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 **g1, g2, g3, h1, h2,** and **h3** are already known to be prf, then so are **f1, f2,** and **f3,** where
f1(x,0) = g1(x); f1(x,y+1) = h1(f1(x,y),f2(x,y),f3(x,y))
f2(x,0) = g2(x); f2(x,y+1) = h2(f1(x,y),f2(x,y),f3(x,y))
f3(x,0) = g3(x); f3(x,y+1) = h3(f1(x,y),f2(x,y),f3(s,y))

Proof is by construction

Three-Way Mutual Induction (Co-Recursion)

F will do all three computations in “parallel”

Define $\langle z\rangle_{31} = \langle z\rangle_1$; $\langle z\rangle_{32} = \langle\langle z\rangle_2\rangle_1$; $\langle z\rangle_{32} = \langle\langle z\rangle_2\rangle_2$

$F(x,0) = \langle g1(x), g2(x), g3(x)\rangle$ // bases for all three

$$F(x, y+1) = \langle\, h1(\langle F(x,y)\rangle_{31},\langle F(x,y)\rangle_{32},\langle F(x,y)\rangle_{33}),$$
$$h2(\langle F(x,y)\rangle_{31},\langle F(x,y)\rangle_{32},\langle F(x,y)\rangle_{33}),$$
$$h3(\langle F(x,y)\rangle_{31},\langle F(x,y)\rangle_{32},\langle F(x,y)\rangle_{33})\,\rangle$$

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:

$f1(x,y) = \langle F(x,y)\rangle_{31}$

$f2(x,y) = \langle F(x,y)\rangle_{32}$

$f3(x,y) = \langle F(x,y)\rangle_{33}$

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 $\mathbf{f_s}$. Show that **S** is the range of some total recursive function, call it $\mathbf{h_s}$, that never repeats, i.e, $\mathbf{(h_s(x) = h_s(y)}$ iff $\mathbf{x = y)}$.

// if y=0, Search fails; if y>0 we check for next item never seen before

$\mathbf{h_S(y) = \langle\ \mu\langle x,t\rangle\ STP(f_s, x, t\ )\ \&\&\ {\sim}Search(x,y)\ \rangle_1}$

// 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 $\mathbf{h_S}$**

Search(x, y+1) = Search(x, y) || x == $\mathbf{h_S(y)}$

// 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 $\mathbf{f_s}$. Show that **S** is the range of some total recursive function, call it $\mathbf{h_s}$**,** that never repeats a value ($\mathbf{h_s(x) = h_s(y)}$ iff $\mathbf{x = y}$).

// if y=0, Search fails; if y>0 we check for next item never seen before

$\mathbf{h_S(y) = f_S(\mu\ x \sim Search(f_S(x),y))}$

// 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 $\mathbf{h_S}$**

Search(x, y+1) = Search(x, y) || x == $\mathbf{h_S(y)}$
// 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 $\mathbf{h_S}$ 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 $\mathbf{h_S}$, 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 $\mathbf{h_S(y)}$ every time
  - Not so as $\mathbf{h_S(y-1)}$, **y>0**, is already available before we compute **search(x,y)**
  - The special case of $\mathbf{h_S(0)}$ is based on constant false for **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 μ for h_S

- $H_S(0) = \mu\langle x,t\rangle\ STP(f_s, x, t)$
- $H_S(y+1) = \mu\langle x,t\rangle > H_S(y)\ [\ STP(f_s, x, t)\ \&\$\ {\sim}Search(x,y+1)]$

- $h_S(y) = \langle H_S(y)\rangle_1$

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)) |$ Search(x, y)