Homework 3

Chirag Sachdev

Week 3

Abstract

This project is a part of HW3 of Assurance Foundations. The homework deals with integration of ML and HOL to LaTeX. The goal of this report is to show reproducibility which is the groundwork for credibility that I have done this on my own without any external help. Every Chapter demonstrates the following sections:

- Problem Statement
- Relevant Code
- Test Results

This project includes the following packages:

634format.sty A format style for this course

listings Package for displaying and inputting ML source code

holtex HOL style files and commands to display in the report

This document also demonstrates my ability to :

- Easily generate a table of contents,
- Refer to chapter and section labels

My skills and my professional details can be found at `https://www.linkedin.in/in/chiragsachdev`.

acknowledgments

I would gratefully acknowledge Dr. Shiu-Kai Chin and my other professors at Syracuse University and my Professors at Drexel University for being the wonderful mentors they are to guide me through my journey of obtaining a Master's Dregree.

Contents

Chapter 1

Executive Summary

Some requirements haven't been met Specifically,

Chapter 4 Excercise 5.3.4

Chapter 2

Excercise 4.6.3

2.1 Problem statement

In ML, define functions:

1. A function that takes a 3-tuple of integers *(x, y, z)* as input and returns the value corresponding to the *sum x + y + z*.
2. A function that takes two integer inputs x and y (where x is supplied first followed by y) and returns the boolean value corresponding to *x ¡ y*.
3. A function that takes two strings s1 and s2 (where s 1 is supplied first followed by s 2) and concatentates them, where "" denotes string concatenation. For example, "*Hi*" " *there*" results in the string "*Hi there*".
4. A function that takes two lists list 1 and list 2 (where list 1 comes first) and appends them, where @ denotes list append. For example *[true,false]* @ *[false, false, false]* results in the list *[true,false,false,false,false]*.
5. A function that takes a pair of integers (x, y) and returns the larger of the two values. You note that the conditional statement *if condition then a else b* returns *a* if condition is true, otherwise it returns *b*.

For each of the functions , you will define two ML functions, (1) the first using *fn* and *val* to define and name the function, and (2) the other using *fun* to define and name the function. Make sure you use pattern matching. For example, suppose the function is x.(y.2x + y). We would define in ML

1. *val funEx1 = (fn x =¿ (fn y =¿ 2*x + y))*, and
2. *fun funEx2 x y = 2*x + y*

2.2 Relevant Code

```
val funA1 = (fn (x,y,z) => x+y+z);
fun funA2 (x,y,z) = x+y+z;

val funB1 = (fn x => (fn y => (if x<y then true else false)));
fun funB2 x y = if x<y then true else false;

val funC1 = (fn s1 => (fn s2 =>(s1^s2)));
fun funC2 s1 s2 = s1^s2;

val funD1 = (fn l1 => (fn l2 => l1@l2));
fun funD2 l1 l2 = l1@l2;

val funE1 = (fn (x,y) => (if x>y then x else y));
fun funE2 (x,y) = (if x>y then x else y);
```

2.3 Test Case

```
(*****************************************************************************)
(* Test Cases
*)
(*****************************************************************************)
val testListA = [(1,2,3),(4,5,6),(7,8,9)]

val outputsA = map funA2 testListA

val testResultA = test463A funA1 funA2 testListA

val testListB = [(0,0),(1,2),(4,3)]

val outputsB = map (f2P funB1) testListB

val testResultB = test463B funB1 funB2 testListB

val testListC = [("Hi","␣there!"),("Oh␣","no!"),("What","␣the␣...")]

val outputsC = map (f2P funC1) testListC

val testResultC = test463B funC1 funC2 testListC

val testListD1 = [([0,1],[2,3,4]),([],[0,1])]
val testListD2 = [([true,true],[])]

val outputsD1 = map (f2P funD1) testListD1
val outputsD2 = map (f2P funD2) testListD2

val testResultD1 = test463B funD1 funD2 testListD1
val testResultD2 = test463B funD1 funD2 testListD2

val testListE = [(2,1),(5,5),(5,10)]

val sampleResultE = map funE1 testListE

val testResultE = test463A funE1 funE2 testListE
```

