Project 2

Alfred Murabito

March 22, 2020

Abstract

This report details results for the following exercise from *Certified Security by Design Using Higher Order Logic*: 4.6.3, 4.6.4, 5.3.4, 5.3.5, and 6.2.1. We define functions using pattern matching and got famaliar with HOL's logical symbols.

Acknowledgments: I received no assistance with this project.

Contents

Chapter 1

Executive Summary

All requirements for this project have been satisfied. A description and the results of each exercise are detailed in this project report.

Chapter 2

Exercise 4.6.3

2.1 Problem Statement

In this exercise I defined functions using the both the fun keyword and by using val assignmetns on lambda functions. Test results verifying the functions' intended behaviors follows.

Capabilities: ML functions defined to perform the following tasks:

- return the sum of elements in a 3-tuple
- determine whether one integer value is less than another
- concatenate two strings
- append lists
- determine whether one integer value is greater than another and return the larger integer value

2.2 Relevant Code

The following code snippets illustrate relevant sections of code developed to perform the above described tasks.

Part A:

```
val funa1 = fn (x, y, z) => x + y + z;
fun funa2 (x, y, z) = x + y + z;
```

Part B:

```
val funb1 = (fn x => (fn y => x < y));
fun funb2 x y = x < y;
```

Part C:

```
val func1 = fn s1 => (fn s2 => s1 ^ s2);
fun func2 s1 s2 = s1 ^ s2;
```

Part D:

```
val fund1 = fn list1 => (fn list2 => list1 @ list2);
fun fund2 list1 list2 = list1 @ list2;
```

Part E:

```
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 and Results

Test cases for Part A:

```
funa1 (1,3,4);              (* Expect 8 *)
funa2 (1,3,4);
funa1 (~1,0,1);             (* Expect 0 *)
funa2 (~1,0,1);
funa1 (489,2,34);           (* Expect 525 *)
funa2 (489,2,34);
```

```
> val it = 8: int
# val it = 8: int
> val it = 0: int
# val it = 0: int
> val it = 525: int
# val it = 525: int
>
```

Test cases for Part B:

```
funb1 1 2;                  (* Expect true *)
funb2 1 2;
funb1 ~1 0;                 (* Expect true *)
funb2 ~1 0;
funb1 ~3 ~44;               (* Expect false *)
funb2 ~3 ~44;
```

Results for Part B:

```
> val it = true: bool
# val it = true: bool
> val it = true: bool
# val it = true: bool
> val it = false: bool
# val it = false: bool
>
```

Test cases for Part C:

```
func1 "hello" " world";       (* Expect "hello world" *)
func2 "hello" " world";
func1 "" "asdf";              (* Expect "asdf" *)
func2 "" "asdf";
```

Results for Part C:

```
> val it = "hello world": string
# val it = "hello world": string
> val it = "asdf": string
# val it = "asdf": string
>
```

Test cases for Part D:

```
fund1 [1,2,3] [4,5];        (* Expect [1,2,3,4,5] *)
fund2 [1,2,3] [4,5];
fund1 [] [1,2,3];           (* Expect [1,2,3] *)
fund2 [] [1,2,3];
fund1 ["a","b","c"] [];          (* Expect ["a","b","c"] *)
fund2 ["a","b","c"] [];          (* Expect ["a","b","c"] *)
fund1 [true, false] [false, false, false]  (* Expect [true, false, false, false] *)
fund2 [true, false] [false, false, false]  (* Expect [true, false, false, false] *)
```

Results for Part D:

```
> val it = [1, 2, 3, 4, 5]: int list
# val it = [1, 2, 3, 4, 5]: int list
> val it = [1, 2, 3]: int list
# val it = [1, 2, 3]: int list
> val it = ["a", "b", "c"]: string list
# val it = ["a", "b", "c"]: string list
# val it = [true, false, false, false, false]: bool list
# val it = [true, false, false, false, false]: bool list
>
*** Emacs/HOL command completed ***

>
```

Test cases for Part E:

```
fune1 (34, 55);                  (* Expect 55 *)
fune2 (34, 55);
fune1 (~11, 478);                (* Expect 478 *)
fune2 (~11, 478);
fune1 (0, 0);                    (* Expect 0 *)
fune2 (0, 0);
```

Results for Part E:

```
> val it = 55: int
# val it = 55: int
> val it = 478: int
# val it = 478: int
> val it = 0: int
# val it = 0: int
> >
```

Chapter 3

Exercise 4.6.4

3.1 Problem Statement

For this exercise a function is defined that squares each element of a list, unless the list is empty. We use a "let" expression to define the function

3.2 Relevant Code

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

3.3 Test Cases and Results

Test cases for Exercise 4.6.4:

