Project 3

Jinhao Wei

Feburary 1, 2019

Abstract

This report provides solutions to problems in Project 3. Project 3 is a set of problems that are involved with ML programming. Within this problem set, higher order functions in ML are applied. For each of the problems, our report includes:

- Problem Statement
- Relevant Code
- Test Cases, Transcripts and Explanations

Acknowledgments: This project report follows the format of document *project2ExampleReport.pdf* provided by Professor **Shiu-Kai Chin**. Also, this report used package *634format.sty*, *holtex* and *holtexbasic* for formatting's purpose.

Contents

Chapter 1

Executive Summary

All requirements for this project are satisfied Specifically,

Report Contents
This report consists of:

Reproducibility in ML and LaTeX
All ML and LaTeX source files compile well on the environment provided by this course.

Chapter 2

Exercise 4.6.3

2.1 Problem Statement

In this problem, we will define several ML functions in two different ways.

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 => (x < y) );
fun funB2 x y = (x<y);

val funC1 = fn x => (fn y => (x^y));
fun funC2 x y = x^y;

val funD1 = fn x => (fn y => (x@y));
fun funD2 x y = x@y;

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 Cases, Execution Transcripts and Explanations

According to the requirements, we should run several test cases in HOL window. Before we run the test cases, let's introduce some test functions. We will begin from *test463A* and *test463B*.

2.3.1 Function test463A and test463B

test463A is a function that compares the running result of *map f1 inList* and *map f2 inList*. If the two results have the same value, then the return value of *test463A* is *True*, otherwise *False* will be returned.

```
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;
```

test463B is a function that compares the running result of *map (f2P f1) inList* and *map (f2P f2) inList*. If the two results are the same, then *True* will be returned, otherwise the return value will be *False*.

```
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;
```

What is different from *test463A* and *test463B* is that, the parameters that *test463A* accepts may vary, depending on how *f1* and *f2* are defined, while *test463B* will only accept a list of int*int, given the definition of *f2P*.

Now we are ready for our test cases:

2.3.2 Question 4.6.3.A

In this section, we will introduce two functions *funA1* and *funA2*, each of them will take a tuple of three integers and return the sum of them.

```
(*********)
(* 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;

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

val outputsA = map funA2 testListA

val testResultA = test463A funA1 funA2 testListA
```

If we send the above region to HOL, we will have the transcript as below. From the transcript we can see that the return value *testResultA* is *True*. That means *funA1* and *funA2* had generated the same value from input list *testListA*, therefore we could know that *funA1* and *funA2* are equivalent.

```
> > # # # # # # # # # val test463A = fn: ('a -> ''b) -> ('a -> ''b) -> 'a list -> bool        1
> > # # # # # # # # # # # val f2P = fn: ('a -> 'b -> 'c) -> 'a * 'b -> 'c
val test463B = fn:
   ('a -> 'b -> ''c) -> ('a -> 'b -> ''c) -> ('a * 'b) list -> bool
> 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
>
```

2.3.3 Question 4.6.3.B

For the second part,

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

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

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

val outputsB = map (f2P funB1) testListB

val testResultB = test463B funB1 funB2 testListB
```

From the code above we can see that *funB1* and *funB2* are two functions that compares two integers. If the first interger is smaller than the second integer, then the return value is *True*, otherwise the return value is *False*.

If we send the above region to HOL, we will have the transcript as below. From the return value of *testResultB* we could know that *funB1* and *funB2* are identical.

```
> > > # # # # val outputsA = [6, 15, 24]: int list                         2
val testListA = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]: (int * int * int) list
val testResultA = true: bool
> 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
val it = (): unit
```

2.3.4 Question 4.6.3.C

From the code below we know that *funC1* and *funC2* are both functions that receive two strings, concatentate the second string after the first string, and return the concatentation.

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

(* ================================================================ *)
(*                                                                  *)
(* Add your code for funC1 and funC2 here.                          *)
(*                                                                  *)
(* ================================================================ *)
val funC1 = fn x => (fn y => (x^y));
fun funC2 x y = x^y;

val testListC = [("Hi"," there!"),("Oh ","no!"),("What"," the ...")]
val outputsC = map (f2P funC1) testListC

val testResultC = test463B funC1 funC2 testListC
```

If we put the above region to HOL, we will see transcript as below:

```
val funC1 = fn: string -> string -> string                                    3
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
val it = (): unit
```

The return value of *testResultC* is true, therefore we could know that *funC1* AND *funC2* are identical.

2.3.5 Question 4.6.3.D

From the code below we know that *funD1* and *funD2* are two functions that receive two lists, append the second list to the end of the first list, and return the appended list as result.