1

```
------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------
> > > > > > # # # # # # # # # # val test463A = fn: ('a -> ''b) -> ('a -> ''b) -> 'a list -> bool
> > # # # # # # # # # # # val f2P = fn: ('a -> 'b -> 'c) -> 'a * 'b -> 'c
val test463B = fn:
   ('a -> 'b -> ''c) -> ('a -> 'b -> ''c) -> ('a * 'b) list -> bool
> >
*** Emacs/HOL command completed ***

> val funA1 = fn: int * int * int -> int
> val funA2 = fn: int * int * int -> int
> > # # # # # val outputsA = [6, 15, 24]: int list
val testListA = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]: (int * int * int) list
val testResultA = true: bool
>
Process HOL finished

------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------
> > > >> > # # # # # # # # # # val test463A = fn: ('a -> ''b) -> ('a -> ''b) -> 'a list -> bool
> > # # # # # # # # # # # val f2P = fn: ('a -> 'b -> 'c) -> 'a * 'b -> 'c
val test463B = fn:
   ('a -> 'b -> ''c) -> ('a -> 'b -> ''c) -> ('a * 'b) list -> bool
> >
*** Emacs/HOL command completed ***

> val funB1 = fn: int -> int -> bool
> val funB2 = fn: int -> int -> bool
> >
> # # # # # val outputsB = [false, true, false]: bool list
val testListB = [(0, 0), (1, 2), (4, 3)]: (int * int) list
val testResultB = true: bool
>
Process HOL finished


------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------
> > > > > > # # # # # # # # # # val test463A = fn: ('a -> ''b) -> ('a -> ''b) -> 'a list -> bool
> > # # # # # # # # # # # val f2P = fn: ('a -> 'b -> 'c) -> 'a * 'b -> 'c
val test463B = fn:
   ('a -> 'b -> ''c) -> ('a -> 'b -> ''c) -> ('a * 'b) list -> bool
> > >
*** Emacs/HOL command completed ***

> val funC1 = fn: string -> string -> string
> val funC2 = fn: string -> string -> string
>
> # # # # val outputsC = ["Hi there!", "Oh no!", "What the ..."]: string list
val testListC = [("Hi", " there!"), ("Oh ", "no!"), ("What", " the ...")]:
   (string * string) list
val testResultC = true: bool
>
Process HOL finished

------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------
> > > > > > # # # # # # # # # # val test463A = fn: ('a -> ''b) -> ('a -> ''b) -> 'a list -> bool
> > # # # # # # # # # # # val f2P = fn: ('a -> 'b -> 'c) -> 'a * 'b -> 'c
val test463B = fn:
   ('a -> 'b -> ''c) -> ('a -> 'b -> ''c) -> ('a * 'b) list -> bool
> >
*** Emacs/HOL command completed ***

> # val funD1 = fn: 'a list -> 'a list -> 'a list
> val funD2 = fn: 'a list -> 'a list -> 'a list
> > # # # # # # # # # val outputsD1 = [[0, 1, 2, 3, 4], [0, 1]]: int list list
val outputsD2 = [[true, true]]: bool list list
val testListD1 = [([0, 1], [2, 3, 4]), ([], [0, 1])]:
   (int list * int list) list
val testListD2 = [([true, true], [])]: (bool list * 'a list) list
val testResultD1 = true: bool
```

Chapter 3

Excercise 4.6.4

3.1 Problem statement

In ML, define a function *listSquares* that when applied to the empty list of integers returns the empty list, and when applied to a non-empty list of integers returns a list where each element is squared. For example, *listSquares [2,3,4]* returns *[4,9,16]*. Define the function using a let expression in ML.

