Report for Project4

Xiaozhi Li

September 27, 2017

Abstract

Project 4 using LaTeX. This project demonstrate some equations using ML.

- Problem statement
- Relevant code
- Test results

For each problem there will be a source code included in the Appendix. They are generated directly using the original source file, hence the changes in the file are represent on the report.

The style files we use:

- a style file for the course, *634format.sty*,
- the *listings* package for displaying and inputting ML source code, and
- HOL style files and commands to display interactive ML/HOL sessions.

Acknowledgments: This report follows the hand book Certified Security by Design Using Higer Order Logic, and course instructions from CIS400-CSBD.

Contents

Chapter 1

Executive Summary

All requirements for this project are satisfied. Specifically,

Report Contents
Our report has the following content:

Chapter : Executive Summary

Chapter 2: Exercise 6.2.1
- Section 2.1: Problem Statement
- Section 2.2: Relevent Code
- Section 2.3: Execution Transcripts
  - Sub-section 2.3.1 : Explain of Results

Chapter 3 : Exercise 7.3.1
- Section 3.1: Problem Statement
- Section 3.2: Relevent Code
- Section 3.3: Test Cases
- Section 3.4: Execution Transcripts

Chapter 4: Exercise 7.3.2
- Section 4.1: Problem Statement
- Section 4.2: Relevent Code
- Section 4.3: Execution Transcripts

Chapter 5: Exercise 7.3.3
- Section 5.1: Problem Statement
- Section 5.2: Relevent Code
- Section 5.3: Execution Transcripts

Chapter 6: Appendix A: Source Code for 6.2.1

Chapter 7: Appendix B: Source Code for 7.3.1

Chapter 8: Appendix C: Source Code for 7.3.2

Chapter 9: Appendix D: Source Code for 7.3.3

Reproducibility in ML and LaTeX
Our LaTeX source files compiles with no errors. During the tests for ML, there is one test from 6.2.1 that could not compile. However this was intended to show how HOL handles type error under constrained type. All other source code compiled with no problem.

Chapter 2

Exercise 6.2.1

2.1 Problem Statement

For exercise 6.2.1 we are using HOL to demonstrate following logic problems:

$$
\begin{aligned}
1, 2. &\quad P(x) \supset Q(x) \\
3. &\quad \forall x\ y.P(x) \supset Q(y) \\
4. &\quad \exists(x : num).R(x : \alpha) \\
5. &\quad \neg\forall x.P(x) \vee Q(x) = \exists x.\neg P(x) \wedge \neg Q(x)
\end{aligned}
$$

There are also exercise that uses logic formulas to demonstrate English sentence

6. All people are mortal, where P(x) represents x is a person **and** M(x) represents x is mortal.

7. Some people are funny, where Funny(x) denotes x is funny.

2.2 Relevent Code

These are the code for chapter 6.2.1:

```
(**1.**)
``P x ==> Q y``;

(**2.**)
``(P:num -> bool) (x:num)  ==> (Q:bool->bool) (y:bool)``;

(**3.**)
``!x y.(P x) ==> (Q y)``;

(**4.**)
``?(x :num).(R (x :'a))``;

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

(**6.**)
``!x.(P x) ==> (M x)``;

(**7.**)
``?x.(P x) ==> (Funny x)``;
```

2.3 Execution Transcripts

We sent code 1-7 to HOL, here are the results excluding NO.4:

1

```
> # <<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
<<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). (P :'a -> bool) x ==> (Funny :'a -> bool) x``:
   term
val it = (): unit
> HOL message: inventing new type variable names: 'a, 'b>>
val it =
   P x  Q y:
   term
> > # # # # val it =
   P x  Q y:
   term
> > # # # <<HOL message: inventing new type variable names: 'a, 'b>>
val it =
   x y. P x  Q y:
   term
> > # # # <<HOL message: inventing new type variable names: 'a>>
val it =
   (x. P x  Q x)  x. P x  Q x:
   term
> > # <<HOL message: inventing new type variable names: 'a>>
val it =
   x. P x  M x:
   term
> # <<HOL message: inventing new type variable names: 'a>>
val it =
   x. P x  Funny x:
   term
> >
*** Emacs/HOL command completed ***
```

2.3.1 Explain of results

The result from 1 to 3 and 5 to 7 are all expected, which means our theory was corrected. However, for problem 4 we got error:

1

```
> # <<HOL message: inventing new type variable names: 'a>>

Type inference failure: unable to infer a type for the application of

(x :num)

at line 33, character 16

to

(:` :)
```

This is caused by type error, in the equation we already set x as a :num type, and try to ask HOL to give result as ‘a type, this is seen as impossible by HOL, hence the error message. This error was expected, and all other tests are passed.

Chapter 3

Exercise 7.3.1

3.1 Problem Statement

For exercise 7.3.1, we are asked to create a function andImp2Imp term, which will opperate to take:

$$p \wedge q \subset r$$

and transform it into:

$$p \subset q \subset r;$$

3.2 Relevent Code

Here is the code for 7.3.1:

```
(* Exercise 7.3.1*********)
fun andImp2Imp term=
let
 val (conjTerm1,r)= dest_imp term
 val (p,q )=dest_conj conjTerm1
 (*val ts= mk_imp (p,q) *)
in
``p ==> q ==> r``
end;
```

3.3 Test Cases

The following code are for testing our results:

```
andImp2Imp ``(p/\q) ==> r``;
```

3.4 Execution Transcripts

We sent the above code to HOL, and here is the output from HOL:

```
> > > ># # # # # # # val andImp2Imp = fn: term -> term
> val it =
   p  q  r:
   term
>
```

1

It appears that our test passed HOL.

