Exam rules: update

I received an update on the exam rules. On the front of the papers, it now says: “Full marks will be given for correct answers to THREE questions. If you opt to answer more than the specified number of questions, you should clearly identify which of your answers you wish to have marked. In cases where you have failed to identify the correct number of answers, the marker is only obliged to consider the answer in the order they appear up to the number of answers required.”

The diagonal argument, intuitively

The diagonal argument, in its most intuitive form, shows that for every enumeration $f_1, f_2, f_3, \ldots$ of functions $N \to A$ where the set A is non-empty, there is a function $g : N \to A$ which is not in that enumeration. This is shown by ensuring that $g(n)$ differs from $f_n(n)$ for every n, e.g.,

n	1	2	3	4	5	$\ldots$
$f_1(n)$	$1^{\perp}$	9	0	8	$\perp$	$\ldots$
$f_2(n)$	0	$\perp^{0}$	1	0	3	
$f_3(n)$	1	4	$9^{\perp}$	2	$\perp$	
$f_4(n)$	4	7	1	$7^{\perp}$	8	
$f_5(n)$	2	3	5	7	$2^{\perp}$	

The diagonal argument, formally

Formally, the proof that there is a function g not in the list $f_1, f_2, f_3, \ldots$ goes as follows:

Let $g : N \to A$ be any function such that $g(n) \neq f_n(n)$ for all n (such a g exists, because we have the choice as to whether $g(n)$ is undefined or an element of A). Claim: the function g is not in the list $f_1, f_2, f_3, \ldots$. The proof proceeds by contradiction. Suppose $g = f_k$ for some k. Then

$$\begin{aligned} f_k(k) &= g(k) && \text{because } g = f_k \\ &\neq f_k(k) && \text{because } g(n) \neq f_n(n) \text{ for all } n. \end{aligned}$$

Contradiction. So the claim is proved.

Different uses of the diagonal argument

1. To show that some set is not enumerable (e.g. the set of functions $N \to N$.)
2. To produce a function that is not contained in some enumerable set. (e.g., to show that the diagonal function is not computable.)

The diagonal function

Let $M_1, M_2, M_3, \ldots$ be an enumeration of Turing machines, and let $f_1, f_2, f_3, \ldots$ be the resulting enumeration of Turing-computable functions. The **diagonal function** d is

$$d(n) = \begin{cases} \text{undefined} & \text{if } f_n(n) \text{ is defined, i.e. if} \\ & M_n \text{ halts on input } n \text{ in a} \\ & \text{standard final configuration} \\ 1 & \text{otherwise.} \end{cases}$$

The function d is not in the list $f_1, f_2, f_3, \ldots$, because it is a special case of a g with $g(n) \neq f_n(n)$ for all n.

Exercise

The n-th **Fibonacci number** $\mathit{fib}(n)$ is determined by the following conditions:

$$\mathit{fib}(0) = \mathit{fib}(1) = 1$$
$$\mathit{fib}(n+2) = \mathit{fib}(n+1) + \mathit{fib}(n),$$

so we get the sequence of pairs $1, 1, 2, 3, 5, 8, 13, 21, \ldots$. Show that fib is primitive recursive. (Hint: consider the sequence $(1,1), (1,2), (2,3), (3,5), (5,8), (8,13), \ldots$, and recall that we have primitive recursive functions for encoding and decoding pairs.)

Solution (part 1/4)

The trouble is that fib has no straightforward definition by primitive recursion

$$fib(0) = 1$$
$$fib(y+1) = ?$$

because $fib(y)$ may depend on $fib(y)$ **and** $fib(y-1)$. To address this issue, we introduce a helper function h such that, essentially,

$$h(n) = (fib(n), fib(n+1)).$$

Solution (part 2/4)

The point about h is that it can be defined by primitive recursion, essentially as follows:

$$h(0) = (1, 1)$$
$$h(y+1) = (right(h(y)), left(h(y)) + right(h(y))).$$

But h is strictly speaking not a primitive recursive function, because primitive recursive functions must return natural numbers (not pairs of natural numbers.) We address this problem by using an encoding function $pair : N \times N \to N$ for pairs, e.g.,

$$pair(x, y) = 2^x \cdot 3^y.$$

Solution (part 3/4)

As we have see, the decodings that corresponds to $pair(x, y) = 2^x \cdot 3^y$ are

$$left(n) = lo(n, 2) \qquad \text{and} \qquad right(n) = lo(n, 3).$$

So the revised version of h is

$$h(0) = pair(1, 1)$$
$$h(y + 1) = pair(right(h(y)), left(h(y)) + right(h(y))).$$

Because the functions $+$, $pair$, $left$ and $right$ are primitive recursive, and h is defined by primitive recursion from those, h is primitive recursive.

Solution (part 4/4)

Finally, let

$$fib(y) = first(h(y)).$$

In other words, $fib = \mathrm{Cn}[first, h]$. So fib too is primitive recursive.

Exercise

Show that the function $\gcd(x, y)$ that returns the greatest common divisor of x and y is primitive recursive. (You can use the primitive recursive function $divides$ introduced earlier.)

Solution (part 1/3)

The greatest common divisor $\gcd(x, y)$ of x and y is the greatest i that divides both x and y — if such an i exists. Let R by the relation defined by

$$R(x, y, i) \quad \text{iff} \quad \mathit{divides}(i, x) \text{ and } \mathit{divides}(i, y).$$

where $\mathit{divides}$ is the primitive-recursive relation from the lecture that checks whether i divides x. Because $\mathit{divides}$ is primitive recursive, and primitive recursive relations are closed under "and", R too is primitive recursive.

Solution (part 2/3)

We have

$$\gcd(x,y) = \begin{cases} \text{the largest } i & \text{if such a } i \text{ exists} \\ \text{for which } R(x,y,i) & \\ ? & \text{otherwise} \end{cases}$$

The question mark comes into play only if $x = y = 0$, in which case might as well choose $? = 0$. Now the definition above looks like bounded maximization, except that the bound is missing.

Solution (part 3/3)

Evidently, a divisor i of x and y can't possibly be greater than $x + y$. So $x + y$ can be used as the bound. So $\gcd$ can be defined by bounded maximization as follows:

$$\gcd(x, y) = \begin{cases} \text{the largest } i \leq x + y & \\ \text{for which } R(x, y, i) & \text{if such a } i \text{ exists} \\ 0 & \text{otherwise.} \end{cases}$$

Because the relation R is primitive recursive, the function $\gcd$ is primitive recursive.