```
listSquares [];                (* Expect [] *)
listSquares [1,2,4,8];         (* Expect [1,4,16,64] *)
listSquares [5,12,7,~1];       (* Expect [25,144,49,1] *)
```

Results for Exercise 4.6.4:

```
> > > > # # # # # # val listSquares = fn: int list -> int list
> val it = []: int list
# val it = [1, 4, 16, 64]: int list
# val it = [25, 144, 49, 1]: int list
>
```

Chapter 4

Exercise 5.3.4

4.1 Problem Statement

The goal of this exercise is to define the function Filter that has identical behavior to the filter function. Filter accepts a function B and a list where every element "x" in the list where "B x" is true is in the list returned. Test cases matching the ML function and our function follows as well.

4.2 Relevant Code

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

4.3 Test Cases and Results

Test cases for Exercise 5.3.4:

```
Filter (fn x => x < 5) [];                    (* Expect [] *)
filter (fn x => x < 5) [];

Filter (fn x => x < 5) [4,6];                 (* Expect [4] *)
filter (fn x => x < 5) [4,6];

Filter (fn x => x < 5) [1,2,4,8,16];          (* Expect [1,2,4] *)
filter (fn x => x < 5) [1,2,4,8,16];

Filter (fn x => x < 5) [5,6,7,8];             (* Expect [] *)
filter (fn x => x < 5) [5,6,7,8];

Filter (fn x => x = "cat") ["dog","bird"]; (* Expect [] *)
filter (fn x => x = "cat") ["dog","bird"];

Filter (fn x => x = "cat") ["cat","dog","cat"];       (* Expect ["cat","cat"] *)
filter (fn x => x = "cat") ["cat","dog","cat"];
```

Results for Exercise 5.3.4:

```
> > > > # val Filter = fn: ('a -> bool) -> 'a list -> 'a list
> val it = []: int list
val it = []: int list
val it = [4]: int list
val it = [4]: int list
val it = [1, 2, 4]: int list
val it = [1, 2, 4]: int list
val it = []: int list
val it = []: int list
```

```
val it = []: string list
val it = []: string list
val it = ["cat", "cat"]: string list
val it = ["cat", "cat"]: string list
val it = (): unit
>
*** Emacs/HOL command completed ***

>
```

Chapter 5

Exercise 5.3.5

5.1 Problem Statement

This exercise involves defining a function which accepts a number "n" and a list of pairs of integers and returns a list with the sum of all pairs that were both greter than the given parameter "n".

5.2 Relevant Code

```
fun addPairsGreaterThan n list =
let
  fun addPair (x,y) = x + y
  val filteredList = filter (fn (x,y) => x > n andalso y > n) list
in
  map addPair filteredList
end;
```

5.3 Test Cases and Results

Test cases for Exercise 5.3.5:

```
(* Test Cases                        *)
addPairsGreaterThan 0 [(0,1),(2,0),(2,3),(4,5)];    (* expect [5,9] *)
addPairsGreaterThan 2 [(0,1),(2,0),(2,3),(4,5)];    (* expect [9] *)
addPairsGreaterThan 4 [(0,1),(2,0),(2,3),(4,5)];    (* expect [] *)
```

Results for Exercise 5.3.5:

```
> > > > # # # # # val addPairsGreaterThan = fn: int -> (int * int) list -> int list
> val it = [5, 9]: int list
# val it = [9]: int list
# val it = []: int list
```

Chapter 6

Exercise 6.2.1

6.1 Problem Statement

For this exercise HOL functionality is implemented and tested for basic HOL types and operators.

6.2 Relevant Code

```
(* 1: Enter the HOL equivalent of P(x) *)
``P x ==> Q y``;

(* 2: Constrain x to HOL type :num and y to :bool *)
``P (x:num) ==> Q (y:bool)``;

(* 3: Enter the HOL equivalent of forall x y.P(x) ==> Q(y) *)
``!x y.P(x) ==> Q(y)``;

(* 4: Enter the HOL equivalent of \exists (x:num).R(x:'a) *)
``?(x:num).R(x:num)``;

(* 5: *)
```~!x.P(x)\/Q(x) = ?x.~P(x)/\~Q(x)``;

(* 6: All people are mortal, where P(x) means x is a person, *)
(*   and M(x) means x is mortal. *)
``!x.P(x) ==> M(x)``;

(* 7: Some people are funny. *)
``?x.Funny(x)``;

(* 4b: Enter the HOL equivalent of \exists (x:num).R(x:'a) *)
``?(x:num).R(x:'a)``;
```

