Project 2

Michael Hrishenko, LaTeX DAEMON

20 January 2020

Abstract

Basic knowledge and application of AUCTeX, emacs, ML, and LaTeX. Beginning to work on natural language translation into ML code, as well as different ways of declaring and building functions. Work presented as relevant in the Exercise Chapters, with full source code included in the Appendices.

Acknowledgements: Professors Marvine Hamner, Shiu-Kai Chin, & Susan Older

Contents

1 Executive Summary **4**

2 Exercise 4.6.3 **5**
2.1 Problem Statement 5
2.2 Relevant Code 5
2.3 Test Cases 6
2.4 Execution Transcripts 6

3 Exercise 4.6.4 **7**
3.1 Problem Statement 7
3.2 Relevant Code 7
3.3 Test Cases 7

4 Exercise 5.3.4 **8**
4.1 Problem Statement 8
4.2 Relevant Code 8
4.3 Test Cases 8
4.4 Execution Transcripts 8

5 Exercise 5.3.5 **11**
5.1 Problem Statement 11
5.2 Relevant Code 11
5.3 Test Cases 11
5.4 Execution Transcripts 11

6 Exercise 6.2.1 **12**
6.1 Problem Statement and Results 12

A Source Code for 4.6.3 **13**

B Source Code for 4.6.4 **14**

C Source Code for 5.3.4 **15**

D Source Code for 5.3.5 **16**

E Source Code for 6.2.1 **17**

Chapter 1

Executive Summary

Unable to complete parts of report. Partway solution to 5.3.4, problem 5.3.5 not attempted and 6.2.1 incomplete as well. Understand I am falling behind, not sure hot to correct but will try some changes in the coming week. Many thanks to Prof. Hamner, and my apologies for missing the posting of the Report Example the first time. This and all subsequent reports should be formatted completely to spec, please call me out if that is not the case. Have used `https://www.overleaf.com` to help mitigate lack of time for classwork. Due to my inability to install software on my work computer, and time spent at work, I do not believe I could complete the required amount of school material without a web-based solution such as this. I especially like the error reports upon compilation, and extensive in-house documentation provided by Overleaf. If a class subscription is available I would highly recommend it for future sessions.

Chapter 2

Exercise 4.6.3

2.1 Problem Statement

1. In ML, define functions (and test them on examples) corresponding to each function below. For each of the functions, you will define two ML functions,
   (a) the first using fn and val to define and name the function, and
   (b) the other using fun to define and name the function.
      i. 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.
      ii. 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.
      iii. A function that takes two strings s1 and s2 (where s1 is supplied first followed by s2) and concatenates them, where "ˆ" denotes string concatenation. For example, "Hi" ˆ " there" results in the string "Hi there".
      iv. A function that takes two lists list1 and list2 (where list1 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].
      v. 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.
2. Make sure you use pattern matching. For example, suppose the function is lx:(ly: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 As a naming convention, use the names funA1, funA2, funB1, funB2, etc.

2.2 Relevant Code

```
(*** A ***)
val funA1 = (fn x => (fn y => (fn z => x + y + z)));
fun funA2 x y z = x + y + z;
(*** B ***)
val funB1 = (fn y => (fn x => x < y));
fun funB2 x y = x < y;
(*** C ***)
val funC1 = (fn a => (fn b => a ^ b));
fun funC2 a b = a ^ b;
(*** D ***)
val funD1 = (fn x => (fn y => x @ y));
fun funD2 x y = x @ y;
(*** E ***)
val funE1 = (fn a => (fn b => if a > b then a else b));
fun funE2 a b = if a > b then a else b;
```

2.3 Test Cases

```
(*** A ***)
funA1 1 1 1;
funA2 1 1 1;
(*** B ***)
funB1 1 2;
funB2 1 2;
(*** C ***)
funC1 "hi" " you";
funC2 "hi" " you";
(*** D ***)
funD1 [1] [2];
funD2 [1] [2];
(*** E ***)
funE1 7 8;
funE2 7 8;
```

2.4 Execution Transcripts

```
# val it = 3: int
> val it = 3: int
> # val it = false: bool
> val it = true: bool
> # val it = "hi you": string
> val it = "hi you": string
> # val it = [1, 2]: int list
> val it = [1, 2]: int list
> # val it = 8: int
> val it = 8: int
```

Chapter 3

Exercise 4.6.4

3.1 Problem Statement

1. 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].