3.2 Relevant Code

```
fun listSquares xlist=
let
fun square x =x*x
(*fun outList [] = []   | outList (y::ys) = (square y)::(outList ys) *)
in
(map square xlist)
end;
```

3.3 Test Case

2

```
------------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------------
> > > > # # # # # # val listSquares = fn: int list -> int list
> # # val testList = [1, 2, 3, 4, 5]: int list
val testResults = [1, 4, 9, 16, 25]: int list
>
Process HOL finished
```

Chapter 4

Excercise 5.3.4

4.1 Problem statement

Define a function *Filter* in ML, whose behavior is identical to *filter*. Note: you cannot use filter in the definition of *Filter*. However, you can adapt the definition of *filter* and use it in your definition. Show test cases of your function returning the expected results by comparing the outputs of both *Filter* and *filter*.

4.2 Relevant Code

```
fun less_than5 xlist=
let
val templist = []:int list
val helper = 1
fun lessthan x = (if(x<5)then (templist@[x]) else (templist@([]:in)))
```

4.3 Test Case

The section did not meet the requirements.

Chapter 5

Excercise 5.3.5

5.1 Problem statement

Define an ML function *addPairsGreaterThan n list*, whose behavior is defined as follows: (1) given an integer n, and (2) given a list of pairs of integers *list, addPairsGreaterThan n list* will return a list of integers where each element is the sum of integer pairs in *list* where both elements of the pairs are greater than n.

5.2 Relevant Code

```
fun addpair (x,y)=x+y
fun Filter n xlist = filter(fn (x,y) => y>n andalso x>n ) xlist
fun addlist xlist = map addpair xlist
fun addPairsGreaterThan y xlist = (addlist (Filter y xlist))
```

5.3 Test Case

3

```
------------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------------
> > > > # val addpair = fn: int * int -> int
> # val Filter = fn: int -> (int * int) list -> (int * int) list
> # val addlist = fn: (int * int) list -> int list
> # val addPairsGreaterThan = fn: int -> (int * int) list -> int list
> val it = [5, 9]: int list
> >
Process HOL finished
```

Appendix A

Source code for Ex 4.6.3

```
(*****************************************************************************)
(* Exercise 4.6.3                                                            *)
(* Author: Chirag Sachdev                                                    *)
(* Date: February 5, 2019                                                    *)
(*****************************************************************************)

(*****************************************************************************)
(* Test functions you will need.                                             *)
(*                                                                           *)
(*                                                                           *)
(*****************************************************************************)

fun test463A f1 f2 inList =
let
 val list1 = map f1 inList
 val list2 = map f2 inList
in
 foldr
 (fn (x,y)  => (x andalso y))
 true
 (ListPair.map (fn (x,y) => x = y) (list1,list2))
end;

fun f2P f (x,y) = f x y

fun test463B f1 f2 inList =
let
 val list1 = map (f2P f1) inList
 val list2 = map (f2P f2) inList
in
 foldr
 (fn (x,y)  => (x andalso y))
 true
 (ListPair.map (fn (x,y) => x = y) (list1,list2))
end;


(**********)
(* Part A *)
(**********)
```

```
(* ========================================================================== *)
(*                                                                            *)
(* Add your code for funA1 and funA2 here.                                    *)
(*                                                                            *)
(* ========================================================================== *)

val funA1 = (fn (x,y,z) => x+y+z);
fun funA2 (x,y,z) = x+y+z;

(* Test cases*)
val testListA = [(1,2,3),(4,5,6),(7,8,9)]

val outputsA = map funA2 testListA

val testResultA = test463A funA1 funA2 testListA

(**********)
(* Part B *)
(**********)

(* ========================================================================== *)
(*                                                                            *)
(* Add your code for funB1 and funB2 here.                                    *)
(*                                                                            *)
(* ========================================================================== *)
val funB1 = (fn x => (fn y => (if x<y then true else false)));
fun funB2 x y = if x<y then true else false;


(* Test cases*)
val testListB = [(0,0),(1,2),(4,3)]

val outputsB = map (f2P funB1) testListB

val testResultB = test463B funB1 funB2 testListB

(**********)
(* Part C *)
(**********)

(* ========================================================================== *)
(*                                                                            *)
(* Add your code for funC1 and funC2 here.                                    *)
(*                                                                            *)
(* ========================================================================== *)

val funC1 = (fn s1 => (fn s2 =>(s1^s2)));
fun funC2 s1 s2 = s1^s2;


(* Test cases*)
val testListC = [("Hi","␣there!"),("Oh␣","no!"),("What","␣the␣...")]
```

```
val outputsC = map (f2P funC1) testListC

val testResultC = test463B funC1 funC2 testListC


(**********)
(* Part D *)
(**********)

(* ========================================================================= *)
(*                                                                           *)
(* Add your code for funD1 and funD2 here.                                   *)
(*                                                                           *)
(* ========================================================================= *)

val funD1 = (fn l1 => (fn l2 => l1@l2));
fun funD2 l1 l2 = l1@l2;


(* Test cases*)
val testListD1 = [([0,1],[2,3,4]),([],[0,1])]
val testListD2 = [([true,true],[])]

val outputsD1 = map (f2P funD1) testListD1
val outputsD2 = map (f2P funD2) testListD2

val testResultD1 = test463B funD1 funD2 testListD1
val testResultD2 = test463B funD1 funD2 testListD2

(**********)
(* Part E *)
(**********)

(* ========================================================================= *)
(*                                                                           *)
(* Add your code for funE1 and funE2 here.                                   *)
(*                                                                           *)
(* ========================================================================= *)

val funE1 = (fn (x,y) => (if x>y then x else y));
fun funE2 (x,y) = (if x>y then x else y);

val testListE = [(2,1),(5,5),(5,10)]

val sampleResultE = map funE1 testListE

val testResultE = test463A funE1 funE2 testListE
```