Chapter 4

Exercise 7.3.2

4.1 Problem Statement

For exercise 7.3.2, we are asked to create a function andImp2Imp term, which will opperate to take:

$$p \subset q \subset r;$$

and transform it into:

$$p \wedge q \subset r$$

Notice this is just the reverse of what we were doing in 7.3.1.

4.2 Relevent Code

For testing purpose, we included code from 7.3.1 and 7.3.2 into same file.

```
(* Exercise 7.3.2*********)
fun impImpAnd impTerm=
let
 val (p,qr)= dest_imp impTerm
 val (q,r )=dest_imp qr
in
 mk_imp (mk_conj (p,q), r)
end;

(* Exercise 7.3.1*********)
fun andImp2Imp term=
let
 val (conjTerm1,r)= dest_imp term
 val (p,q )=dest_conj conjTerm1
 (*val ts= mk_imp (p,q) *)
in
``p ==> q ==> r``
end;

(* test case for 7.3.2:  *)
impImpAnd ``(p ==> q ==> r)``;
impImpAnd (andImp2Imp `` (p /\ q) ==> r``);
andImp2Imp (impImpAnd ``p ==>q ==>r``);
```

4.3 Execution Transcripts

Here is a transcript of 7.3.2:

1

```
> > > > # # # # # # # # val andImp2Imp = fn: term -> term
> > # # # # # # # # val impImpAnd = fn: term -> term
> > # val it =
   ``(p :bool) /\ (q :bool) ==> (r :bool)``:
   term
> val it =
   ``(p :bool) /\ (q :bool) ==> (r :bool)``:
   term
> val it =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
>
*** Emacs/HOL command completed ***

>
Process HOL killed
```

It appears that all our tests has passed for 7.3.2.

Chapter 5

Exercise 7.3.3

5.1 Problem Statement

In 7.3.3, we are asked to create a function *notExists term* ,such that this function will oiperate on the form $\neg\exists x.P(x)$ and returns $\forall x.\neg P(x)$.

5.2 Relevent Code

Here is the code for 7.3.3:

```
(* exercise 7.3.3        *)

fun forallTerm longTerm=
let
val (x,P)=dest_exists (dest_neg a)
in
``!x.~(P x)``
end;
```

The following code are for testing our results:

```
forallTerm ``~?x.(P x)``;
forallTerm ``~?z.Q z``;
```

5.3 Execution Transcripts

We sent above code to HOL, and here is the output from HOL:

1

```
> # # # # # val forallTerm = fn: 'a -> term
> <<HOL message: inventing new type variable names: 'a>>
<<HOL message: inventing new type variable names: 'a>>
val it =
   ``!(x :'a). ~(P :'a -> bool) x``:
   term
> # <<HOL message: inventing new type variable names: 'a>>
<<HOL message: inventing new type variable names: 'a>>
val it =
   ``!(x :'a). ~(P :'a -> bool) x``:
   term
> >
```

All our tests have been passed.

Chapter 6

Appendix A: source code for 6.2.1

The following code is from *ex6-2-1.sml*

```
(*Exercise 6.2.1****For HOL*)
(*Written By XIaozhi Li ****)
(**17/9/27**)

(**1.**)
(****P(x) sup set Q(y)              *)

``P x ==> Q y``;

(**2.**)
(****P(x) sup set Q(y) with x constrain to HOL type :num **)
(**  and y to Hol type :bool*********************************)

``(P:num -> bool) (x:num)  ==> (Q: bool->bool) (y:bool)``;

(****************)

(**3.**)
``!x y.(P x) ==> (Q y)``;

(* question 4 is at the bottom because it has type error*)

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

(**6.**)
``!x.(P x) ==> (M x)``;
(**7.**)
``?x.(P x) ==> (Funny x)``;


(**4.**)
``?(x :num).(R (x :'a))``;

(* this one cannot evaluate, because x was specified as num, then specify to be alpha, hen
```

Chapter 7

Appendix B: source code for 7.3.1

The following code is from *731backup.sml*

```
(* created by Xiaozhi Li *)
(* 17/9/27                *)
(* Exercise 7.3.1*********)
fun andImp2Imp term=
let
 val (conjTerm1,r)= dest_imp term
 val (p,q )=dest_conj conjTerm1
 (*val ts= mk_imp (p,q) *)
in
``p ==> q ==> r``
end;

andImp2Imp ``(p/\q) ==> r``;
```

Chapter 8

Appendix C: source code for 7.3.2

The following code is from *ex-7-3-2.sml*

```
(* created by Xiaozhi Li *)
(* 17/9/27                *)
(* Exercise 7.3.1*********)
fun andImp2Imp term=
let
 val (conjTerm1,r)= dest_imp term
 val (p,q )=dest_conj conjTerm1
 (*val ts= mk_imp (p,q) *)
in
``p ==> q ==> r``
end;

(* Exercise 7.3.2*********)
fun impImpAnd impTerm=
let
 val (p,qr)= dest_imp impTerm
 val (q,r )=dest_imp qr
in
 mk_imp (mk_conj (p,q), r)
end;

(* test case for 7.3.2:  *)
impImpAnd ``(p ==> q ==> r)``;
impImpAnd (andImp2Imp `` (p /\ q) ==> r``);
andImp2Imp (impImpAnd ``p ==>q ==>r``);
```

Chapter 9

Appendix D: source code for 7.3.3

The following code is from *ex-7-3-3.sml*

```
(* created by Xiaozhi Li*)
(* exercise 7.3.3        *)

fun forallTerm longTerm=
let
val (x,P)=dest_exists (dest_neg a)
in
``!x.~(P x)``
end;

(*     these are for easy testing.        *)
forallTerm ``~?x.(P x)``;

forallTerm ``~?z.Q z``;
```