Assignment#3 Key
1. Show prfs are closed under three-way mutual induction

Three-way mutual induction means that each induction step after calculating the base is computed using the previous value of the other function.

The formal hypothesis is:
Assume $g_1$, $g_2$, $g_3$, $h_1$, $h_2$, and $h_3$ are already known to be prf, then so are $f_1$, $f_2$, and $f_3$, where
\[
\begin{align*}
    f_1(x,0) &= g_1(x); \\
    f_1(x,y+1) &= h_1(f_1(x,y),f_2(x,y),f_3(x,y)) \\
    f_2(x,0) &= g_2(x); \\
    f_2(x,y+1) &= h_2(f_1(x,y),f_2(x,y),f_3(x,y)) \\
    f_3(x,0) &= g_3(x); \\
    f_3(x,y+1) &= h_3(f_1(x,y),f_2(x,y),f_3(s,y))
\end{align*}
\]

Proof is by construction
Three-Way Mutual Induction (Co-Recursion)

F will do all three computations in “parallel”

Define \(<z>_{31} = <z>_1; <z>_{32} = <<z>_2>_1; <z>_{32} = <<z>_2>_2

\[ F(x,0) = \langle g_1(x), g_2(x), g_3(x) \rangle \] // bases for all three

\[ F(x, y+1) = \langle h_1(<F(x,y)>_{31},<F(x,y)>_{32},<F(x,y)>_{33}), \]
\[ h_2(<F(x,y)>_{31},<F(x,y)>_{32},<F(x,y)>_{33}), \]
\[ h_3(<F(x,y)>_{31},<F(x,y)>_{32},<F(x,y)>_{33}) \rangle \]

F produces triples containing the values of f1, f2, and f3, in its first, second and third, component, respectively. The above shows F is a prf.

f1, f2, and f3 are then defined from F as follows:

\[ f_1(x,y) = <F(x,y)>_{31} \]
\[ f_2(x,y) = <F(x,y)>_{32} \]
\[ f_3(x,y) = <F(x,y)>_{33} \]

This shows that f1, f2, and f3 are also prf’s, as was desired.
2a. Show every infinite, re set is the range of a total recursive function that never repeats
Let $S$ be an arbitrary infinite re set. Furthermore, let $S$ be the domain of some partial recursive function $f_s$. Show that $S$ is the range of some total recursive function, call it $h_s$, that never repeats, i.e., $(h_s(x) = h_s(y) \text{ iff } x = y)$.

// if y=0, Search fails; if y>0 we check for next item never seen before
$h_s(y) = < \mu <x,t> \ STP(f_s, x, t ) \&\& \neg \text{Search}(x,y) >_1$

// I originally had a solution involving keeping a history, but I then realized that
// simple co-recursion works perfectly. Expand this to y=3 to see how it works.
Search(x, 0) = 0 // Fails because no prior or current values of $h_s$
Search(x, y+1) = Search(x, y) || x == $h_s(y)$
    // See if listed earlier than y-th or it is the y-th
2b. Version 2 (based on range of algorithm rather domain of procedure)

Let $S$ be an arbitrary infinite re set. Furthermore, let $S$ be the range of some total recursive function $f_s$. Show that $S$ is the range of some total recursive function, call it $h_s$, that never repeats a value ($h_s(x) = h_s(y)$ iff $x = y$).

// if $y=0$, Search fails; if $y>0$ we check for next item never seen before
$h_S(y) = f_S(\mu x \sim \text{Search}(f_S(x),y))$

// This again uses simple co-recursion. Expand this to $y=3$ to see how it works.
Search($x$, 0) = 0  // Fails because no prior or current values of $h_S$
Search($x$, $y+1$) = Search($x$, $y$) || $x$ == $h_S(y)$
    // See if listed earlier than $y$-th or it is the $y$-th
The Good about the Solutions

• Compact so easy to understand
  • Just need to see what Search does on a few sample cases
  • As this is Computability, efficiency is not important

• Code Encapsulation:
  • Neither $h_s$ nor Search needs to know how other is implemented

• Looking back, we might be embarrassed as CS people
  • CS folks always look for efficiency, even if just for self-pride
  • However, none of this is necessary; I just want you to think more broadly about alternatives after answering a posed question.
The Bad and the Ugly about the Solutions

• In neither case do we do short-circuit Boolean evaluations
  • This leads to wasteful evaluations
    • TRUE || anything = TRUE
    • FALSE && anything = FALSE

• In the case of $h_S$, we start the search from the beginning every time
  • There is no need to start earlier than one past where we ended previous one

• In the case of search, we keep recursing even if found true
  • Using short-circuit, we stop as soon as we find the first duplicate

• In the case of search, we might think we recompute $h_S(y)$ every time
  • Not so as $h_S(y-1), y>0$, is already available before we compute $\text{search}(x,y)$
  • The special case of $h_S(0)$ is based on constant false for $\text{search}(x,0)$
Short-Circuit Boolean Evaluation

• Might do something like the following in hopes the second value is not computed unless it appears on rhs

• Define |$ as
  • 0 |$ x = x
  • (y+1) |$ x = 1

• Define &$ as
  • 0 &$ x = 0
  • (y+1) &$ x = x
Lower Bound on $\mu$ for $h_S$

- $H_S(0) = \mu<x,t> \text{ STP}(f_s, x, t )$
- $H_S(y+1) = \mu<x,t> > H_S(y) \ [ \text{ STP}(f_s, x, t ) \ & \ ~\text{Search}(x,y+1)]$
- $h_S(y) = \langle H_S(y) \rangle_1$
Use Short-Circuit to Stop Recursion

// Fails because no prior or current values of $h_S$
Search(x, 0) = 0

// Use short-circuit with operands switched
// See if it is the y-th or is listed earlier than y-th
Search(x, y+1) = (x == $h_S(y)) \lor \neg \text{Search}(x, y)$