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

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

val funD1 = fn x => (fn y => (x@y));
fun funD2 x y = x@y;

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
```

By putting the above region to HOL, we have the transcript as below. Both *testResultD1* and *testResultD2* are returned with value *true*. Therefore we know that we have made *funD1* and *funD2* identical.

```
> val funD1 = fn: 'a list -> 'a list -> 'a list                               4
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
val testResultD2 = true: bool
val it = (): unit
>
*** Emacs/HOL command completed ***
```

2.3.6 Question 4.6.3.E

funE1 and *funE2* are functions that both return the greater value in a interger tuple.

```
(*********)
(* 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
```

By sending the above region to HOL we receive the transcript as below. From the return value of *testResultE* we could know that *funE1* and *funE2* are identical.

```
> val funE1 = fn: int * int -> int                                    5
val funE2 = fn: int * int -> int
val sampleResultE = [2, 5, 10]: int list
val testListE = [(2, 1), (5, 5), (5, 10)]: (int * int) list
val testResultE = true: bool
val it = (): unit
>
*** Emacs/HOL command completed ***
```

Chapter 3

Exercise 4.6.4

3.1 Problem Statement

This problem asks us to define a function named *listSquares*, which returns a list of the squares of each item in the input list. We are required to use *let* expression in ML.

3.2 Relevant Code

```
fun listSquares  [] = []
    | listSquares (x::xs) =
      let
        fun square x = x*x
      in
        (square x) :: listSquares xs
      end;
```

3.3 Test Cases, Execution Transcripts and Explanations

3.3.1 Test Cases

```
(*****************************************************************************)
(* Exercise 4.6.4                                                             *)
(* Author: Shiu-Kai Chin                                                      *)
(* Date: 20 September 2015                                                    *)
(*****************************************************************************)

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

fun listSquares  [] = []
    | listSquares (x::xs) =
      let
        fun square x = x*x
      in
        (square x) :: listSquares xs
      end;


val testList = [1,2,3,4,5]
val testResults = listSquares testList
```

3.3.2 Execution Transcripts

```
> # # # # # # 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
>
```

1

3.3.3 Explanations

From the definition in 3.2 we could see that, when the received parameter is an empty list *[]*, then the return value is also a *[]*. When the parameter is a non-empty list, then we use a *let* expression to define a function named *square* to return the square of a integer value. Later, we apply this function *square* in our recursive definition of *listSquares*. So we can return the correct result.

Chapter 4

Exercise 5.3.4

4.1 Problem Statement

In this problem, we will manually define a function *Filter*, which is identical to *filter*.

4.2 Relevant Code

```
fun Filter func [] = []
        |Filter func (x::xs) = if (func x) then (x :: Filter func xs)
           else (Filter func xs)
```

4.3 Test Cases, Execution Transcripts and Explanations

4.3.1 Test Cases

According to the requirement, we will run test cases in both 5-3-4Test.sml and in exercise 5.3.3.

```
(***********************************************************************)
(* Exercise 5.3.4                                                      *)
(* Author: Shiu-Kai Chin                                               *)
(* Date: 20 September 2015                                             *)
(***********************************************************************)

(* =================================================================== *)
(*                                                                     *)
(* Your code for Filter here                                           *)
(*                                                                     *)
(* =================================================================== *)
fun Filter func [] = []
        |Filter func (x::xs) = if (func x) then (x :: Filter func xs)
                else (Filter func xs) ;

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

val testResults_filter = filter(fn x => x < 5) [1,2,3,4,5,6,7,8,9];

val testResults_5_3_3 = Filter (fn x => x < 5) [4, 6];

val testResults_filter_5_3_3 = filter (fn x => x < 5) [4, 6];
```

4.3.2 Execution Transcripts

```
> # # val Filter = fn: ('a -> bool) -> 'a list -> 'a list                    1
> > val testResults = [1, 2, 3, 4]: int list
> > val testResults_filter = [1, 2, 3, 4]: int list
> > val testResults_5_3_3 = [4]: int list
> > val testResults_filter_5_3_3 = [4]: int list
>
*** Emacs/HOL command completed ***
```

4.3.3 Explanations

From the recursive definition in 4.2, we can see that *Filter* can take in a function *func* and a list. If the list is an empty list *[]*, then the return value will be an empty list. Otherwise, the list can be interrepted as *x::xs*, then *func* will be applied on *x*, if the return value is *true*, then we return *x::Filter func xs*, if the return value is *false* we return *Filter func xs*.

Chapter 5

Exercise 5.3.5

5.1 Problem Statement

In this problem, we are requried to define a function *addPairsGreaterThan n l*, which takes in a integer *n* and a list of tuple *l*. If the input tuple list *l* is empty, then the return value of *addPairsGreaterThan n l* will be an empty list. Otherwise, the return value will be a list of the sums of those tuple whose values are all greater than *n*.

5.2 Relevant Code