2. Define the function using a let expression in ML.

3.2 Relevant Code

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

3.3 Test Cases

```
listSquares [1,2];
listSquares [3,4];
```

Chapter 4

Exercise 5.3.4

4.1 Problem Statement

1. 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.

2. Your example should include the case $\lambda x.x < 5)[4, 6]$

4.2 Relevant Code

```
fun flt a b = map a b; (* THIS RETURNS THE BOOLS NEEDED *)
fun Filtr proc1 arr =
let
        val out = []
        fun retr proc2 (z::zs) = if proc2 z then z::out else out :: retr proc2 zs
    in
        retr proc1 arr
end;
```

4.3 Test Cases

```
Filtr (fn x => x < 5) [1,4,6];
```

4.4 Execution Transcripts

```
---------------------------------------------------------------------
       HOL-4 [Kananaskis 13 (stdknl, built Thu Jan 16 23:56:17 2020)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
---------------------------------------------------------------------
> fun flt a b = map a b;
val flt = fn: ('a -> 'b) -> 'a list -> 'b list
> flt (fn x => x < 5) [1,4,6];
val it = [true, true, false]: bool list
> filter (fn x => x < 5) [1,4,6];
val it = [1, 4]: int list
> fun Filtr proc arr =
let
        fun t# # ru proc arr = map proc arr;
    #     fun retr (x::xs) (z::zs) = if x then z :: retr xs zs
    in
      # poly: : error:   retr (else expectedtru pro but inc arr)  was arr
end;found
poly: : error: Expression expected but in was found
# #
Static Errors
> fun Filtr proc arr =
le# t
        fun t# ru proc arr = map proc arr;
        val#  out = []
        fun re# tr (x::xs) (z::zs) = if x then z @ out else out :: retr xs zs
    in
# #        retr (tru proc arr) arr
end;
# poly: : warning: Matches are not exhaustive.
Found near
  fun retr (x :: xs) (... :: ...) = if x then z @ out else out :: ... ...
val Filtr = fn: (’a list list -> bool) -> ’a list list list -> ’a list list
> Filtr (fn x => x < 5) [1,4,6];
poly: : error: Type error in function application.
   Function: Filtr :
      (’a list list -> bool) -> ’a list list list -> ’a list list
   Argument: (fn x => x < 5) : int -> bool
   Reason:
      Can’t unify int (*In Basis*) with ’a list list (*In Basis*)
         (Different type constructors)
Found near Filtr (fn x => x < 5) [1, 4, 6]
poly: : error: Type error in function application.
   Function: Filtr (fn x => x < 5) : ’a list list list -> ’a list list
   Argument: [1, 4, 6] : int list
   Reason:
      Can’t unify int (*In Basis*) with ’a list list (*In Basis*)
         (Different type constructors)
Found near Filtr (fn x => x < 5) [1, 4, 6]
Static Errors
> Filtr [1,4,6] (fn x => x < 5);
poly: : error: Type error in function application.
   Function: Filtr :
      (’a list list -> bool) -> ’a list list list -> ’a list list
   Argument: [1, 4, 6] : int list
   Reason: Can’t unify int list to ’a list list -> bool (Incompatible types)
Found near Filtr [1, 4, 6] (fn x => x < 5)
poly: : error: Type error in function application.
   Function: Filtr [1, 4, 6] : ’a list list list -> ’a list list
   Argument: (fn x => x < 5) : int -> bool
   Reason: Can’t unify ’a list list list to int -> bool (Incompatible types)
Found near Filtr [1, 4, 6] (fn x => x < 5)
Static Errors
> fun Filtr proc arr =
le# t
        fun t# ru proc arr = map proc arr;
        val#  out = []
      #   fun retr (x::xs) (z::zs) = if x then [z] @ out else out :: retr xs zs
    in
      # #   retr (tru proc arr) arr
end;#
poly: : warning: Matches are not exhaustive.
Found near
  fun retr (x :: xs) (... :: ...) = if x then [...] @ out else out :: ... ...
val Filtr = fn: (’a list -> bool) -> ’a list list -> ’a list list
> Filtr (fn x => x < 5) [1,4,6];
