Project 3 Report

Kyle Peppe

January 23, 2020

Abstract

This project began my work in the HOL environment and using HOL terms. For these exercises I was creating a function that would start with a certain form and then returns a different form.

- Problem statement
- Relevant code
- Test results
- Execution Transcripts

For each problem or exercise-oriented chapter in the main body of the report is a corresponding chapter in the Appendix containing the source code in ML. This source code is not pasted into the Appendix. Rather, it is input directly from the source code file itself.

Acknowledgments: I would like to acknowledge the 2 professors, Professor Chin and Professor Hamner, that have helped me begin to understand this new ML programming language. Also to Syracuse University for accepting me to this Masters program in Cybersecurity.

Contents

Chapter 1

Executive Summary

All requirements for this project are satisfied. Specifically,

Report Contents
The report has the following content:

Reproducibility in ML and LaTeX
Our ML and LaTeX source files compile with no errors.

Chapter 2

Exercise 7.3.1

2.1 Problem Statement for Exercise 7.3.1

For this problem I solved the Exercise 7.3.1 from the PDF book. We were given a starting function and had to use the terms provided and get that term to return a desired term.

2.2 Relevant Code for Exercise 7.3.1

The relevant code wil be in the corresponding Appendix at the end of the report.

2.3 Test cases for Exercise 7.3.1

There were no test cases here were giving a starting function and then had to use logic to get to a desired funtion. The starting function was:

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

2.4 Execution Transcripts for Exercise 7.3.1

Below are the results from running the test cases:
The following is output from *chapter7Answer.sml*

```
---------------------------------------------------------------------
       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
---------------------------------------------------------------------
> > > > # # # # # # # # # ** types trace now on
> # # # # # # # # # ** Unicode trace now off
> val andImpTerm = ``p /\ q ==> r``;
val andImpTerm =
   ``(p :bool) /\ (q :bool) ==> (r :bool)``:
   term
> fun andImp2Imp term = ``p ==> q ==> r``;
val andImp2Imp = fn: 'a -> term
> andImp2Imp andImpTerm;
val it =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
>
Process HOL killed
```

Chapter 3

Exercise 7.3.2

3.1 Problem Statement for Exercise 7.3.2

For this problem I solved the Exercise 7.3.2 from the PDF book. In this exercise we were given a term and instructed to get this to return a new term similiar to 7.3.1. But there was the added step of once getting our original term impImpAnd to return andImp2Imp then I needed reverse andImp2Imp back to impImpAnd.

3.2 Relevant Code for Exercise 7.3.2

The relevant code wil be in the corresponding Appendix at the end of the report.

3.3 Test cases for Exercise 7.3.2

Once again we weren't necessarily given test cases but a started value and an ending value to get the term to (and then reverse back):

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

3.4 Execution Transcripts for Exercise 7.3.2

Below are the results from running the test cases:
The following is output from *chapter7Answer.sml*

```
---------------------------------------------------------------------
        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
---------------------------------------------------------------------
> > > > # # # # # # # # # # ** types trace now on
> # # # # # # # # # # ** Unicode trace now off
> fun andImp2Imp term = ``p ==> q ==> r``;
val andImp2Imp = fn: 'a -> term
> fun impImpAnd term = ``p /\ q ==> r``;
val impImpAnd = fn: 'a -> term
> val v1 = andImp2Imp ``p /\ q ==> r``;
val v1 =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
> val v2 = impImpAnd v1;
```

```
val v2 =
   ``(p :bool) /\ (q :bool) ==> (r :bool)``:
   term
> val v3 = andImp2Imp v2;
val v3 =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
>
Process HOL killed
```

Chapter 4

Exercise 7.3.3

4.1 Problem Statement for Exercise 7.3.3

For this problem I solved the Exercise 7.3.3 from the PDF book. Here we were given a term and had to get the function to return a different form. This is similiar to the previous 2 exercises, just more difficult.

4.2 Relevant Code for Exercise 7.3.3

The relevant code wil be in the corresponding Appendix at the end of the report.

4.3 Test cases for Exercise 7.3.3

Once again we weren't necessarily given test cases but a started value and an ending value to get the term to (and then reverse back):

```
notExists = ~?z.Q(z)
```

4.4 Execution Transcripts for Exercise 7.3.3

Below are the results from running the test cases:

The following is output from *chapter7Answer.sml*

```
         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

> > > > # # # # # # # # # # ** types trace now on
> # # # # # # # # # # ** Unicode trace now off
> fun notExists term = ``!x.~P(x)``;
val notExists = fn: 'a -> term
> notExists ``~?z.Q(z)``;
<<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
>
Process HOL killed
```

Chapter 5

Relevant Information

5.1 Specific Test Cases

5.1.1 Test Cases for Exercise 7.3.1

There were no test cases here were giving a starting function and then had to use logic to get to a desired funtion. The starting function was:

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

5.1.2 Test Cases for Exercise 7.3.2

Once again we weren't necessarily given test cases but a started value and an ending value to get the term to (and then reverse back):

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

5.1.3 Test Cases for Exercise 7.3.3

Once again we weren't necessarily given test cases but a started value and an ending value to get the term to (and then reverse back):

```
notExists = ~?z.Q(z)
```

Appendix A

Source Code for Exercise 7.3.1, 7.3.2, and 7.3.3

The following code is from *chapter7Answer.sml*

```
(**************************************************************************)
(* Author: Kyle Peppe                                                     *)
(* Exercise 7.3.1, 7.3.2, and 7.3.3                                       *)
(* Date: 1/21/20                                                          *)
(**************************************************************************)

(* Exercise 7.3.1                         *)
(* Val first                              *)
val andImpTerm = ``p /\ q ==> r``;

(* Now creating Function                  *)
fun andImp2Imp term = ``p ==> q ==> r``;

(* Combining the 2                        *)
andImp2Imp andImpTerm;

(* Exercise 7.3.2                         *)
fun andImp2Imp term = ``p ==> q ==> r``;
fun impImpAnd term = ``p /\ q ==> r``;

val v1 = andImp2Imp ``p /\ q ==> r``;
val v2 = impImpAnd v1;
val v3 = andImp2Imp v2;

(* Exercise 7.3.3                         *)
fun notExists term = ``!x.~P(x)``;
notExists ``~?z.Q(z)``;
```