```
fun addPairsGreaterThan n [] = []
  | addPairsGreaterThan n (x :: xs) =
    let
      fun GreaterThan n (x, y) = x > n andalso y>n
      fun sum (x, y) = x + y
    in
      if GreaterThan n x then (sum x :: addPairsGreaterThan n xs)
          else (addPairsGreaterThan n xs)
    end;
```

5.3 Test Cases, Execution Transcripts and Explanations

5.3.1 Test Cases

```
(***********************************************************************)
(* Exercise 5.3.5                                                      *)
(* Author: Shiu-Kai Chin                                               *)
(* Date: 20 September 2015                                             *)
(***********************************************************************)

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

fun addPairsGreaterThan n [] = []
  | addPairsGreaterThan n (x :: xs) =
    let
      fun GreaterThan n (x, y) = x > n andalso y>n
      fun sum (x, y) = x + y
    in
```

```
        if GreaterThan n x then (sum x :: addPairsGreaterThan n xs)
            else (addPairsGreaterThan n xs)
    end;

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

5.3.2 Execution Transcripts

```
> # # # # # # # # val addPairsGreaterThan = fn: int -> (int * int) list -> int list      1
>
*** Emacs/HOL command completed ***

> val it = [5, 9]: int list
val it = (): unit
>
*** Emacs/HOL command completed ***
```

5.3.3 Explanations

From the definition in 5.2 we could see, if the input list is an empty list, then the return value is also an empty list. In other case, we will use a recursive definiton on *addPairsGreaterThan*. We use *let* expression to define two functions *GreaterThan n* and *sum (x,y)*. Later, we will recursively check each tuple in the list and decide wheter we should insert its sum to the returned list.

Appendix A

Source Code for Exercise 4.6.3

The following code is from *ex-4-6-3.sml*

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

val funB1 = fn x => (fn y => (x < y) );
fun funB2 x y = (x<y);

val funC1 = fn x => (fn y => (x^y));
fun funC2 x y = x^y;

val funD1 = fn x => (fn y => (x@y));
fun funD2 x y = x@y;

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

(****************************************************************************)
(* Exercise 4.6.3                                                           *)
(* Author: Shiu-Kai Chin                                                    *)
(* Date: 18 September 2015                                                  *)
(****************************************************************************)

(****************************************************************************)
(* 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;

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 => (x < y) );
fun funB2 x y = (x<y);

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 x => (fn y => (x^y));
fun funC2 x y = x^y;

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 x => (fn y => (x@y));
fun funD2 x y = x@y;

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 Exercise 4.6.4

The following code is from *ex-4-6-4.sml*

```
(***************************************************************************)
(* Exercise 4.6.4                                                          *)
(* Author: Shiu-Kai Chin                                                   *)
(* Date: 20 September 2015                                                 *)
(***************************************************************************)

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

fun listSquares  [] = []
  | listSquares (x::xs) =
    let
      fun square x = x*x
    in
      (square x) :: listSquares xs
    end;

val testList = [1,2,3,4,5]

val testResults = listSquares testList
```

Appendix C

Source Code for Exercise 5.3.4

The following code is from *ex-5-3-4.sml*

```
(***********************************************************************)
(* Exercise 5.3.4                                                      *)
(* Author: Shiu-Kai Chin                                               *)
(* Date: 20 September 2015                                             *)
(***********************************************************************)

(* ================================================================== *)
(*                                                                     *)
(* Your code for Filter here                                           *)
(*                                                                     *)
(* ================================================================== *)
fun Filter func [] = []
        |Filter func (x::xs) = if (func x) then (x :: Filter func xs)
                 else (Filter func xs) ;

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

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

val testResults_5_3_3 = Filter (fn x => x < 5) [4, 6]

val testResults_filter_5_3_3 = filter (fn x => x < 5) [4, 6]
```

Appendix D

Source Code for Exercise 5.3.5

The following code is from *ex-5-3-5.sml*

```
(**********************************************************************)
(* Exercise 5.3.5                                                     *)
(* Author: Shiu-Kai Chin                                              *)
(* Date: 20 September 2015                                            *)
(**********************************************************************)

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

fun addPairsGreaterThan n [] = []
  | addPairsGreaterThan n (x :: xs) =
    let
      fun GreaterThan n (x, y) = x > n andalso y>n
      fun sum (x, y) = x + y
    in
      if GreaterThan n x then (sum x :: addPairsGreaterThan n xs)
      else (addPairsGreaterThan n xs)
    end;


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