# Homework 4

**Chirag Sachdev**

Week 4

**Abstract**

This project is a part of HW4 of Assurance Foundations. The homework deals with integration of ML and HOL to LaTeX. The goal of this report is to show reproducibility which is the groundwork for credibility that I have done this on my own without any external help. Every Chapter demonstrates the following sections:

- Problem Statement
- Relevant Code
- Test Results

This project includes the following packages:

***634format.sty*** A format style for this course

***listings*** Package for displaying and inputting ML source code

***holtex*** HOL style files and commands to display in the report

This document also demonstrates my ability to :

- Easily generate a table of contents,
- Refer to chapter and section labels

My skills and my professional details can be found at `https://www.linkedin.in/in/chiragsachdev`.

## Acknowledgments

I would gratefully acknowledge Dr. Shiu-Kai Chin and my other professors at Syracuse University and my Professors at Drexel University for being the wonderful mentors they are to guide me through my journey of obtaining a Master's Dregree. I used the following table to indert symbols to the LaTeX document: `https://en.wikipedia.org/wiki/List_of_mathematical_symbols`

# Contents

Chapter 1

# Executive Summary

**All requirements have been met** The reason I have submitted this assignment late is because I have not been keeping well. This document illustrates the abilities of work with terms in *HOL*.

**Chapter 2**

# Excercise 6.2.1

## 2.1 Problem statement

In the following problems, enable HOLs Show types capability and disable Unicode so only ACSII characters are displayed.

1. Enter the HOL equivalent of $P(x) \supset Q(y)$. Show what HOL returns. What are the types of *x, y, P,* and $Q$?
2. Consider again $P(x) \supset Q(y)$. Suppose we wish to constrain $x$ to HOL type *:num* and $y$ to HOL type *:bool*. Re-enter your expression corresponding to $P(x) \supset Q(y)$ and show that the types of *x, y, P,* and $Q$ are appropriately typed.
3. Enter the HOL equivalent of $\forall xy.P(x) \supset Q(y)$, without explicitly specifying types. What do you get and why?
4. Enter the HOL equivalent of $\exists(x : num).R(x : \alpha)$. What happens and why?
5. Enter the HOL equivalent of $\neg\forall x.P(x) \vee Q(x) = \exists x.\neg P(x) \wedge \neg Q(x)$
6. 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.
7. Enter the HOL equivalent of the English sentence, Some people are funny, where $Funny(x)$ denotes $x$ is funny.

## 2.2 Relevant Code

```
(*1*)
``P x ==> Q y``;
(*2*)
``P (x:num) ==> Q (y:bool)``;
(*3*)
``!x. !y. P x ==> Q y``;
(*4*)
``?x. R (x:num)``;
(*5*)
```~!x.(P x \/ Q x) = ?x.(~P x /\ ~Q x)``
(*6*)
(* All people are mortal*)
``!x. P x ==> M x``
(*7*)
(*Some people are funny*)
``!x. P x ==> ?x.Funny x``
```
```

## 2.3 Test Case

*1*

```
------------------------------------------------------------------------
       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
> <<HOL message: inventing new type variable names: 'a>>
val it = ``(x :'a)``: term
> <<HOL message: inventing new type variable names: 'a>>
val it = ``(y :'a)``: term
> <<HOL message: inventing new type variable names: 'a, 'b>>
val it =
   ``(P :'a -> 'b) (x :'a)``:
   term
> <<HOL message: inventing new type variable names: 'a>>
val it = ``(y :'a)``: term
> <<HOL message: inventing new type variable names: 'a, 'b>>
val it =
   ``(Q :'a -> 'b) (y :'a)``:
   term
> <<HOL message: inventing new type variable names: 'a, 'b>>
val it =
   ``(P :'a -> bool) (x :'a) ==> (Q :'b -> bool) (y :'b)``:
   term>
Process HOL finished


------------------------------------------------------------------------
       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
------------------------------------------------------------------------
> > > ># # # # # # # # # ** Unicode trace now off
> # # # # # # # # # ** types trace now on
> val it =
   ``(P :num -> bool) (x :num) ==> (Q :bool -> bool) (y :bool)``:
   term
>
Process HOL finished


------------------------------------------------------------------------
       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
> <<HOL message: inventing new type variable names: 'a, 'b>>
val it =
   ``!(x :'a) (y :'b). (P :'a -> bool) x ==> (Q :'b -> bool) y``:
   term
>
Process HOL finished


------------------------------------------------------------------------
       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 it = ``(x :num)``: term
> <<HOL message: inventing new type variable names: 'a, 'b>>
val it =
   ``(R :'a -> 'b) (x :'a)``:
   term
> <<HOL message: inventing new type variable names: 'a>>
val it =
   ``?(x :'a). (R :'a -> bool) x``:
   term
> <<HOL message: inventing new type variable names: 'a>>
val it =
   ``(R :num -> 'a) (x :num)``:
   term
> val it =
   ``?(x :num). (R :num -> bool) x``:
   term
```

