Project 4 Report

Kyle Peppe

February 2, 2020

Abstract

This project was mainly to use ML to set a goal and then use ML tactics to prove a goal or to set a theorem. The exercises became more difficult with chapter 10 being a bit more difficult. I have completed the exercises 9.5.1, 9.5.2, 9.5.3, 10.4,1, 10.4.2, and 10.4.3.

- 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

.

Contents

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.

Chapter 1

Executive Summary

All requirements for this project are satisfied. Specifically,

Report Contents

Our report has the following content:

Reproducibility in ML and LaTeX

The ML and LaTeX source files compile with no errors.

Chapter 2

Excercise 9.5.1

2.1 Problem statement

Do a tactic based proof of the absorption rule:

```
``!(p:bool)(q:bool).(p ==> q) ==> p ==> p /\ q``
```

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``)),
        RES_TAC])
);

(* Save the theorem                                            *)
val _ = save_thm("absorptionRule", absorptionRule);
```

2.3 Session Transcript

```
> val absorptionRule =
TAC_PROOF
(
([], ``!(p:bool)(q:bool).(p ==> q) ==> p ==> p /\ q``),
(REPEAT STRIP_TAC THENL
[(ACCEPT_TAC (ASSUME ``p:bool``)),
RES_TAC])
);
# # # # # # # val absorptionRule =
   |- !(p :bool) (q :bool). (p ==> q) ==> p ==> p /\ q:
   thm
>
```

1

Chapter 3

Excercise 9.5.2

3.1 Problem statement

Do a tactic based proof of the absorption ruleProve the following theorem:

```
``!(p:bool)(q:bool)(r:bool)(s:bool).(p ==> q) /\ (r ==> s) ==> p \/ r ==> q \/ s``)
```

3.2 Relevant Code

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

3.3 Session Transcript

```
> val constructiveDilemmaRule =
TAC_PROOF                                                          2
(
([], ``!(p:bool)(q:bool)(r:bool)(s:bool).
(p ==> q) /\ (r ==> s) ==> p \/ r ==> q \/ s``),
(REPEAT STRIP_TAC THEN
RES_TAC THEN
ASM_REWRITE_TAC[] THEN
RES_TAC THEN
ASM_REWRITE_TAC[])
);
# # # # # # # # # # val constructiveDilemmaRule =
   |- !(p :bool) (q :bool) (r :bool) (s :bool).
     (p ==> q) /\ (r ==> s) ==> p \/ r ==> q \/ s:
   thm
>
```

Chapter 4

Excercise 9.5.3

4.1 Problem statement

For this exercise we use prove tac on the Exercises 9.5.1 and 9.5.2.

4.2 Relevant Code

```
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 \/ r ==> q \/ s``),
        (PROVE_TAC[])
);
```

4.3 Test Case

```
> val absorptionRule2 =
TAC_PROOF                                                            3
(
([], ``!(p:bool)(q:bool).(p ==> q) ==> p ==> p /\ q``),
(PROVE_TAC[])
);
# # # # # Meson search level: ........
val absorptionRule2 =
   |- !(p :bool) (q :bool). (p ==> q) ==> p ==> p /\ q:
   thm
> val constructiveDilemmaRule2 =
TAC_PROOF
(
([], ``!(p:bool)(q:bool)(r:bool)(s:bool).
(p ==> q) /\ (r==> s) ==> p \/ r ==> q \/ s``),
(PROVE_TAC[])
);
# # # # # # Meson search level: ................
val constructiveDilemmaRule2 =
   |- !(p :bool) (q :bool) (r :bool) (s :bool).
     (p ==> q) /\ (r ==> s) ==> p \/ r ==> q \/ s:
   thm
>
```

Chapter 5

Excercise 10.4.1

5.1 Problem statement

Prove the goal without using prove tac

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

5.2 Relevant Code

```
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`` th))) THEN
        RES_TAC)
);
```

5.3 Test Case

