Project 4

Alfred Murabito

March 22, 2020

Abstract

This report details results for the following exercises from *Certified Security by Design Using Higher Order Logic*: 9.5.1, 9.5.2, 9.5.3, 10.4.1, 10.4.2, and 10.4.3. Chapter 9 of the textbook focuses on goal-oriented proofs while Chapter 10 focuses on dealing with assumptions in goal-oriented proofs using PAT_ASSUM to match our supplied terms to those in the assumptions.. Each of the exercises includes a problem statement, relevant code, execution transcripts, and our pretty-printed theorems.

Acknowledgments: I received no assistance with this project.

Contents

1 Executive Summary

All requirements for this project have been satisfied. A description of each exercise from Chapter 9 and 10 are included in the following chapters. Our HOL script files are contained in the HOL directory and pretty-print our theories to reports using EmiTex in the HOLReports directory. In the appendix we also include all HOL source files.

2 Chapter 9 Exercises

2.1 Problem Statement

Exercises 9.5.1 and 9.5.2 involve proving the absorption and constructiveDilemma theorems respectively. In exercise 9.5.3, we prove both theorems again using PROVE_TAC.

2.2 Relevant Code

Exercise 9.5.1

```
val absorptionRule =
TAC_PROOF (
( [], ``!(p:bool)(q:bool).(p ==> q) ==> p ==> (p /\ q)``),
REPEAT STRIP_TAC THENL
[(ACCEPT_TAC (ASSUME ``p:bool``)),
(ACCEPT_TAC (MP (ASSUME ``p:bool==>q:bool``) (ASSUME ``p:bool``)))])
```

Exercise 9.5.2

```
val constructiveDilemmaRule =
TAC_PROOF (
( [], ``!(p:bool)(q:bool)(r:bool)(s:bool).(p==>q)/\(r==>s)==>(p\/r)==>(q\/s)``),
(REPEAT STRIP_TAC THENL
[(RES_TAC THEN (ACCEPT_TAC (DISJ1 (ASSUME ``q:bool``) ``s:bool``))),
(RES_TAC THEN (ACCEPT_TAC (DISJ2 ``q:bool`` (ASSUME ``s:bool``))))])
);
```

Exercise 9.5.3

```
val absorptionRule2 =
TAC_PROOF (
( [], ``!(p:bool)(q:bool).(p ==> q) ==> p ==> (p /\ q)``),
PROVE_TAC [])

val constructiveDilemmaRule2 =
TAC_PROOF (
( [], ``!(p:bool)(q:bool)(r:bool)(s:bool).(p==>q) /\ (r==>s) ==> p ==> (p /\ q)``),
PROVE_TAC [])
```

2.3 Test Results

Exercise 9.5.1:

```
----------------------------------------------------------------------
       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 absorptionRule =
   |- !(p :bool) (q :bool). (p ==> q) ==> p ==> p /\ q:
   thm
>
```

Exercise 9.5.2:

```
----------------------------------------------------------------------
       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 constructiveDilemmaRule =
   |- !(p :bool) (q :bool) (r :bool) (s :bool).
     (p ==> q) /\ (r ==> s) ==> p \/ r ==> q \/ s:
   thm
>
*** Emacs/HOL command completed ***

>
```

Exercise 9.5.3:

```
----------------------------------------------------------------------
       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
> # # # Meson search level: ........
val absorptionRule2 =
   |- !(p :bool) (q :bool). (p ==> q) ==> p ==> p /\ q:
```

```
   thm
> # # # Meson search level: ................
val constructiveDilemmaRule2 =
   |- !(p :bool) (q :bool) (r :bool) (s :bool).
     (p ==> q) /\ (r ==> s) ==> p ==> p /\ q:
   thm
>
```

2.4 Exercise 9 Theory

Built: 22 March 2020
Parent Theories: indexedLists, patternMatches

2.5 Theorems

[absorptionRule]

$\vdash \forall p\ q.\ (p \Rightarrow q) \Rightarrow p \Rightarrow p \land q$

[absorptionRule2]

$\vdash \forall p\ q.\ (p \Rightarrow q) \Rightarrow p \Rightarrow p \land q$

[constructiveDilemmaRule]

$\vdash \forall p\ q\ r\ s.\ (p \Rightarrow q) \land (r \Rightarrow s) \Rightarrow p \lor r \Rightarrow q \lor s$

[constructiveDilemmaRule2]

$\vdash \forall p\ q\ r\ s.\ (p \Rightarrow q) \land (r \Rightarrow s) \Rightarrow p \Rightarrow p \land q$

3 Chapter 10 Exercises

3.1 Problem Statement

We use PAT_ASSUM to deal with assumptions in the goal orientated proofs of chapter 10. All of the theorems were proved without using PROVE_TAC.

