Project 3

Alfred Murabito

March 22, 2020

Abstract

This report details results for the following exercises from *Certified Security by Design Using Higher Order Logic*: 7.3.1, 7.3.2, 7.3.3, 8.4.1, 8.4.2, and 8.4.3. The purpose of this project is to gain skill in generating reproducible proofs and documentation using EmitTeX. In addition, HOL Theories will be used to complete the exercises listed above.

Acknowledgments: I received no assistance with this project.

Contents

Chapter 1

Executive Summary

All requirements for this project have been satisfied. A description of each exercise from Chapter 7, the test cases and the results of each exercise as based on HOL's built-in functions are detailed in this project report.

In addition, a HOL subfolder has been created and contains the proofs for the exercises in Chapter 8.

Chapter 2

Exercise 7.3.1

2.1 Problem Statement

In this exercise we define a function that operates on terms of the form $p \ / \ q \ ==¿ \ r$ and returns $p \ ==¿ \ q \ ==¿ \ r$.

2.2 Relevant Code

```
(* Solution to Exercise 7_3_1  *)

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

2.3 Test Cases and Results

Test cases for Exercise 7.3.1:

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

Results for Exercise 7.3.1:

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

Chapter 3

Exercise 7.3.2

3.1 Problem Statement

In this exercise we define a function that accepts terms of the form p ==¿ q ==¿ r and returns p / q ==¿ r. This function is shown to reverse the effect of the function in 7.3.1.

3.2 Relevant Code

```
(* Exercise 7_3_2 *)
(* Turns terms of p ==> q ==> r to (p /\ q) ==> r *)

fun impImpAnd term_i =
let
  val (p, q_imp_r) = dest_imp term_i
  val (q, r) = dest_imp q_imp_r
in
  mk_imp (mk_conj (p, q), r)
end;
```

3.3 Test Cases and Results

We include test cases to show the function's behavior and the reversal of the function from 7.3.1.

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

Results for Exercise 7.3.2:

```
------------------------------------------------------------------------
       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 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
```

Chapter 4

Exercise 7.3.3

4.1 Problem Statement

In this exercise we define a function that accepts terms of the form @x.P(x) and returns !x. P(x).

4.2 Relevant Code

```
(Exercise 7_3_3  *)
(* Given ~?x.P(x) return !x.~P(x)

fun notExists term_i =
let
  val (x, F ) = dest_exists (dest_neg term_i)
in
  mk_forall (x, F )
end;
```

4.3 Test Cases and Results

Test cases for Exercise 7.3.3:

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

Results for Exercise 7.3.3:

```
------------------------------------------------------------------------
       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 notExists = fn: term -> term
> <<HOL message: inventing new type variable names: 'a>>
val it =
   ``!(z :'a). (Q :'a -> bool) z``:
   term
>
```

Chapter 5

Exercise 8.4.1

5.1 Problem Statement

In this exercise we prove the theorem

$\vdash p \Rightarrow (p \Rightarrow q) \Rightarrow (q \Rightarrow r) \Rightarrow r$

5.2 Relevant Code

```
(* Excercise 8_4_1 *)
val problem1Thm =
let
  val p_term = ``p:bool``
  val q_term = ``q:bool``
  val r_term = ``r:bool``
  val p_imp_q = mk_imp (p_term, q_term)
  val q_imp_r = mk_imp (q_term, r_term)
  val thm1 = ASSUME p_term
  val thm2 = ASSUME p_imp_q
  val thm3 = MP (ASSUME p_imp_q) (ASSUME p_term)
  val thm4 = MP (ASSUME q_imp_r) thm3
in
  DISCH p_term (DISCH p_imp_q (DISCH q_imp_r thm4))
end;
```

5.3 Execution Transcript

Transcript for Exercise 8.4.1:

```
------------------------------------------------------------------------
       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
> > # # # # # # # # # # # # # val problem1Thm =
   |- (p :bool) ==> (p ==> (q :bool)) ==> (q ==> (r :bool)) ==> r:
   thm
>
*** Emacs/HOL command completed ***

>
```

Chapter 6

Exercise 8.4.2

6.1 Problem Statement

In this exercise we prove the theorem

$$\vdash q \wedge p \iff p \wedge q$$

6.2 Relevant Code

```
(* Prove p /\ q <=> q /\ p *)
val conjSymThm =
let
  val p_term = ``p:bool``
  val q_term = ``q:bool``
  val p_and_q = mk_conj (p_term, q_term)
  val q_and_p = mk_conj (q_term, p_term)

  (* Prove -| p /\ q ==> q /\ p *)
  val thm1 = ASSUME p_and_q
  val thm2 = CONJUNCT1 thm1
  val thm3 = CONJUNCT2 thm1
  val thm4 = CONJ thm3 thm2
  val thm5 = DISCH p_and_q thm4

  (* Prove -| q /\ p ==> p /\ q *)
  val thm6 = ASSUME q_and_p
  val thm7 = CONJUNCT1 thm6
  val thm8 = CONJUNCT2 thm6
  val thm9 = CONJ thm8 thm7
  val thm10 = DISCH q_and_p thm9


in
  IMP_ANTISYM_RULE thm10 thm5
end;
```

6.3 Execution Transcript

Results for Exercise 8.4.2:

```
------------------------------------------------------------------------
       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 conjSymThm =
   |- (q :bool) /\ (p :bool) <=> p /\ q:
   thm
val it = (): unit
>
*** Emacs/HOL command completed ***

>
```

Chapter 7

Exercise 8.4.3

7.1 Problem Statement

In this exercise we extend upon the previous theorem through generalization to prove the following theorem:

$$\vdash \forall p\ q.\ q \wedge p \iff p \wedge q$$

.

7.2 Relevant Code