```
> 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`` th))) THEN
RES_TAC)
);
# # # # # # <<HOL message: inventing new type variable names: 'a>>
val problem1_thm =
    [..] |- (M :'a -> bool) (s :'a):
   thm
>
```

4

Chapter 6

Excercise 10.4.2

6.1 Problem statement

Prove the goal without using prove tac

```
([``p /\ q ==> r``, ``r ==> s``, ``~s``], ``p ==> ~q``)
```

6.2 Relevant Code

```
val problem2_thm =
TAC_PROOF
(
        ([``p /\ q ==> r``, ``r ==> s``, ``~s``], ``p ==> ~q``),
        (REPEAT STRIP_TAC THEN
        REPEAT RES_TAC)
);
```

6.3 Test Case

```
> val problem2_thm =
TAC_PROOF
(
([``p /\ q ==> r``, ``r ==> s``, ``~s``], ``p ==> ~q``),
(REPEAT STRIP_TAC THEN
REPEAT RES_TAC)
);
# # # # # # val problem2_thm =
    [...] |- (p :bool) ==> ~(q :bool):
   thm
>
```

Chapter 7

Excercise 10.4.3

7.1 Problem statement

Prove the goal without using prove tac

```
([‘‘~(p /\ q)‘‘, ‘‘~p ==> r‘‘, ‘‘~q ==> s‘‘], ‘‘r \/ s‘‘)
```

7.2 Relevant Code

```
val problem3_thm =
TAC_PROOF
(
        ([‘‘~(p /\ q)‘‘, ‘‘~p ==> r‘‘, ‘‘~q ==> s‘‘], ‘‘r \/ s‘‘),
        (PAT_ASSUM ‘‘A ==> B‘‘ (fn th => ASSUME_TAC(REWRITE_RULE[]
        (DISJ_IMP(ONCE_REWRITE_RULE [DISJ_SYM] (IMP_ELIM th))))) THEN
        PAT_ASSUM ‘‘~(A /\ B)‘‘ (fn th => (ASSUME_TAC(REWRITE_RULE[]
        (DISJ_IMP(REWRITE_RULE [DE_MORGAN_THM] th))))) THEN
        ASSUME_TAC(IMP_TRANS (ASSUME ‘‘p ==> ~q‘‘) (ASSUME ‘‘~q ==> s‘‘)) THEN
        ASSUME_TAC(IMP_TRANS (ASSUME ‘‘~r ==> p‘‘) (ASSUME ‘‘p ==> s‘‘)) THEN
        PAT_ASSUM ‘‘A ==> B‘‘ (fn th => (ASSUME_TAC (REWRITE_RULE[] (IMP_ELIM th))))
        THEN ASM_REWRITE_TAC[])
);
```

7.3 Test Case

```
> val problem3_thm =
TAC_PROOF
(
([‘‘~(p /\ q)‘‘, ‘‘~p ==> r‘‘, ‘‘~q ==> s‘‘], ‘‘r \/ s‘‘),
(PAT_ASSUM ‘‘A ==> B‘‘ (fn th => ASSUME_TAC(REWRITE_RULE[]
(DISJ_IMP(ONCE_REWRITE_RULE [DISJ_SYM] (IMP_ELIM th))))) THEN
PAT_ASSUM ‘‘~(A /\ B)‘‘ (fn th => (ASSUME_TAC(REWRITE_RULE[]
(DISJ_IMP(REWRITE_RULE [DE_MORGAN_THM] th))))) THEN
ASSUME_TAC(IMP_TRANS (ASSUME ‘‘p ==> ~q‘‘) (ASSUME ‘‘~q ==> s‘‘)) THEN
ASSUME_TAC(IMP_TRANS (ASSUME ‘‘~r ==> p‘‘) (ASSUME ‘‘p ==> s‘‘)) THEN
PAT_ASSUM ‘‘A ==> B‘‘ (fn th => (ASSUME_TAC (REWRITE_RULE[] (IMP_ELIM th))))
THEN ASM_REWRITE_TAC[])
);
# # # # # # # # # # # # val problem3_thm =
    [...] |- (r :bool) \/ (s :bool):
   thm
>
```

6

Appendix A

Source code