3.2 Relevant Code

Exercise 10.4.1

```
val problem1_thm =
TAC_PROOF (
([``!x:'a.P(x) ==> M(x)``,``(P:'a->bool)(s:'a)``], ``(M:'a->bool)(s:'a)``),
PAT_ASSUM ``!x.t`` (fn th => (ASSUME_TAC (SPEC ``s:'a`` th))) THEN
RES_TAC );
```

Exercise 10.4.2

```
val problem2_thm =
TAC_PROOF (
([``p /\ q ==> r``,``r ==> s``,`` ~s``],``p ==> ~q``),
PAT_ASSUM ``r ==> s`` (fn th => ASSUME_TAC (IMP_ELIM th)) THEN
PAT_ASSUM ``~r:bool \/ s:bool`` (fn th => ASSUME_TAC (ONCE_REWRITE_RULE[DISJ_SYM] th)) THEN
PAT_ASSUM ``s:bool \/ ~r:bool`` (fn th => ASSUME_TAC (DISJ_IMP th)) THEN
RES_TAC THEN
PAT_ASSUM ``p:bool /\ q:bool ==> r:bool`` (fn th => ASSUME_TAC (ONCE_REWRITE_RULE[DISJ_SYM](IMP_ELIM th))) THEN
PAT_ASSUM ``r:bool \/ ~(p /\ q)`` (fn th => ASSUME_TAC (DISJ_IMP th)) THEN
RES_TAC THEN
(let
  val demorgan = SPEC ``q:bool`` (SPEC ``p:bool`` DE_MORGAN_THM)
in
  PAT_ASSUM ``~(p /\ q)``(fn th => ASSUME_TAC (REWRITE_RULE [] (DISJ_IMP (EQ_MP (CONJUNCT1 demorgan) (ASSUME ``~(p /\ q)``)))))
end) THEN
ASM_REWRITE_TAC []
);
```

Exercise 10.4.3

```
val problem3_thm =
TAC_PROOF (
([`` ~(p /\ q)``, `` ~p ==> r``,`` ~q ==> s``],``r \/ s ``),
(let
  val demorgan = SPEC ``q:bool`` (SPEC ``p:bool`` DE_MORGAN_THM)
in
  PAT_ASSUM ``~(p /\ q)`` (fn th => ASSUME_TAC ((EQ_MP (CONJUNCT1 demorgan) (ASSUME ``~(p /\ q)``))))
end) THEN
PAT_ASSUM ``~p \/ ~q`` (fn th => ASSUME_TAC (REWRITE_RULE [] (DISJ_IMP th))) THEN
PAT_ASSUM ``p ==> ~q`` (fn th => ASSUME_TAC (IMP_TRANS th (ASSUME ``~q ==> s``))) THEN
PAT_ASSUM ``p ==> s`` (fn th => ASSUME_TAC (DISJ_IMP(ONCE_REWRITE_RULE[DISJ_SYM](IMP_ELIM th)))) THEN
PAT_ASSUM ``~s ==> ~p`` (fn th => ASSUME_TAC (IMP_TRANS th (ASSUME ``~p ==> r``))) THEN
PAT_ASSUM ``~s ==> r`` (fn th => ASSUME_TAC (REWRITE_RULE [](IMP_ELIM th))) THEN
ASM_REWRITE_TAC [DISJ_SYM]
)
```

3.3 Test Results

Exercise 10.4.1:

```
---------------------------------------------------------------------
       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
> *** Globals.show_assums now true ***
> # # # # <<HOL message: inventing new type variable names: 'a>>
val problem1_thm =

[(P :'a -> bool) (s :'a),
 !(x :'a). (P :'a -> bool) x ==> (M :'a -> bool) x]
|- (M :'a -> bool) (s :'a):
   thm
>
```

Exercise 10.4.2:

```
---------------------------------------------------------------------
       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
> *** Globals.show_assums now true ***
> val problem2_thm =

[~(s :bool), (r :bool) ==> (s :bool),
 (p :bool) /\ (q :bool) ==> (r :bool)] |- (p :bool) ==> ~(q :bool):
   thm
val it = (): unit
>
*** Emacs/HOL command completed ***

>
```

Exercise 10.4.3:

```
---------------------------------------------------------------------
       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
> *** Globals.show_assums now true ***
> val problem3_thm =

[~((p :bool) /\ (q :bool)), ~(p :bool) ==> (r :bool),
 ~(q :bool) ==> (s :bool)] |- (r :bool) \/ (s :bool):
   thm
val it = (): unit
>
*** Emacs/HOL command completed ***

>
```