```
(* Prove p /\ q <=> q /\ p *)
val conjSymThmAll =
let
  val p_term = ``p:bool``
  val q_term = ``q:bool``
  val p_and_q = mk_conj (p_term, q_term)
  val q_and_p = mk_conj (q_term, p_term)

  (* Prove -| p /\ q ==> q /\ p *)
  val thm1 = ASSUME p_and_q
  val thm2 = CONJUNCT1 thm1
  val thm3 = CONJUNCT2 thm1
  val thm4 = CONJ thm3 thm2
  val thm5 = DISCH p_and_q thm4

  (* Prove -| q /\ p ==> p /\ q *)
  val thm6 = ASSUME q_and_p
  val thm7 = CONJUNCT1 thm6
  val thm8 = CONJUNCT2 thm6
  val thm9 = CONJ thm8 thm7
  val thm10 = DISCH q_and_p thm9
  val thm11 = IMP_ANTISYM_RULE thm10 thm5

in
  GENL [p_term, q_term] thm11
end;
```

7.3 Execution Transcript

Results for Exercise 8.4.3:

Appendix A

Chapter7 Answers Source Code

```
(*****************************************************************************)
(* Author: Alfred Murabito                                                   *)
(* Date: 21 March 2020                                                       *)
(* email: acmurabi@syr.edu                                                   *)
(*****************************************************************************)

(* Solution to Exercise 7_3_1  *)

fun andImp2Imp term_i =
  let
    val (p_and_q, r) = dest_imp term_i
    val (p, q)  = dest_conj p_and_q
  in
    mk_imp (p, mk_imp (q, r))
end;

andImp2Imp ‘‘(p /\ q) ==> r‘‘;

(* Exercise 7_3_2 *)
(* Turns terms of p ==> q ==> r to (p /\ q) ==> r *)

fun impImpAnd term_i =
let
  val (p, q_imp_r) = dest_imp term_i
  val (q, r) = dest_imp q_imp_r
in
  mk_imp (mk_conj (p, q), r)
end;

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


(Exercise 7_3_3  *)
(* Given ~?x.P(x) return !x.~P(x)

fun notExists term_i =
let
  val (x, F ) = dest_exists (dest_neg term_i)
in
  mk_forall (x, F )
end;
```

```
notExists ‘‘~?z.Q z‘‘;
```

Appendix B

Chapter8Script Source Code

```
(*****************************************************************************)
(* Author: Alfred Murabito                                                   *)
(* Date: 21 March 2020                                                       *)
(* email: acmurabi@syr.edu                                                   *)
(*****************************************************************************)

structure chapter8Script = struct

open HolKernel boolLib Parse bossLib

val _ = new_theory "chapter8"

(* Excercise 8_4_1 *)
val problem1Thm =
let
  val p_term = ``p:bool``
  val q_term = ``q:bool``
  val r_term = ``r:bool``
  val p_imp_q = mk_imp (p_term, q_term)
  val q_imp_r = mk_imp (q_term, r_term)
  val thm1 = ASSUME p_term
  val thm2 = ASSUME p_imp_q
  val thm3 = MP (ASSUME p_imp_q) (ASSUME p_term)
  val thm4 = MP (ASSUME q_imp_r) thm3
in
  DISCH p_term (DISCH p_imp_q (DISCH q_imp_r thm4))
end;

val _ = save_thm("problem1Thm",problem1Thm)

(* Exercise 8_4_2 *)

(* Prove p /\ q <=> q /\ p *)
val conjSymThm =
let
  val p_term = ``p:bool``
  val q_term = ``q:bool``
  val p_and_q = mk_conj (p_term, q_term)
  val q_and_p = mk_conj (q_term, p_term)

  (* Prove -| p /\ q ==> q /\ p *)
  val thm1 = ASSUME p_and_q
  val thm2 = CONJUNCT1 thm1
```

```
  val thm3 = CONJUNCT2 thm1
  val thm4 = CONJ thm3 thm2
  val thm5 = DISCH p_and_q thm4

  (* Prove -| q /\ p ==> p /\ q *)
  val thm6 = ASSUME q_and_p
  val thm7 = CONJUNCT1 thm6
  val thm8 = CONJUNCT2 thm6
  val thm9 = CONJ thm8 thm7
  val thm10 = DISCH q_and_p thm9


in
  IMP_ANTISYM_RULE thm10 thm5
end;

val _ = save_thm("conjSymThm",conjSymThm)

(* Exercise 8_4_3 *)

(* Prove p /\ q <=> q /\ p *)
val conjSymThmAll =
let
  val p_term = ``p:bool``
  val q_term = ``q:bool``
  val p_and_q = mk_conj (p_term, q_term)
  val q_and_p = mk_conj (q_term, p_term)

  (* Prove -| p /\ q ==> q /\ p *)
  val thm1 = ASSUME p_and_q
  val thm2 = CONJUNCT1 thm1
  val thm3 = CONJUNCT2 thm1
  val thm4 = CONJ thm3 thm2
  val thm5 = DISCH p_and_q thm4

  (* Prove -| q /\ p ==> p /\ q *)
  val thm6 = ASSUME q_and_p
  val thm7 = CONJUNCT1 thm6
  val thm8 = CONJUNCT2 thm6
  val thm9 = CONJ thm8 thm7
  val thm10 = DISCH q_and_p thm9
  val thm11 = IMP_ANTISYM_RULE thm10 thm5

in
  GENL [p_term, q_term] thm11
end;

val _ = save_thm("conjSymThmAll",conjSymThmAll)

val _ = export_theory ()
val _ = print_theory "-"

end (* struct *)
```