Precept 4: Proofs about Fuctional Programs

This precept will help familiarize you with material on proving things about programs. As part of your homework this week, you must read the online notes about proving things about programs. Refer to these notes to help you through this week's precept materials.

http://www.cs.princeton.edu/~dpw/courses/cos326-12/notes/reasoning.php

http://www.cs.princeton.edu/~dpw/courses/cos326-12/notes/reasoning-data.php

Part I

1. Consider the function tail:

```
let tail (xs: 'a list) : 'a list =
  match xs with hd :: tail -> tail
;;
```

Is tail a total function? No

Is tail [] a valuable expression? No

Is tail [3] a valuable expression? Yes

2. Consider safediv:

```
let safediv (nums : int * int) : int option =
  let (x,y) = nums in
  if y == 0 then None
  else Some (x/y)
;;
```

Is safediv a total function? Yes

Is safediv (1, 0) valuable? Yes

3. Consider the following type and function declarations

```
type form =
  Var of string
| And of form list

let rec free_var (f : form) =
  match f with
    Var s -> [s]
  | And fs -> free_vars fs

and free_vars (fs: form list) =
  match fs with
    [] -> []
  | f :: rest -> free_var f @ free_vars rest
;;
```

Is free_var total? Yes

Is free_vars total? Yes

4.

```
let rec f (x:int) =
  if x > 50 then 1 + f (x-1) else g x

and g (y: int) =
  if y > 0 then 1 else f (x-1)
;;
```

Is f total? No

Is g total? No

5.

```
let f x = … ;;

let g x =
  if f x then 1 else 0
;;
```

What do we need to know about f to know that g is total? f must be total.

Part II

Give justifications for each of the following equations using the equational rules given in the online notes. Whenever you need to use reflexivity, transitivity, symmetry, congruence, etc., say so.

let inc x = x + 1;;

(1) inc 3 == 4 — eval, math, transitivity _____

(2) inc 4 == 5 — eval, math, transitivity _____

(3) inc (inc 3) == 5 — (2), congruence, (3), transitivity_____

(4) fun x -> x + 1 == inc — syntactic sugar, substitution, transitivity_____

(5) for all values v, v + 1 == inc v — eval _____

(6) for all valuable expressions e, e + 1 == inc e — eval (since e valuable)_____

Part III

Consider the following code:

```
let multo (x:int option) (y:int option) =
  match (x,y) with
    (Some m, Some n) -> Some ((m + m)*n)
  | (_, _) -> None
;;
```

Prove the following equation holds, step by step, for all `o : int option`

```
multo o o == (match o with Some i -> Some (2*(i*i)) | None -> None)
```

Proof: (*Note: You can start top down, or you can start bottom up, or go from both ends to the middle*)

```
Proof is by cases on the structure of o.

case o = None:
   multo None None
== match (None, None) with (... | (_, _) -> None)            (eval)
== None                                                      (eval)
== match None with Some i -> Some (2*(i*i)) | None -> None   (eval, reverse)

case o = Some j
   multo (Some j) (Some j)
== match (Some j, Some j) with (Some m, Some n) -> Some (m+m)*n | None -> None) (eval)
== Some (j+j)*j                                                                 (eval)
== Some (2*(j*j))                                                               (math)
== match Some j with Some i -> Some (2*(i*i)) | None -> None)          (eval, reverse)

QED!
```

Part IV

Consider the following function.

```
let compose (f:'a -> 'b) (g:'b -> 'c) (x:'a) = g (f x);;
```

Prove using equational reasoning that for all n : int

```
   compose (fun x -> x * 2) (fun y -> y * 8) n
== compose (fun z -> z * 4) (fun w -> w * 4) n
```

Proof (put one reasoning step on each line with a justification):

(*again, recall you can start from the left-hand side and prove to the right; or start on the right-hand side and prove to the left or go from both sides and try to meet in the middle*)

```
   compose (fun x -> x * 2) (fun y -> y * 8) n
== (fun y -> y*8) ((fun x -> x*2) n)              (eval)
== (fun y -> y*8) (n*2)                           (eval)
== (n*2)*8                                        (eval,
                                                    since n*2 valuable)
== (n*4)*4                                        (math)
== (fun w -> w*4) (n*4)                           (eval, reverse)
== (fun w -> w*4) ((fun z -> z*4) n)              (eval, reverse)
== compose (fun z -> z * 4) (fun w -> w * 4) n    (eval, reverse)
```

Part V

Consider the functions double and half:

```
let rec double (xs: int list) : int list =
  match xs with
    [] -> []
  | hd :: rest -> hd::hd::double rest
;;

let rec half (xs: int list) : int list =
  match xs with
    [] -> []
  | [x] -> []
  | x::y::rest -> y::half rest
;;
```

(a) Disprove this conjecture: for all l, double(half l) == l.
(Rhetorical question: How does one disprove such a conjecture?)

A counter-example to the conjecture is l == [3] since:

double(half [3]) == double [] == []

and:

[] =/= [3]

(b) Prove that for all integer lists l. half (double l) == l.

```
Proof:  By induction on the structure of the list l:

case l = []
To show:  half (double []) == []

Proof:
   half (double [])
== half []                  (eval 2 steps)
== []                       (eval 2 steps)

case l = hd::tail
To show:  half (double (hd::tail)) == hd::tail
IH:  half (double tail) == tail

Proof:
   half (double (hd::tail))
== half (hd::hd::double tail)  (eval 2 steps)
== hd::(half(double tail))     (eval 2 step, since hd::hd::double tail valuable)
== hd::tail                    (IH)
```

Part VI

Ask in precept about solutions to this problem. (Do not post solutions to this problem online.)