Appendix B

Source code for Ex 4.6.4

```
(*************************************************************************)
(* Exercise 4.6.3                                                        *)
(* Author: Chirag Sachdev                                                *)
(* Date: February 5, 2019                                                *)
(*************************************************************************)

(* ===================================================================== *)
(*                                                                       *)
(* Your code for listSquares here                                        *)
(*                                                                       *)
(* ===================================================================== *)

fun listSquares xlist=
let
fun square x =x*x
(*fun outList [] = []   | outList (y::ys) = (square y)::(outList ys) *)
in
(map square xlist)
end;
val testList = [1,2,3,4,5]

val testResults = listSquares testList
```

Appendix C

Source code for Ex 5.3.4

```
(*************************************************************************)
(* Exercise 4.6.3                                                        *)
(* Author: Chirag Sachdev                                                *)
(* Date: February 5, 2019                                                *)
(*************************************************************************)

(* ===================================================================== *)
(*                                                                       *)
(* Your code for Filter here                                             *)
(*                                                                       *)
(* ===================================================================== *)

fun less_than5 xlist=
let
val templist = []:int list
val helper = 1
fun lessthan x = (if(x<5)then (templist@[x]) else (templist@([]:in)))

in
(map lessthan xlist)
end;

val testResults = Filter (fn x => x < 5) [1,2,3,4,5,6,7,8,9]
```

Appendix D

Source code for Ex 5.3.5

```
(*************************************************************************)
(* Exercise 4.6.3                                                        *)
(* Author: Chirag Sachdev                                                *)
(* Date: February 5, 2019                                                *)
(*************************************************************************)

(* ===================================================================== *)
(*                                                                       *)
(* Your code for addPairsGreaterThan here                                *)
(*                                                                       *)
(* ===================================================================== *)

fun addpair (x,y)=x+y
fun Filter n xlist = filter(fn (x,y) => y>n andalso x>n ) xlist
fun addlist xlist = map addpair xlist
fun addPairsGreaterThan y xlist = (addlist (Filter y xlist))



addPairsGreaterThan 0 [(0,1),(2,0),(2,3),(4,5)];
```