## 6.3 Test Cases and Results

For Exercise 6.2.1 the test cases are the HOL types and operators shown in the "Relevant Code" section above.

```

       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

>
```
```

```
> <<HOL message: inventing new type variable names: 'a, 'b>>
val it =
   ``(P :'a -> bool) (x :'a) ==> (Q :'b -> bool) (y :'b)``:
   term
val it =
   ``(P :num -> bool) (x :num) ==> (Q :bool -> bool) (y :bool)``:
   term
<<HOL message: inventing new type variable names: 'a, 'b>>
val it =
   ``!(x :'a) (y :'b). (P :'a -> bool) x ==> (Q :'b -> bool) y``:
   term
val it =
   ``?(x :num). (R :num -> bool) x``:
   term
<<HOL message: inventing new type variable names: 'a>>
val it =
   ``~!(x :'a).
     (P :'a -> bool) x \/ (Q :'a -> bool) x <=> ?(x :'a). ~P x /\ ~Q x``:
   term
<<HOL message: inventing new type variable names: 'a>>
val it =
   ``!(x :'a). (P :'a -> bool) x ==> (M :'a -> bool) x``:
   term
<<HOL message: inventing new type variable names: 'a>>
val it =
   ``?(x :'a). (Funny :'a -> bool) x``:
   term

Type inference failure: the term

(x :num)

on line 31, characters 13-17

can not be constrained to be of type

:'a

unification failure message: ???
Exception-
   HOL_ERR
     {message =
      "on line 31, characters 13-17:\n\nType inference failure: the term\n\n(x :num)\n\non line 31, cha
      origin_function = "type-analysis", origin_structure = "Preterm"} raised
>
*** Emacs/HOL command completed ***

>
```

Results Below are the observations for each test case

1. x is type 'a, y is type 'b', P is type ('a -¿ bool), Q is type ('b -¿ bool)
2. x is type "num" and y is type "bool" as expected

3. x and y are polymorphic types 'a, 'b, since none specified. P and Q must return bools since they are part of the implication statement
4. The HOL term fails since x is defined as type "num" then again as 'a. The correct statement with x as "num" is shown with the rest of the correct ones.
5. Entered as described and executed
6. M(x) represents subject is mortal, P(x) represents subject is a person
7. Funny(x) represents the subject is funny.

Appendix A

Exercise 4.6.3 Source Code

```
(*****************************************************************************)
(* Author: Alfred Murabito                                                   *)
(* Date: 25 January 2020                                                     *)
(* email: acmurabi@syr.edu                                                   *)
(*****************************************************************************)

(* Exercise 4.6.3 *)

(* function takes 3-tuple integer input and outputs the sum *)
val funa1 = fn (x, y, z) => x + y + z;
fun funa2 (x, y, z) = x + y + z;

funa1 (1,3,4);              (* Expect 8 *)
funa2 (1,3,4);
funa1 (~1,0,1);             (* Expect 0 *)
funa2 (~1,0,1);
funa1 (489,2,34);           (* Expect 525 *)
funa2 (489,2,34);

(* function takes two integer input and returns x < y *)
val funb1 = (fn x => (fn y => x < y));
fun funb2 x y = x < y;

funb1 1 2;                  (* Expect true *)
funb2 1 2;
funb1 ~1 0;                 (* Expect true *)
funb2 ~1 0;
funb1 ~3 ~44;               (* Expect false *)
funb2 ~3 ~44;


(* function takes two strings and concatenates them *)
val func1 = fn s1 => (fn s2 => s1 ^ s2);
fun func2 s1 s2 = s1 ^ s2;

func1 "hello" " world";      (* Expect "hello world" *)
func2 "hello" " world";
func1 "" "asdf";            (* Expect "asdf" *)
func2 "" "asdf";

(* Function that takes two lists list1 and list2 and appends them *)
val fund1 = fn list1 => (fn list2 => list1 @ list2);
fun fund2 list1 list2 = list1 @ list2;
```

```
fund1 [1,2,3] [4,5];       (* Expect [1,2,3,4,5] *)
fund2 [1,2,3] [4,5];
fund1 [] [1,2,3];          (* Expect [1,2,3] *)
fund2 [] [1,2,3];
fund1 ["a","b","c"] [];          (* Expect ["a","b","c"] *)
fund2 ["a","b","c"] [];          (* Expect ["a","b","c"] *)
fund1 [true, false] [false, false, false];  (* Expect [true, false, false, false] *)
fund2 [true, false] [false, false, false];  (* Expect [true, false, false, false] *)

