amount of programs that when you give its own program as input, and it will not compute. The halting problem, for example, if given it's own input, is not defined. You can add no ops to get infinite number of programs.
The following function meets the criteria of $H(x)$:

1	$[A1]$	$X1 < -X1 + 1$
2	$[B1]$	$IFX1! = 0 \quad GOTOA1$

Any amounts of no ops could be added to the above function to make it have a different program number.

Problem 3

Solution

$g(x_1, ...x_n)$ is primitive recursive and its output is given as input into the step function as the max number of steps that the program may halt on for the step function to return true.
The step function in this problem returns true if the program halts in $g(x_1, ...x_n)$ or fewer steps. The problem statement says that the step function returns true for all inputs. This means that $g(x_1, ...x_n)$ will return an appropriate upper bound on steps as the third argument to the step function.
If the step function is always returning true, then it is always halting. Halting on every input implies that it is computable.

Problem 4

Solution

We can prove that $HALT(x, y)$ is not computable with a proof by contradiction.

Suppose that $HALT(x, y)$ is computable. Consider the following L program P:

$$\begin{array}{lllll} 1 & [A1] & IF & HALT(X1, X1) & GOTOA1 \\ 2 & & Y < -Y + 1 & & \end{array}$$

P is a valid L program, it considers X1 to be a program number(Godel number).
If X1 halts on its own Godel number as its only input, P goes into an infinite loop, so it's output is undefined.
If X1 does not halt on its own Godel number, P goes to line 2, and increments Y.
Since P is a valid L program, we can compute the program number.
Let $\#(P) = y_0$. Then,

1. $HALT(x, y_0) = 0$ if $HALT(x, x) = 1$
2. $HALT(x, y_0) = 1$ if $HALT(x, x) = 0$

We know that $(\forall x)HALT(x, y_0)$ iff $\neg HALT(x, x)$.
Let $x = y_0$. Then $HALT(y_0, y_0)$ iff $\neg HALT(y_0, y_0)$

We have a contradiction. No algorithm, given a program P in L, and an input x, can determine whether or not P will eventually halt on x.

Problem 5

Solution

The problem statement states that $f(x_1, ...x_n) \leq k$ for all inputs. This means that the function f is bounded. The function will always return on any input because it is total. If a function is total and bounded, then it is computable.

Problem 6

Solution

The remainder operation is primitive recursive. Accessing the nth prime is also primitive recursive. Using these operations and others proved to be primitive recursive, the function can easily be written with L, meaning it is primitive recursive.
The following predicate computes f(x) where p_i computes the ith prime:

$$\text{f(x)} = \begin{cases} x & \text{if } (\exists i)_{\leq x}(\exists j)_{\leq x}(p_i + p_j = n) = 1 \\ 0 & \text{otherwise} \end{cases}$$

All of the operations in the above predicate are primitive recursive, thus $f(x)$ is primitive recursive.