CSE 4111/5111 —Winter 2014

Posted: Feb 26, 2014

Problem Set No. 1 —Solutions

This is not a course on *formal* recursion theory. Your proofs should be *informal* (but NOT sloppy), *completely argued*, correct, and informative (and if possible **short**). Please do not trade length for correctness or readability.

All problems are from the "Theory of Computation Text", or are improvisations that I completely articulate here.

(1) Dress up the primitive recursion

$$\begin{aligned} two(0) &= 1 \\ two(x+1) &= two(x) + two(x) \end{aligned} \tag{1}$$

to make it conform with the rigid primitive recursion schema.

Answer. We employ the same method we followed for $p = \lambda x.x \dot{-} 1$ in the text/class:

First define

$$TWO \stackrel{Def}{=} \lambda xy.two(x) \tag{2}$$

Then recast equations (1) with "dressing", so that the basis is a 1-argument function and the iterator is a 3-argument function (I omit some brackets in compositions for visual clarity; e.g., SZx rather than $S(Z(x))$):

$$\begin{aligned} TWO(0,y) &= SZy \\ TWO(x+1,y) &= add\Big(U^3_3\big(x,y,TWO(x,y)\big), U^3_3\big(x,y,TWO(x,y)\big)\Big) \end{aligned} \tag{3}$$

In (3) we employed $add = \lambda xy.x + y$ that we know (from class/text) is in $\mathcal{PR}$. Moreover, the "basis" is $H = \lambda y.SZy$ and the iterator is $G = \lambda xyz.add\Big(U^3_3\big(x,y,z\big), U^3_3\big(x,y,z\big)\Big)$.

We get two from TWO by composition (identification of variables): $two = \lambda x.TWO(x,x)$ —or, in slow motion, $two = \lambda x.TWO\Big(U^1_1(x), U^1_1(x)\Big)$. □

From Section 2.12.

(2) Do problems 6, 11, 12, 19.

6. Prove that every finite set is primitive recursive.

Proof. Let $S = \{a_1, \ldots, a_n\}$. We want to show that the predicate $x \in S$ is in $\mathcal{PR}_*$. Well,

$$x \in S \equiv x = a_1 \vee x = a_2 \vee \ldots \vee x = a_n \tag{1}$$

Since $\lambda xy.x = y$ is in $\mathcal{PR}_*$, so is $\lambda x.x = y$, and we are done by closure of $\mathcal{PR}_*$ under $\vee$.

Important: The ... in (1) do not imply a "variable length formula", since the number of $\vee$ terms is fixed, independent of the input value x, that is. If we had a specific as opposed to "general" S like, say, $\{a, b, 3, 11\}$ we would have written (1) as $x \in S \equiv x = a \vee x = b \vee x = 3 \vee x = 11$ without ... $\square$

11. Prove that if we know that (1) g is primitive recursive; (2) $f(\vec{x}) \leq g(\vec{x})$, for all $\vec{x}$; and (3) $\lambda z\vec{x}.z = f(\vec{x})$ is in $\mathcal{PR}_*$, then f is primitive recursive.

Proof. This is the general case of the example $\lambda n.p_n$ that we know from class/text:

Condition (2) forces f to be total (since g is), for if $f(\vec{a}) \uparrow$ for some $\vec{a}$, then we cannot have (2) to hold: it requires both sides —$f(\vec{a})$ and $g(\vec{a})$— be defined as, say, c and d respectively, *and* to have $c \leq d$.

But then

$$f(\vec{x}) = (\mu y)_{\leq g(\vec{x})}(y = f(\vec{x}))$$

which proves the primitive recursiveness of f, since h given by $h(z, \vec{x}) = (\mu y)_{\leq z}(y = f(\vec{x}))$ in in $\mathcal{PR}$ by closure under $(\mu y)_{\leq z}$. f is obtained by substitution of $g(\vec{x})$ into the variable z in $h(z, \vec{x})$. $\square$

12. Are the conditions (1) and (2) above necessary in order to arrive to the same conclusion from just (3)?

As the case of the primitive recursive predicate $\lambda nxz.z = A_n(x)$ shows, we need to bound the "output" of our "f" by a primitive recursive function. This *bounding* is not available, as we know, in the case of $f = \lambda nx.A_n(x)$ and we also know that *this* f is not in $\mathcal{PR}$. So without conditions (1) and (2) in **11** above the result cannot go through in *all cases*. Necessary conditions!