Chapter 3

# Excercise 7.3.1

## 3.1 Problem statement

In the following problems, enable HOLs Show types capability and disable Unicode so only ACSII characters are displayed. **Exercise 7.3.1** Create a function *andImp2Imp* term that operates on terms of the form $p \wedge q \supset r$ and returns $p \supset q \supset r$.

## 3.2 Relevant Code

```
val andImpTerm = ``p/\q==>r``
fun andImp2Imp term = ``p==>q==>r``
andImp2Imp andImpTerm
```

## 3.3 Test Case

2

```
------------------------------------------------------------------------
       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 :bool) /\ (q :bool) ==> (r :bool)``:
   term
> val andImp2Imp = fn: 'a -> term
> val it =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
>
Process HOL finished
```

Chapter 4

# Excercise 7.3.2

## 4.1 Problem statement

**Exercise 7.3.2** Create a function *impImpAnd* term that operates on terms of the form $p \supset q \supset r$ and returns $p \vee q \supset r$. Show that *impImpAnd* reverses the effects of *andImp2Imp*, and vice verse.

## 4.2 Relevant Code

```
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;
```

## 4.3 Test Case

3

```
------------------------------------------------------------------------
       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 andImp2Imp = fn: 'a -> term
> > val impImpAnd = fn: 'a -> term
> > val v1 =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
> > val v2 =
   ``(p :bool) /\ (q :bool) ==> (r :bool)``:
   term
> val v3 =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
>
Process HOL finished
```

Chapter 5

# Excercise 7.3.3

## 5.1 Problem statement

**Exercise 7.3.3** Create a function *notExists* term that operates on terms of the form $\neg \exists\ x.P(x)$ and returns $\forall\ x.\neg\ P(x)$.

## 5.2 Relevant Code

```
fun notExists term = ``!x. ~P x``;
notExists ``~?z.Q z``;
```

## 5.3 Test Case

*4*

```
------------------------------------------------------------------------
       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 notExists = 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
>
Process HOL finished
```

**Appendix A**

# Source code for Ex 6.2.1

```
(* ========================================================================= *)
(* Ex 6.2.1                                                                  *)
(* Chirag Sachdev                                                            *)
(*                                                                           *)
(* ========================================================================= *)
(*1*)
``P x ==> Q y``;
(*2*)
``P (x:num) ==> Q (y:bool)``;
(*3*)
``!x. !y. P x ==> Q y``;
(*4*)
``?x. R (x:num)``;
(*5*)
```

**Appendix B**

# Source code for Ex 7.3.1

```
(* ========================================================================= *)
(* Ex 7.3.1                                                                  *)
(* Chirag Sachdev                                                            *)
(*                                                                           *)
(* ========================================================================= *)
val andImpTerm = ``p /\ q ==> r``
fun andImp2Imp term = ``p ==> q ==> r``
andImp2Imp andImpTerm
```

**Appendix C**

# Source code for Ex 7.3.2

```
(* ========================================================================= *)
(* Ex 7.3.2                                                                  *)
(* Chirag Sachdev                                                            *)
(*                                                                           *)
(* ========================================================================= *)

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;
```

**Appendix D**

# Source code for Ex 7.3.3

```
(* ========================================================================= *)
(* Ex 7.3.3                                                                  *)
(* Chirag Sachdev                                                            *)
(*                                                                           *)
(* ========================================================================= *)

fun notExists term = ``!x. ~P x``;
notExists ``~?z.Q z``;
```