Project 5 Report

Kyle Peppe

February 9, 2020

Abstract

This project was to demonstrate my command of using the new tacticals we have learned in order to prove theorems. This was also the first set of exercises where I had to load and open HOL saved theory files. The exercises became more difficult with chapter 10 being a bit more difficult. I have completed the exercises 11.6.1, 11.6.2, and 11.6.3.

- 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

1 Executive Summary **2**

2 Excercise 11.6.1 **3**
2.1 Problem statement 3
2.2 Relevant Code 3
2.3 Session Transcript 3

3 Excercise 11.6.2 **4**
3.1 Problem statement 4
3.2 Relevant Code 4
3.3 Session Transcript 5

4 Excercise 11.6.3 **6**
4.1 Problem statement 6
4.2 Relevant Code 6
4.3 Session Transcripts 8

A Source code **9**

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:

Chapter 1: Executive Summary
Chapter 2: Exercise 11.6.1
Section 2.1: Problem Statement
Section 2.2: Relevant Code
Section 2.3: Session Transcripts
Chapter 3: Exercise 11.6.2
Section 3.1: Problem Statement
Section 3.2: Relevant Code
Section 3.3: Session Transcripts
Chapter 4: Exercise 11.6.3
Section 4.1: Problem Statement
Section 4.2: Relevant Code
Section 4.3: Session Transcripts
Appendix A: Source Code

Reproducibility in ML and LaTeX

The ML and LaTeX source files compile with no errors.

Chapter 2

Excercise 11.6.1

2.1 Problem statement

Add to the theory exTypeTheory by proving the following theorem

```
``!(l1:'a list)(l2:'a list).LENGTH (APP l1 l2) = LENGTH l1 + LENGTH l2``)
```

2.2 Relevant Code

```
(* Exercise 11.6.1                          *)
val LENGTH_APP =
TAC_PROOF(
        ([], ``!(l1:'a list)(l2:'a list).LENGTH (APP l1 l2)
        = LENGTH l1 + LENGTH l2``),
        Induct_on `l1` THEN
        ASM_REWRITE_TAC [APP_def, LENGTH, ADD_CLAUSES]
);

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

2.3 Session Transcript

```
> val LENGTH_APP =
TAC_PROOF(
([], ``!(l1:'a list)(l2:'a list).LENGTH (APP l1 l2)
= LENGTH l1 + LENGTH l2``),
Induct_on `l1` THEN
ASM_REWRITE_TAC [APP_def, LENGTH, ADD_CLAUSES]
);
# # # # # # val LENGTH_APP =
   [oracles: DISK_THM] [axioms: ] []
|- !(l1 :'a list) (l2 :'a list).
     LENGTH (APP l1 l2) = LENGTH l1 + LENGTH l2:
   thm
>
```

1

Chapter 3

Excercise 11.6.2

3.1 Problem statement

Add to the theory exTypeTheory by defining the function Map (Map f [1;2;3;4] = [f1; f2; f3; f4] and then prove the following theorem

```
``Map f(APP l1 l2) = APP (Map f l1) (Map f l2)``)
```

3.2 Relevant Code

```
(* Exercise 11.6.2                              *)
val Map_def =
Define
        `(Map (f:'a -> 'b) [] = []) /\ (Map f (h::(l1:'a list))
        = (f h)::Map f (l1:'a list))`;


val Map_APP =
TAC_PROOF (
          ([], ``Map f(APP l1 l2) = APP (Map f l1) (Map f l2)``),
          Induct_on `l1` THEN
          ASM_REWRITE_TAC[Map_def, APP_def]
);

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

3.3 Session Transcript

```
> val Map_def =
Define                                                                    2
`(Map (f:'a -> 'b) [] = []) /\ (Map f (h::(l1:'a list))
= (f h)::Map f (l1:'a list))`;
# # # Definition has been stored under "Map_def"
val Map_def =
   [oracles: DISK_THM] [axioms: ] []
|- (!(f :'a -> 'b). Map f ([] :'a list) = ([] :'b list)) /\
   !(f :'a -> 'b) (h :'a) (l1 :'a list). Map f (h::l1) = f h::Map f l1:
   thm
> val Map_APP =
TAC_PROOF (
  ([], ``Map f(APP l1 l2) = APP (Map f l1) (Map f l2)``),
  Induct_on `l1` THEN
  ASM_REWRITE_TAC[Map_def, APP_def]
);
# # # # # <<HOL message: inventing new type variable names: 'a, 'b>>
val Map_APP =
   [oracles: DISK_THM] [axioms: ] []
|- Map (f :'b -> 'a) (APP (l1 :'b list) (l2 :'b list)) =
   APP (Map f l1) (Map f l2):
   thm
>
```