3.4 Exercise 10 Theory

Built: 22 March 2020
Parent Theories: indexedLists, patternMatches

3.5 Theorems

[problem1_thm]

$\vdash M\ s$

[problem2_thm]

$\vdash p \Rightarrow \neg q$

[problem3_thm]

$\vdash r \lor s$

A Source Code for Exercise9 Theory

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

structure exercise9Script = struct

open HolKernel boolLib Parse bossLib

val _ = new_theory "exercise9"

(*Exercise 9.5.1 Proof *)
val absorptionRule =
TAC_PROOF (
( [], ``!(p:bool)(q:bool).(p ==> q) ==> p ==> (p /\ q)``),
REPEAT STRIP_TAC THENL
[(ACCEPT_TAC (ASSUME ``p:bool``)),
(ACCEPT_TAC (MP (ASSUME ``p:bool==>q:bool``) (ASSUME ``p:bool``)))])

val _ = save_thm("absorptionRule",absorptionRule)

(* Exercise 9.5.2 *)

val constructiveDilemmaRule =
TAC_PROOF (
( [], ``!(p:bool)(q:bool)(r:bool)(s:bool).(p==>q)/\(r==>s)==>(p\/r)==>(q\/s)``),
(REPEAT STRIP_TAC THENL
[(RES_TAC THEN (ACCEPT_TAC (DISJ1 (ASSUME ``q:bool``) ``s:bool``))),
(RES_TAC THEN (ACCEPT_TAC (DISJ2 ``q:bool`` (ASSUME ``s:bool``))))])
);

val _ = save_thm("constructiveDilemmaRule",constructiveDilemmaRule)

(* ex9_5_3.sml for Exercise 9.5.3 *)

(* Prove ! p q. (p  q)  p  p  q *)
val absorptionRule2 =
TAC_PROOF (
( [], ``!(p:bool)(q:bool).(p ==> q) ==> p ==> (p /\ q)``),
PROVE_TAC [])

(*  p q r s. (p  q)  (r  s)  p  r  q  s *)

val constructiveDilemmaRule2 =
```

```
TAC_PROOF (
( [], ``!(p:bool)(q:bool)(r:bool)(s:bool).(p==>q) /\ (r==>s) ==> p ==> (p /\ q)``),
PROVE_TAC [])

val _ = save_thm("absorptionRule2",absorptionRule2)

val _ = save_thm("constructiveDilemmaRule2",constructiveDilemmaRule2)

val _ = print_theory "-";

val _ = export_theory();

