Exercises for Computability and Complexity, Spring 2019, Sheet 3 – Solutions

Please return your solutions in class, in the Tuesday lecture on February 26. You may (like always in this course) work in teams of 2.

Exercise 1. Consider the ultra-simple TM M with tape alphabet $\Sigma = \{0, 1, \sqcup, \triangleright\}$ and states $\{s, yes, no\}$ that has the following transition table:

$p \in K$	$\sigma \in \Sigma$	$\delta(q, \sigma)$
s	0	$(yes, 0, -)$
s	1	$(s, 1, \rightarrow)$
s	$\sqcup$	$(no, \sqcup, -)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$

What is the language $L(M)$ decided by M? Describe that language in plain English. Write a RAM program that decides the same language, in the following sense. Your RAM should compute a string function $f: \Sigma^* \rightarrow \{0,1\}$, such that $f(w) = 1$ iff w is in $L(M)$.

Solution. M decides the language of all words that contain at least one 0.

```
1    read
2    if c(0)= 0 goto 5
3    if c(0)= 1 goto 1
4    if c(0)= ⊔ goto 7
5    print 1
6    end
7    print 0
8    end
```

Problem 2 (easy) Show that the function $plus2: \mathbb{N} \rightarrow \mathbb{N}$, $plus2(n) = n + 2$, is primitive recursive.

Solution. From rule 2 we know that the successor function σ is p.r. By rule 4, using $r = 1$ and $m = 1$, and setting $f = g = \sigma$, we get that $h(n) = \sigma(\sigma(n))$ is p.r., but obviously h is just the desired function that adds 2.

Problem 3 (a little easier than medium) Show that the function $minus1: \mathbb{N} \rightarrow \mathbb{N}$, $minus1(n) = \max(0, n - 1)$ is primitive recursive. Hint: there is a very compact way of doing this, exploiting the fact that the primitive recursion scheme condition $h(n + 1, x) = g(n, h(n, x), x)$ has the "minus 1" operation already built in in the first arguments n +1 and n passed to h and g.

Solution. We use the rule 5. of primitive recursion with $r = 0$, put $f = \mathbf{0}$ and $g = p^2_1$. Then $h = minus1$.

Challenge problem (a bit demanding; optional) Show that the function $evensquare: \mathbb{N} \rightarrow \mathbb{N}$, defined by $evensquare(n) = n$ if n is uneven, else $evensquare(n) = n^2$, is primitive recursive. You may assume that $square: \mathbb{N} \rightarrow \mathbb{N}$, $square(n) = n^2$, is primitive recursive. Hint: you may

find it helpful to construct *evensquare* from a number of helper functions which you construct before assembling *evensquare*.

Solution. There are many ways to do this – and I would say this task has a "programming" flavour. Here is one way to do it.

We first procure a p.r. function $flip: \mathbb{N} \to \mathbb{N}$ that satisfies $flip(0) = 1$ and $flip(1) = 0$. Using the rule 5., put $r = 0$, $f = \sigma \circ \mathbf{0}$ (i.e., f is the constant 1), $g = p^2_1$. Then put $flip = h$.

Next, create a p.r. function $even: \mathbb{N} \to \mathbb{N}$, $even(n) = 1$ if n is even, else = 0. Again we do this by using the recursion rule 5., putting $r = 0$, $f = \sigma \circ \mathbf{0}$, and $g = flip \circ p^2_2$, which yields $h = even$.

Next, create a conditional function $cond: \mathbb{N}^3 \to \mathbb{N}$, which satisfies $cond(0, n, m) = m$ and $cond(1, n, m) = n$ (remark: *cond* can be seen as implementing an *if-then-else* operator.) Again using the rule 5., put $f = p^2_2$ and $g = p^5_4$ to obtain $cond = h$.

Finally, put $evensquare = cond(even(\cdot), square(\cdot), id(\cdot))$ -- which is an obvious shorthand for an application of rule 4.