Chapter 4

Excercise 11.6.3

4.1 Problem statement

Solve the below theorems after using/setting up the new datatypes, definitions, and theorems: datatype nexp, definition of semantics of nexp experssions.

```
Add_0:
  ``!(f:nexp).nexpVal (Add (Num 0) f) = nexpVal f``

Add_SYM:
``!(f1:nexp)(f2:nexp).nexpVal (Add f1 f2) = nexpVal (Add f2 f1)``

Sub_0:
``!(f:nexp).(nexpVal (Sub (Num 0) f) = 0) /\
   (nexpVal (Sub f (Num 0)) = nexpVal f)``

Mult_ASSOC:
``!(f1:nexp)(f2:nexp)(f3:nexp).nexpVal (Mult f1 (Mult f2 f3)) =
        nexpVal (Mult (Mult f1 f2) f3)``
```

4.2 Relevant Code

```
(* Add_0                                    *)
val Add_0 =
TAC_PROOF(
        ([], ``!(f:nexp).nexpVal (Add (Num 0) f) = nexpVal f``),
        Induct_on `f` THEN
        ASM_REWRITE_TAC[ADD_CLAUSES, nexpValDef]
);

(* Save the Theorem                         *)
val _ = save_thm("Add_0", Add_0);

(* Add_SYM                                  *)
val Add_SYM =
TAC_PROOF(
        ([], ``!(f1:nexp)(f2:nexp).nexpVal (Add f1 f2) = nexpVal (Add f2 f1)``),
        Induct_on `f1` THEN
        PROVE_TAC[nexpValDef, ADD_SYM]
);

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

```
(* Sub_0                                    *)
val Sub_0 =
TAC_PROOF(
        ([],``!(f:nexp).(nexpVal (Sub (Num 0) f) = 0) /\
        (nexpVal (Sub f (Num 0)) = nexpVal f)``),
        Induct_on `f` THEN
        PROVE_TAC[nexpValDef, SUB_0]
);

(* Save the Theorem                         *)
val _ = save_thm("Sub_0", Sub_0);

(* Mult_ASSOC                               *)
val Mult_ASSOC =
TAC_PROOF(
        ([],``!(f1:nexp)(f2:nexp)(f3:nexp).nexpVal (Mult f1 (Mult f2 f3)) =
        nexpVal (Mult (Mult f1 f2) f3)``),
        Induct_on `f1` THEN
        ASM_REWRITE_TAC[MULT_ASSOC, nexpValDef]
);

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

4.3 Session Transcripts

3

```
> val Add_0 =
TAC_PROOF(
([], ``!(f:nexp).nexpVal (Add (Num 0) f) = nexpVal f``),
Induct_on `f` THEN
ASM_REWRITE_TAC[ADD_CLAUSES, nexpValDef]
);
# # # # # val Add_0 =
    [] |- !(f :nexp). nexpVal (Add (Num (0 :num)) f) = nexpVal f:
   thm
> val Add_SYM =
TAC_PROOF(
([], ``!(f1:nexp)(f2:nexp).nexpVal (Add f1 f2) = nexpVal (Add f2 f1)``),
Induct_on `f1` THEN
PROVE_TAC[nexpValDef, ADD_SYM]
);
# # # # # Meson search level: ..........
Meson search level: ..........
Meson search level: ..........
Meson search level: ..........
val Add_SYM =
    [] |- !(f1 :nexp) (f2 :nexp). nexpVal (Add f1 f2) = nexpVal (Add f2 f1):
   thm
> val Sub_0 =
TAC_PROOF(
([],``!(f:nexp).(nexpVal (Sub (Num 0) f) = 0) /\
(nexpVal (Sub f (Num 0)) = nexpVal f)``),
Induct_on`f` THEN
PROVE_TAC[nexpValDef, SUB_0]
);
# # # # # # Meson search level: .......................
Meson search level: .......................
Meson search level: .......................
Meson search level: .......................
val Sub_0 =
    []
|- !(f :nexp).
     (nexpVal (Sub (Num (0 :num)) f) = (0 :num)) /\
     (nexpVal (Sub f (Num (0 :num))) = nexpVal f):
   thm
> val Mult_ASSOC =
TAC_PROOF(
([],``!(f1:nexp)(f2:nexp)(f3:nexp).nexpVal (Mult f1 (Mult f2 f3)) =
nexpVal (Mult (Mult f1 f2) f3)``),
Induct_on`f1` THEN
ASM_REWRITE_TAC[MULT_ASSOC, nexpValDef]
);
# # # # # # val Mult_ASSOC =
    []
|- !(f1 :nexp) (f2 :nexp) (f3 :nexp).
     nexpVal (Mult f1 (Mult f2 f3)) = nexpVal (Mult (Mult f1 f2) f3):
   thm
>
```