end (* structure *)
```

B Source Code for Exercise10 Theory

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

structure exercise9Script = struct

open HolKernel boolLib Parse bossLib

val _ = new_theory "exercise10"

(* Exercise 10_4_1
set_goal
([``!x:'a.P(x) ==> M(x)``,``(P:'a->bool)(s:'a)``],
``(M:'a->bool)(s:'a)``);
*)

(*  Proof Below   *)
val problem1_thm =
TAC_PROOF (
([``!x:'a.P(x) ==> M(x)``,``(P:'a->bool)(s:'a)``], ``(M:'a->bool)(s:'a)``),
PAT_ASSUM ``!x.t`` (fn th => (ASSUME_TAC (SPEC ``s:'a`` th))) THEN
RES_TAC );

val _ = save_thm("problem1_thm",problem1_thm)

(* Solution to exercise 10.4.2
set_goal([``p /\ q ==> r``,``r ==> s``,`` ~s``],``p ==> ~q``);

PAT_ASSUM ``r ==> s`` (fn th => ASSUME_TAC (IMP_ELIM th));
PAT_ASSUM ``~r:bool \/ s:bool`` (fn th => ASSUME_TAC (ONCE_REWRITE_RULE[DISJ_SYM] th));
PAT_ASSUM ``s:bool \/ ~r:bool`` (fn th => ASSUME_TAC (DISJ_IMP th));
RES_TAC;
PAT_ASSUM ``p:bool /\ q:bool ==> r:bool`` (fn th => ASSUME_TAC (ONCE_REWRITE_RULE[DISJ_SYM](IMP_E
PAT_ASSUM ``r:bool \/ ~(p /\ q)`` (fn th => ASSUME_TAC (DISJ_IMP th));
(* PAT_ASSUM ``~r ==> ~(p /\ q)`` (fn th => ASSUME_TAC (MP th (ASSUME ``~r``))) *);
RES_TAC
let
  val demorgan = SPEC ``q:bool`` (SPEC ``p:bool`` DE_MORGAN_THM)
in
  PAT_ASSUM ``~(p /\ q)``(fn th => ASSUME_TAC (REWRITE_RULE [] (DISJ_IMP (EQ_MP (CONJUNCT1 demorg
end;
ASM_REWRITE_TAC [];
*)
```

```
(* Packed together in TAC_PROOF *)
val problem2_thm =
TAC_PROOF (
([``p /\ q ==> r``,``r ==> s``,`` ~s``],``p ==> ~q``),
PAT_ASSUM ``r ==> s`` (fn th => ASSUME_TAC (IMP_ELIM th)) THEN
PAT_ASSUM ``~r:bool \/ s:bool`` (fn th => ASSUME_TAC (ONCE_REWRITE_RULE[DISJ_SYM] th)) THEN
PAT_ASSUM ``s:bool \/ ~r:bool`` (fn th => ASSUME_TAC (DISJ_IMP th)) THEN
RES_TAC THEN
PAT_ASSUM ``p:bool /\ q:bool ==> r:bool`` (fn th => ASSUME_TAC (ONCE_REWRITE_RULE[DISJ_SYM](
PAT_ASSUM ``r:bool \/ ~(p /\ q)`` (fn th => ASSUME_TAC (DISJ_IMP th)) THEN
RES_TAC THEN
(let
  val demorgan = SPEC ``q:bool`` (SPEC ``p:bool`` DE_MORGAN_THM)
in
  PAT_ASSUM ``~(p /\ q)``(fn th => ASSUME_TAC (REWRITE_RULE [] (DISJ_IMP (EQ_MP (CONJUNCT1 d
end) THEN
ASM_REWRITE_TAC []
);

val _ = save_thm("problem2_thm",problem2_thm)

(* Exercise 10_4_3
set_goal([`` ~(p /\ q)``, `` ~p ==> r``,`` ~q ==> s``],``r \/ s ``);

(let
  val demorgan = SPEC ``q:bool`` (SPEC ``p:bool`` DE_MORGAN_THM)
in
  PAT_ASSUM ``~(p /\ q)`` (fn th => ASSUME_TAC ((EQ_MP (CONJUNCT1 demorgan) (ASSUME ``~(p /\
end);
PAT_ASSUM ``~p \/ ~q`` (fn th => ASSUME_TAC (REWRITE_RULE [] (DISJ_IMP th)));
PAT_ASSUM ``p ==> ~q`` (fn th => ASSUME_TAC (IMP_TRANS th (ASSUME ``~q ==> s``)));
PAT_ASSUM ``p ==> s`` (fn th => ASSUME_TAC (DISJ_IMP(ONCE_REWRITE_RULE[DISJ_SYM](IMP_ELIM th
PAT_ASSUM ``~s ==> ~p`` (fn th => ASSUME_TAC (IMP_TRANS th (ASSUME ``~p ==> r``)));
PAT_ASSUM ``~s ==> r`` (fn th => ASSUME_TAC (REWRITE_RULE [](IMP_ELIM th)));
ASM_REWRITE_TAC [DISJ_SYM];
*)

(* Bundled together using TAC_PROOF  *)
val problem3_thm =
TAC_PROOF (
([`` ~(p /\ q)``, `` ~p ==> r``,`` ~q ==> s``],``r \/ s ``),
(let
  val demorgan = SPEC ``q:bool`` (SPEC ``p:bool`` DE_MORGAN_THM)
in
  PAT_ASSUM ``~(p /\ q)`` (fn th => ASSUME_TAC ((EQ_MP (CONJUNCT1 demorgan) (ASSUME ``~(p /\
end) THEN
```

```
PAT_ASSUM ``~p \/ ~q`` (fn th => ASSUME_TAC (REWRITE_RULE [] (DISJ_IMP th))) THEN
PAT_ASSUM ``p ==> ~q`` (fn th => ASSUME_TAC (IMP_TRANS th (ASSUME ``~q ==> s``))) THEN
PAT_ASSUM ``p ==> s`` (fn th => ASSUME_TAC (DISJ_IMP(ONCE_REWRITE_RULE[DISJ_SYM](IMP_ELIM th))))
PAT_ASSUM ``~s ==> ~p`` (fn th => ASSUME_TAC (IMP_TRANS th (ASSUME ``~p ==> r``))) THEN
PAT_ASSUM ``~s ==> r`` (fn th => ASSUME_TAC (REWRITE_RULE [](IMP_ELIM th))) THEN
ASM_REWRITE_TAC [DISJ_SYM]
)

val _ = save_thm("problem3_thm",problem3_thm)

val _ = print_theory "-";

val _ = export_theory();

end (* structure *)
```