poly: : error: Type error in function application.
   Function: Filtr : (’a list -> bool) -> ’a list list -> ’a list list
   Argument: (fn x => x < 5) : int -> bool
   Reason:
      Can’t unify int (*In Basis*) with ’a list (*In Basis*)
         (Different type constructors)
Found near Filtr (fn x => x < 5) [1, 4, 6]
poly: : error: Type error in function application.
   Function: Filtr (fn x => x < 5) : ’a list list -> ’a list list
   Argument: [1, 4, 6] : int list
   Reason:
      Can’t unify int (*In Basis*) with ’a list (*In Basis*)
         (Different type constructors)
Found near Filtr (fn x => x < 5) [1, 4, 6]
Static Errors
> 1::[2,4];
val it = [1, 2, 4]: int list
> fun Filtr proc arr =
le# t
        fun t# ru proc arr = map proc arr;
        val#  out = []
        fun r# etr (x::xs) (z::zs) = if x then z::out else out :: retr xs zs
    in
       # #  retr (tru proc arr) arr
end;#
poly: : warning: Matches are not exhaustive.
Found near
  fun retr (x :: xs) (... :: ...) = if x then z :: out else out :: ... ...
val Filtr = fn: (’a list -> bool) -> ’a list list -> ’a list list
```

```
> Filtr (fn x => x < 5) [1,4,6];
poly: : error: Type error in function application.
   Function: Filtr : ('a list -> bool) -> 'a list list -> 'a list list
   Argument: (fn x => x < 5) : int -> bool
   Reason:
      Can't unify int (*In Basis*) with 'a list (*In Basis*)
         (Different type constructors)
Found near Filtr (fn x => x < 5) [1, 4, 6]
poly: : error: Type error in function application.
   Function: Filtr (fn x => x < 5) : 'a list list -> 'a list list
   Argument: [1, 4, 6] : int list
   Reason:
      Can't unify int (*In Basis*) with 'a list (*In Basis*)
         (Different type constructors)
Found near Filtr (fn x => x < 5) [1, 4, 6]
Static Errors
> ^[[Cfun Filtr proc1 arr =
lpoly: : error: et
        val unknown characterout = [] "\^["

        fun re# tr proc2#  (z::# zs) = if proc2 z then z::out else out :: retr proc2 zs
    in
     #    retr # proc1 arr
end;#
poly: : error: ] expected but ; was found
Static Errors
> fun Filtr proc1 arr =
let
        val # out = []#
        fun re# tr proc2 (z::zs) = if proc2 z then z::out else out :: retr proc2 zs
    in
     #    retr # proc1 arr
end;#
poly: : warning: Matches are not exhaustive.
Found near
  fun retr proc2 (... :: ...) = if proc2 z then z :: out else out :: ... ...
val Filtr = fn: ('a list -> bool) -> 'a list list -> 'a list list
> Filtr (fn x => x < 5) [1,4,6];
poly: : error: Type error in function application.
   Function: Filtr : ('a list -> bool) -> 'a list list -> 'a list list
   Argument: (fn x => x < 5) : int -> bool
   Reason:
      Can't unify int (*In Basis*) with 'a list (*In Basis*)
         (Different type constructors)
Found near Filtr (fn x => x < 5) [1, 4, 6]
poly: : error: Type error in function application.
   Function: Filtr (fn x => x < 5) : 'a list list -> 'a list list
   Argument: [1, 4, 6] : int list
   Reason:
      Can't unify int (*In Basis*) with 'a list (*In Basis*)
         (Different type constructors)
Found near Filtr (fn x => x < 5) [1, 4, 6]
Static Errors
> (fn x => x < 5) 1;
val it = true: bool
> %
```

Chapter 5

Exercise 5.3.5

5.1 Problem Statement

1. Define an ML function addPairsGreaterThan n list, whose behavior is defined as follows:
   (a) given an integer n, and
   (b) given a list of pairs of integers list,
   (c) 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.
2. An example session is shown below.

```
addPairsGreaterThan 0 [(0,1),(2,0),(2,3),(4,5)];
> val it = [5, 9] : int list
```

   Hint: the logical operators for negation, conjunction, and disjunction ML are not, andalso, and orelse.