```
(*************************************************************************)
(* Author: Kyle Peppe                                                    *)
(* Exercises 9.5.1, 9.5.2, and 9.5.3                                     *)
(* Date: 1/26/20                                                         *)
(*************************************************************************)

(* Commands needed to create the theory file              *)
structure exercise9Script = struct

open HolKernel Parse boolLib bossLib;

val _ = new_theory "exercise9";


(* 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``)),
        RES_TAC])
);

(* Save the theorem                                       *)
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 THEN
        RES_TAC THEN
        ASM_REWRITE_TAC[] THEN
        RES_TAC THEN
        ASM_REWRITE_TAC[])
);

(* Save the theorem                                       *)
val _ = save_thm("constructiveDeilemmaRule", constructiveDilemmaRule);
```

```
(* Exercise 9.5.3                                         *)
(* Using PROVE_TAC on Exercise 9.5.1                      *)
val absorptionRule2 =
TAC_PROOF
(
        ([], ``!(p:bool)(q:bool).(p ==> q) ==> p ==> p /\ q``),
        (PROVE_TAC[])
);

(* Save the theorem                                       *)
val _ = save_thm("absorptionRule2", absorptionRule2);


(* Exercise 9.5.3                                         *)
(* Using PROVE_TAC on Exercise 9.5.2                      *)
val constructiveDilemmaRule2 =
TAC_PROOF
(
        ([], ``!(p:bool)(q:bool)(r:bool)(s:bool).
        (p ==> q) /\ (r==> s) ==> p \/ r ==> q \/ s``),
        (PROVE_TAC[])
);

(* Save the theorem                                       *)
val _ = save_thm("constructiveDilemmaRule2", constructiveDilemmaRule2);

(* Command to export the above theories                   *)
val _ = export_theory();

end

(***************************************************************************)
(* Author: Kyle Peppe                                                      *)
(* Exercises 10.4.1, 10.4.2, and 10.4.3                                    *)
(* Date: 1/28/20                                                           *)
(***************************************************************************)

(* Commands that help create the theory file              *)
structure exercise10Script = struct

open HolKernel Parse boolLib bossLib;

val _ = new_theory "exercise10";

(* 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`` th))) THEN
        RES_TAC)
```

```
);

(* Save the Theorem                                          *)
val _ = save_thm("problem1_thm", problem1_thm);


(* Exercise 10.4.2                                           *)
val problem2_thm =
TAC_PROOF
(
        ([``p /\ q ==> r``, ``r ==> s``, ``~s``], ``p ==> ~q``),
        (REPEAT STRIP_TAC THEN
        REPEAT RES_TAC)
);

(* Save the Theorem                                          *)
val _ = save_thm("problem2_thm", problem2_thm);


(* Exercise 10.4.3                                           *)
val problem3_thm =
TAC_PROOF
(
        ([``~(p /\ q)``, ``~p ==> r``, ``~q ==> s``], ``r \/ s``),
        (PAT_ASSUM ``A ==> B`` (fn th => ASSUME_TAC(REWRITE_RULE[]
        (DISJ_IMP(ONCE_REWRITE_RULE [DISJ_SYM] (IMP_ELIM th))))) THEN
        PAT_ASSUM ``~(A /\ B)`` (fn th => (ASSUME_TAC(REWRITE_RULE[]
        (DISJ_IMP(REWRITE_RULE [DE_MORGAN_THM] th))))) THEN
        ASSUME_TAC(IMP_TRANS (ASSUME ``p ==> ~q``) (ASSUME ``~q ==> s``)) THEN
        ASSUME_TAC(IMP_TRANS (ASSUME ``~r ==> p``) (ASSUME ``p ==> s``)) THEN
        PAT_ASSUM ``A ==> B`` (fn th => (ASSUME_TAC (REWRITE_RULE[] (IMP_ELIM th))))
        THEN ASM_REWRITE_TAC[])
);

(* Save the Theorem                                          *)
val _ = save_thm("problem3thm", problem3_thm);


(*Export theory and ending commands                          *)
val _ = export_theory();

end
```