Project 3A

Bharath Karumudi

July 26, 2019

Abstract

This project is to demonstrate the capabilities of implementiong constructing and deconstructing HOL Terms using the tools and techniques - LaTeX, AcuTeX, emacs and ML.

Each chapter documents the given problems with a structure of:

1. Problem Statement
2. Relevant Code
3. Test Cases
4. Execution Transcripts
5. Explanation of results

Acknowledgments: Professor Marvine Hamner and Professor Shiu-Kai Chin who taught the Certified Security By Design.

Contents

Chapter 1

Executive Summary

All the requirements for this project are statisfied specifically,

Contents

Our report has the following content:

1. Chapter 1: Executive Summary
2. Chapter 2 Exercise 7.3.1
   (a) Section 2.1 Problem Statement
   (b) Section 2.2 Relevant Code
   (c) Section 2.3 Test Cases
   (d) Section 2.4 Execution Transcripts
   (e) Section 2.4.1 Explanation of Results
3. Chapter 3 Exercise 7.3.2
   (a) Section 3.1 Problem Statement
   (b) Section 3.2 Relevant Code
   (c) Section 3.3 Test Cases
   (d) Section 3.4 Execution Transcripts
   (e) Section 3.4.1 Explanation of Results
4. Chapter 4 Exercise 7.3.3
   (a) Section 4.1 Problem Statement
   (b) Section 4.2 Relevant Code
   (c) Section 4.3 Test Cases
   (d) Section 4.4 Execution Transcripts
   (e) Section 4.4.1 Explanation of Results

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

In this exercise we need to create a function *andImp2Imp term*, which will take:

$$p \wedge q \subset r$$

and results to:

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

2.2 Relevant Code

```
fun andImp2Imp term =
let
 val(conjTerm,r)= dest_imp(term)
 val(p,q) = dest_conj(conjTerm)
in
  mk_imp(p,(mk_imp(q,r)))
end;
```

2.3 Test Cases

The required test cases are:

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

2.4 Execution Transcripts

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
> *** Globals.show_assums now true ***
> # # # # # # # # # ** Unicode trace now off
>
> # # # # # # # val andImp2Imp = fn: term -> term
> >
>
> andImp2Imp ``(p/\q) ==> r``;
val it =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
>
```

2.4.1 Explanation of Results

The above test results shows the test case has been passed.

Chapter 3

Exercise 7.3.2

3.1 Problem Statement

In this exercise, we have to create *andImp2Imp term*, which takes the term

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

and results to:

$$p \wedge q \subset r$$

and also should act as a reverse function for 7.3.1

3.2 Relevant Code

```
(**** 7.3.1 ****)
fun andImp2Imp term =
let
 val(conjTerm,r)= dest_imp(term)
 val(p,q) = dest_conj(conjTerm)

in
  mk_imp(p,(mk_imp(q,r)))
end;

(**** 7.3.2 ****)
fun impImpAnd term =
let

 val(term1,imp) = dest_imp(term)
 val(term2, term3) = dest_imp(imp)
 val new_conj = mk_conj(term1, term2)
in
 mk_imp(new_conj,term3)
end;
```

3.3 Test Cases

The required test cases are:

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

3.4 Execution Transcripts

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
> *** Globals.show_assums now true ***
> # # # # # # # # # ** Unicode trace now off
>
> # # # # # # # 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
> val it =
   ``(p :bool) ==> (q :bool) ==> (r :bool)``:
   term
>
```

3.4.1 Explanation of Results

The above transcript shows the given test cases has been passed.

Chapter 4

Exercise 7.3.3

4.1 Problem Statement

In this exercise we have to create a function *notExists term* which takes the term $\neg\exists x.P(x)$ and returns $\forall x.\neg P(x)$.

4.2 Relevant Code

```
fun notExists term =
let
 val (t1, t2) = dest_exists(dest_neg(term))
in
 mk_forall(t1,t2)
end;
```

4.3 Test Cases

The required test cases are:

```
notExists ``~?z.Q z``;
```

4.4 Execution Transcripts

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
> *** Globals.show_assums now true ***
> # # # # # # # # # ** Unicode trace now off
>
> # # # # # val notExists = fn: term -> term
> >
> <<HOL message: inventing new type variable names: 'a>>
val it =
   ``!(z :'a). (Q :'a -> bool) z``:
   term
>
```

4.4.1 Explanation of Results

The above transcript shows the given tests has been passed.

Chapter 5

Appendix A: Exercise 7.3.1

The following code is from the file ex-7-3-1.sml.

```
(*************************************************************************)
(* Exercise 7.3.1                                                        *)
(* Author: Bharath Karumudi                                              *)
(* Date: Jul 25, 2019                                                    *)
(*************************************************************************)

fun andImp2Imp term =
let
 val(conjTerm,r)= dest_imp(term)
 val(p,q) = dest_conj(conjTerm)

in
   mk_imp(p,(mk_imp(q,r)))
end;

(**** Test Case *****)
andImp2Imp ``(p/\q) ==> r``
```

Chapter 6

Appendix B: Exercise 7.3.2

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

```
(*****************************************************************************)
(* Exercise 7.3.2                                                            *)
(* Author: Bharath Karumudi                                                  *)
(* Date: Jul 25, 2019                                                        *)
(*****************************************************************************)

(**** Function andImp2Imp ~ same as from 7.3.1  ****)

fun andImp2Imp term =
let
 val(conjTerm,r)= dest_imp(term)
 val(p,q) = dest_conj(conjTerm)

in
  mk_imp(p,(mk_imp(q,r)))
end;

(**** Function impImpAnd  ****)

fun impImpAnd term =
let

 val(term1,imp) = dest_imp(term)
 val(term2, term3) = dest_imp(imp)
 val new_conj = mk_conj(term1, term2)
in
 mk_imp(new_conj,term3)
end;

(****** Test Cases **********)

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

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

Appendix C: Exercise 7.3.3

The following code is from the file ex-7-3-3.sml.

```
(*****************************************************************************)
(* Exercise 7.3.3                                                            *)
(* Author: Bharath Karumudi                                                  *)
(* Date: Jul 25, 2019                                                        *)
(*****************************************************************************)


fun notExists term =
let
 val (t1, t2) = dest_exists(dest_neg(term))
in
 mk_forall(t1,t2)
end;


(****** Test Cases **********)
notExists ``~?z.Q z``;
```