5.2 Relevant Code

5.3 Test Cases

5.4 Execution Transcripts

Chapter 6

Exercise 6.2.1

6.1 Problem Statement and Results

1. In the following problems, enable HOL's Show types capability and disable Unicode so only ACSII characters are displayed.

   (a) Enter the HOL equivalent of P(x) Q(y). Show what HOL returns. What are the types of x, y, P, and Q?

   ```
   > ‘‘P x ==> Q y‘‘;
   <<HOL message: inventing new type variable names: ’a, ’b>>
   val it = \P x ==> Q y": term
   ```

   (b) Consider again P(x) Q(y). Suppose we wish to constrain x to HOL type :num and y to HOL type :bool. Re-enter your expression corresponding to P(x) Q(y) and show that the types of x, y, P, and Q are appropriately typed.

   ```
          > ‘‘P x:num ==> Q y:bool‘‘;
   Type inference failure: unable to infer a type for the application of
   $==> :bool -> bool -> bool
   roughly on line 6, characters 2-5
   to
   (P :x -> num) (x :x)
   on line 6, characters 2-4
   which has type
   :num
   unification failure message: Attempt to unify different type operators: min$bool and num$num
   ```

   (c) Enter the HOL equivalent of 8x y:P(x) Q(y), without explicitly specifying types. What do you get and why?

   (d) Enter the HOL equivalent of 9(x : num):R(x : a). What happens and why?

   (e) Enter the HOL equivalent of :8x:P(x)_Q(x) = 9x::P(x)ˆ:Q(x)

   (f) Enter the HOL equivalent of the English sentence, All people are mortal, where P(x) represents x is a person and M(x) represents x is mortal.

   (g) Enter the HOL equivalent of the English sentence, Some people are funny, where Funny(x) denotes x is funny.

Appendix A

Source Code for 4.6.3

```
(********************************************************)
(* Exercise 4.6.3 *)
(* Author: Michael Hrishenko *)
(* Date: 20JAN2020 *)
(********************************************************)
(fn x => (fn y => 2*x + y))1 2; (* EXAMPLE *)
(*** A ***)
val funA1 = (fn x => (fn y => (fn z => x + y + z)));
fun funA2 x y z = x + y + z;
(*** B ***)
val funB1 = (fn y => (fn x => x < y));
fun funB2 x y = x < y;
(*** C ***)
val funC1 = (fn a => (fn b => a ^ b));
fun funC2 a b = a ^ b;
(*** D ***)
val funD1 = (fn x => (fn y => x @ y));
fun funD2 x y = x @ y;
(*** E ***)
val funE1 = (fn a => (fn b => if a > b then a else b));
fun funE2 a b = if a > b then a else b;
```

Appendix B

Source Code for 4.6.4

```
(********************************************************)
(* Exercise 4.6.4 *)
(* Author: Michael Hrishenko *)
(* Date: 20JAN2020 *)
(********************************************************)
fun listSquares arr =
let
        fun sqr x = x * x
        fun arq [] = [] | arq (x::xs) = sqr x :: arq xs
    in
        arq arr
end;

fun sqr [] = [] | sqr (x::xs) = (x * x) :: sqr xs;
```

Appendix C

Source Code for 5.3.4

```
(*******************************************************)
(* Exercise 5.3.4 *)
(* Author: Michael Hrishenko *)
(* Date: 20JAN2020 *)
(*******************************************************)
fun flt a b = map a b; (* THIS RETURNS THE BOOLS NEEDED *)
fun Filtr proc1 arr =
let
        val out = []
        fun retr proc2 (z::zs) = if proc2 z then z::out else out :: retr proc2 zs
    in
        retr proc1 arr
end;
```

Appendix D

Source Code for 5.3.5

Appendix E

Source Code for 6.2.1