(* Function that returns larger of a pair of values *)
val fune1 = fn (x,y) => if x > y then x else y;
fun fune2 (x,y) = if x > y then x else y;

fune1 (34, 55);                  (* Expect 55 *)
fune2 (34, 55);
fune1 (~11, 478);          (* Expect 478 *)
fune2 (~11, 478);
fune1 (0, 0);                    (* Expect 0 *)
fune2 (0, 0);
```

Appendix B

Exercise 4.6.4 Source Code

```
(*****************************************************************************)
(* Author: Alfred Murabito                                                   *)
(* Date: 26 January 2020                                                     *)
(* email: acmurabi@syr.edu                                                   *)
(*****************************************************************************)

(* Exercise 4.6.4 *)

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

(* Test cases as stated in the requirements *)
listSquares [];               (* Expect [] *)
listSquares [1,2,4,8];        (* Expect [1,4,16,64] *)
listSquares [5,12,7,~1];      (* Expect [25,144,49,1] *)
```

Appendix C

Exercise 5.3.4 Source Code

```
(*****************************************************************************)
(* Author: Alfred Murabito                                                   *)
(* Date: 25 January 2020                                                     *)
(* email: acmurabi@syr.edu                                                   *)
(*****************************************************************************)

(* Exercise 5.3.4 * Defining a function with behavior identical to filter)

fun Filter B [] = []
  | Filter B (x::xs) = if B x then x::(Filter B xs) else (Filter B xs);


Filter (fn x => x < 5) [];    (* Expect [] *)
filter (fn x => x < 5) [];

Filter (fn x => x < 5) [4,6];    (* Expect [4] *)
filter (fn x => x < 5) [4,6];

Filter (fn x => x < 5) [1,2,4,8,16];    (* Expect [1,2,4] *)
filter (fn x => x < 5) [1,2,4,8,16];

Filter (fn x => x < 5) [5,6,7,8];    (* Expect [] *)
filter (fn x => x < 5) [5,6,7,8];

Filter (fn x => x = "cat") ["dog","bird"]; (* Expect [] *)
filter (fn x => x = "cat") ["dog","bird"];

Filter (fn x => x = "cat") ["cat","dog","cat"];      (* Expect ["cat","cat"] *)
filter (fn x => x = "cat") ["cat","dog","cat"];
```

Appendix D

Exercise 5.3.5 Source Code

```
(*****************************************************************************)
(* Author: Alfred Murabito                                                   *)
(* Date: 25 January 2020                                                     *)
(* email: acmurabi@syr.edu                                                   *)
(*****************************************************************************)

(* Function addPairsGreaterThan n list, given integer n, given a list of
pairs of integers list, returns a list where each elemetn sum of pair
in list where both elements are greater than *)

fun addPairsGreaterThan n list =
let
  val list1 = filter (fn (a,b) => (a > n) andalso (b > n)) list
in
  map (fn (a,b) => a + b) list1
end;

(* Test case as specified in the requirements *)
addPairsGreaterThan 0 [(0,1),(2,0),(2,3),(4,5)];   (* expect [5,9] *)
addPairsGreaterThan 2 [(0,1),(2,0),(2,3),(4,5)];   (* expect [9] *)
addPairsGreaterThan 4 [(0,1),(2,0),(2,3),(4,5)];   (* expect [] *)
```

Appendix E

Exercise 6.2.1 Source Code

```
(*****************************************************************************)
(* Author: Alfred Murabito                                                   *)
(* Date: 25 January 2020                                                     *)
(* email: acmurabi@syr.edu                                                   *)
(*****************************************************************************)

(* Exercise 6.2.1 *)
(* 1: Enter the HOL equivalent of P(x) *)
``P x ==> Q y``;

(* 2: Constrain x to HOL type :num and y to :bool *)
``P (x:num) ==> Q (y:bool)``;

(* 3: Enter the HOL equivalent of forall x y.P(x) ==> Q(y) *)
``!x y.P(x) ==> Q(y)``;

(* 4: Enter the HOL equivalent of \exists (x:num).R(x:'a) *)
``?(x:num).R(x:num)``;

(* 5: *)
``~!x.P(x)\/Q(x) = ?x.~P(x)/\~Q(x)``;

(* 6: All people are mortal, where P(x) means x is a person, *)
(*   and M(x) means x is mortal. *)
``!x.P(x) ==> M(x)``;

(* 7: Some people are funny. *)
``?x.Funny(x)``;

(* 4b: Enter the HOL equivalent of \exists (x:num).R(x:'a) *)
``?(x:num).R(x:'a)``;
```