Appendix A

Source code

```
(*************************************************************************)
(* Author: Kyle Peppe                                                    *)
(* Exercises 11.6.1 and 11.6.2                                           *)
(* Date: 2/3/20                                                          *)
(*************************************************************************)

(* Beginning theory commands                    *)
structure exTypeScript = struct
open HolKernel Parse boolLib bossLib;
open listTheory TypeBase arithmeticTheory

val _ = new_theory "exType";

(* APP_def from the book                        *)
val APP_def =
Define
        `(APP [] (l:'a list) = l) /\ (APP (h::(l1:'a list)) (l2:'a list)
        = h::(APP l1 l2))`;

(* APP_ASSOC from the book                      *)
val APP_ASSOC =
TAC_PROOF(
        ([],
        ``!(l1:'a list)(l2:'a list)(l3:'a list).(APP(APP l1 l2) l3)
        = (APP l1 (APP l2 l3))``),
        Induct_on `l1` THEN
        PROVE_TAC[APP_def]
);

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

(* Exercise 11.6.1                              *)
val LENGTH_APP =
TAC_PROOF(
        ([], ``!(l1:'a list)(l2:'a list).LENGTH (APP l1 l2)
        = LENGTH l1 + LENGTH l2``),
        Induct_on `l1` THEN
        ASM_REWRITE_TAC [APP_def, LENGTH, ADD_CLAUSES]
);

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

```
(* Exercise 11.6.2                              *)
val Map_def =
Define
        `(Map (f:'a -> 'b) [] = []) /\ (Map f (h::(l1:'a list))
        = (f h)::Map f (l1:'a list))`;


val Map_APP =
TAC_PROOF (
          ([], ``Map f(APP l1 l2) = APP (Map f l1) (Map f l2)``),
          Induct_on `l1` THEN
          ASM_REWRITE_TAC[Map_def, APP_def]
);

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

(* Exporting and printing the theory            *)
val _ = export_theory();
val _ = print_theory "-";

end

(****************************************************************************)
(* Author: Kyle Peppe                                                       *)
(* Exercise 11.6.3                                                          *)
(* Date: 2/4/20                                                             *)
(****************************************************************************)


(* Beginning Commands                           *)
structure exTypeScript = struct
open HolKernel Parse boolLib bossLib;
open boolTheory TypeBase arithmeticTheory

(* Command to set the theory                    *)
val _ = new_theory "nexp";

val _ = Datatype `nexp = Num num | Add nexp nexp | Sub nexp nexp | Mult nexp nexp`;

val nexpValDef =
Define
        `(nexpVal (Num num) = num) /\ (nexpVal (Add f1 f2) = (nexpVal f1) + (nexpVal f2))
        /\ (nexpVal (Sub f1 f2) = (nexpVal f1) - (nexpVal f2))
        /\ (nexpVal (Mult f1 f2) = (nexpVal f1) * (nexpVal f2))`;

(* Add_0                                        *)
val Add_0 =
TAC_PROOF(
        ([], ``!(f:nexp).nexpVal (Add (Num 0) f) = nexpVal f``),
        Induct_on `f` THEN
        ASM_REWRITE_TAC[ADD_CLAUSES, nexpValDef]
```

```
);

(* Save the Theorem                    *)
val _ = save_thm("Add_0", Add_0);

(* Add_SYM                             *)
val Add_SYM =
TAC_PROOF(
        ([], ``!(f1:nexp)(f2:nexp).nexpVal (Add f1 f2) = nexpVal (Add f2 f1)``),
        Induct_on `f1` THEN
        PROVE_TAC[nexpValDef, ADD_SYM]
);

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

(* Sub_0                               *)
val Sub_0 =
TAC_PROOF(
        ([],``!(f:nexp).(nexpVal (Sub (Num 0) f) = 0) /\
        (nexpVal (Sub f (Num 0)) = nexpVal f)``),
        Induct_on`f` THEN
        PROVE_TAC[nexpValDef, SUB_0]
);

(* Save the Theorem                    *)
val _ = save_thm("Sub_0", Sub_0);

(* Mult_ASSOC                          *)
val Mult_ASSOC =
TAC_PROOF(
        ([],``!(f1:nexp)(f2:nexp)(f3:nexp).nexpVal (Mult f1 (Mult f2 f3)) =
        nexpVal (Mult (Mult f1 f2) f3)``),
        Induct_on`f1` THEN
        ASM_REWRITE_TAC[MULT_ASSOC, nexpValDef]
);

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

(*Exporting the Theorem                *)
val _ = export_theory();

val _ = print_theory "-";

end
```