19. Define

$$(\mathring{\mu} y)_{\le z} f(y, \vec{x}) \stackrel{\text{Def}}{=} \begin{cases} \min\{y : y \le z \wedge f(y, \vec{x}) = 0\} \\ 0, \text{ if the min does not exist} \end{cases}$$

Prove that $\mathcal{PR}$ is closed under $(\mathring{\mu} y)_{\le z}$.

Proof. The easiest thing to do is to piggy back on closure under $(\mu y)_{\le z}$ and composition! So,

$$(\mathring{\mu} y)_{\le z} f(y, \vec{x}) = \textbf{if } (\mu y)_{\le z} f(y, \vec{x}) \le z \textbf{ then } (\mu y)_{\le z} f(y, \vec{x}) \textbf{ else } 0$$

settles it! $\square$

(3) Write a "nice and clean" loop program which computes $\lambda x.\lfloor x/2 \rfloor$. The program must only allow instruction-types $X \leftarrow 0$, $X \leftarrow X + 1$, $X \leftarrow Y$ and **Loop** $X \ldots$ **end**. It must *not* nest the Loop-end instruction! It is required that you give a convincing general argument (*not* a "trace") as to why your program works as specified.

Answer. This is related to the $rem(x, 2)$ case we did in class.

We will set up a simultaneous recursion that will readily be translatable into a Loop program.

So let $f = \lambda x.\lfloor x/2 \lfloor$. We plot it as a sequence (of outputs) and also plot the same sequence shifted one position to the left; we call the corresponding function of the latter g.

$$\begin{aligned} f &= 0,0,1,1,2,2,3,3,4,4,\ldots \\ g &= 0,1,1,2,2,3,3,4,4,5,\ldots \end{aligned}$$

The simultaneous recursion, by inspection, is

$$\begin{aligned} f(0) &= 0 \\ g(0) &= 0 \end{aligned}$$

and

$$\begin{aligned} f(x+1) &= g(x) \\ g(x+1) &= f(x) + 1 \end{aligned}$$

The straightforward translation of the above recursion to a Loop program only needs us to save the $f(x)$ value in a temporary variable T before it is changed in the first recurrence equation.

So:

$$\begin{aligned}&F \leftarrow 0\\&G \leftarrow 0\\&\textbf{Loop}X\\&T \leftarrow F\\&F \leftarrow G\\&G \leftarrow T+1\end{aligned}$$

To make the last instruction "legal" we replace it by two:

$$\begin{aligned}&F \leftarrow 0\\&G \leftarrow 0\\&\textbf{Loop}X\\&T \leftarrow F\\&F \leftarrow G\\&T \leftarrow T+1\\&G \leftarrow T\end{aligned}$$

If the last Loop program is called M, then $f = M_F^X$. □

(4) Do problem 27, 28.

27. Show that the set K_0 defined as $\{\langle x, y\rangle : \phi_x(y)\downarrow\}$ is semi-computable.

Proof. The question is the same as saying "prove that the predicate $\langle x, y\rangle \in K_0$ —that is, $\phi_x(y)\downarrow$— is semi-computable".
Easy!

$$\phi_x(y)\downarrow \equiv (\exists z)T(x, y, z)$$

and we are done by the strong projection theorem since $T(x, y, z)$ is primitive recursive. □

28. Prove the "definition by *recursive cases*" theorem $\mathcal{R}$ and $\mathcal{P}$. The assumptions are:

(1) For the $\mathcal{R}$ case, all the f_i are in $\mathcal{R}$, while for the $\mathcal{P}$ case they all are in $\mathcal{P}$.

(2) In *both cases*, the R_i are in $\mathcal{R}_*$ (*recursive cases*).

The result to prove is that the defined f is in $\mathcal{R}$ and $\mathcal{P}$, respectively.

Proof. We have an f given from f_i and R_i as follows:

$$f(\vec{x}) = \begin{cases} f_1(\vec{x}) & \text{if } R_1(\vec{x}) \\ f_2(\vec{x}) & \text{if } R_2(\vec{x}) \\ \vdots & \vdots \\ f_k(\vec{x}) & \text{othw} \end{cases} \tag{1}$$

Since if-then-else is in $\mathcal{R}$ and hence $\mathcal{P}$, and both sets are closed under composition, we are done by rewriting (1) as

$$\begin{aligned} f(\vec{x}) = \quad & \text{if } R_1(\vec{x}) \text{ then } f_1(\vec{x}) \\ & \text{else if } R_2(\vec{x}) \text{ then } f_2(\vec{x}) \\ & \vdots \quad \vdots \\ & \text{else if } R_{k-1}(\vec{x}) \text{ then } f_{k-1}(\vec{x}) \\ & \text{else} f_k(\vec{x}) \end{aligned}$$

$\square$