Project 5

Bharath Karumudi

August 19, 2019

Abstract

This project is to demonstrate the capabilities of implementing 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. Execution Transcripts
4. 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 requirements for this project are statisfied specifically, and by using HOL proved the below theorems:

$\vdash \forall l_1\ l_2.\ \texttt{LENGTH (APP } l_1\ l_2\texttt{) = LENGTH } l_1 \texttt{ + LENGTH } l_2$

$\vdash \texttt{Map } f \texttt{ (APP } l_1\ l_2\texttt{) = APP (Map } f\ l_1\texttt{) (Map } f\ l_2\texttt{)}$

$\vdash \forall f.\ \texttt{nexpVal (Add (Num 0) } f\texttt{) = nexpVal } f$

$\vdash \forall f_1\ f_2.\ \texttt{nexpVal (Add } f_1\ f_2\texttt{) = nexpVal (Add } f_2\ f_1\texttt{)}$

$\vdash \forall f.$
$\quad \texttt{(nexpVal (Sub (Num 0) } f\texttt{) = 0)} \land$
$\quad \texttt{(nexpVal (Sub } f \texttt{ (Num 0)) = nexpVal } f\texttt{)}$

$\vdash \forall f_1\ f_2\ f_3.$
$\quad \texttt{nexpVal (Mult } f_1 \texttt{ (Mult } f_2\ f_3\texttt{)) =}$
$\quad \texttt{nexpVal (Mult (Mult } f_1\ f_2\texttt{) } f_3\texttt{)}$

Chapter 2

Exercise 11.6.1

2.1 Problem Statement

In this exercise we need to prove the theorem: $\vdash \forall l_1\ l_2.\ \texttt{LENGTH}\ (\texttt{APP}\ l_1\ l_2) = \texttt{LENGTH}\ l_1 + \texttt{LENGTH}\ l_2$

2.2 Relevant Code

```
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[ADD_CLAUSES, APP_def,LENGTH])

val _ = save_thm("LENGTH_APP", LENGTH_APP);
```

2.3 Execution Transcripts

```
-------------------------------------------------------------------- 1
> > # # # Definition has been stored under "APP_def"
val APP_def =
   |- (!(l :'a list). APP ([] :'a list) l = l) /\
   !(h :'a) (l1 :'a list) (l2 :'a list). APP (h::l1) l2 = h::APP l1 l2:
   thm
> # # # # # # # # val LENGTH_APP =
   |- !(l1 :'a list) (l2 :'a list).
     LENGTH (APP l1 l2) = LENGTH l1 + LENGTH l2:
   thm
```

2.3.1 Explanation of Results

The above results shows that the requirements are satisfied.

Chapter 3

Exercise 11.6.2

3.1 Problem Statement

In this exercise we need to prove the theorem: $\vdash$ `Map` f (`APP` l_1 l_2) $=$ `APP` (`Map` f l_1) (`Map` f l_2)

3.2 Relevant Code

```
val Map_def =
Define
`(Map f [] = []) /\ (Map f (x::f1) = f x::Map f (f1))`;

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

val _ = save_thm("Map_APP", Map_APP);
```

3.3 Execution Transcripts

```
---------------------------------------------------------------------                1
> > # # # # # # # <<HOL message: inventing new type variable names: 'a, 'b>>
val Map_APP =
    |- Map (f :'b -> 'a) (APP (l1 :'b list) (l2 :'b list)) =
    APP (Map f l1) (Map f l2):
    thm
```

3.3.1 Explanation of Results

The above results shows that the requirements are satisfied.

Chapter 4

Exercise 11.6.3

4.1 Problem Statement

In this exercise we need to prove the following:

$\vdash \forall f.$ `nexpVal (Add (Num 0)` f`) = nexpVal` f

$\vdash \forall f_1\ f_2.$ `nexpVal (Add` $f_1\ f_2$`) = nexpVal (Add` $f_2\ f_1$`)`

$\vdash \forall f.$
`(nexpVal (Sub (Num 0)` f`) = 0)` $\wedge$
`(nexpVal (Sub` f `(Num 0)) = nexpVal` f`)`

$\vdash \forall f_1\ f_2\ f_3.$
`nexpVal (Mult` f_1 `(Mult` $f_2\ f_3$`)) =`
`nexpVal (Mult (Mult` $f_1\ f_2$`)` f_3`)`

4.2 Relevant Code

```
(*************************************************************************)
(* Introduce the syntax of natural number expression nexp               *)
(*************************************************************************)
val _ = Datatype
`nexp = Num num | Add nexp nexp | Sub nexp nexp | Mult nexp nexp`;

(*************************************************************************)
(* Prove  nexpVal (Add (Num 0) f ) = nexpVal f                          *)
(*************************************************************************)

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

val _ = save_thm("Add_0",Add_0)


(*************************************************************************)
(* Prove nexpVal (Add f1 f2) = nexpVal (Add f2 f1)                      *)
(*************************************************************************)

val Add_SYM =
TAC_PROOF(([],``!f1 f2.nexpVal (Add f1 f2) = nexpVal (Add f2 f1)``),
```

```
Induct_on `f1` THEN
PROVE_TAC [nexpVal_def, ADD_SYM]);

val _ = save_thm("Add_SYM",Add_SYM)


(****************************************************************************)
(* Prove (nexpVal (Sub (Num 0) f ) = 0)
*)
(*        (nexpVal (Sub f (Num 0)) = nexpVal f )                           *)
(****************************************************************************)

val Sub_0 =
TAC_PROOF(([],
``!f.(nexpVal (Sub (Num 0) f ) = 0)
  (nexpVal (Sub f (Num 0)) = nexpVal f )``),
STRIP_TAC THEN
ASM_REWRITE_TAC[nexpVal_def] THEN
PROVE_TAC[nexpVal_def, SUB_0]);

val _ = save_thm("Sub_0",Sub_0)

(****************************************************************************)
(* Prove   f1 f2 f3.
*)
(*        nexpVal (Mult f1 (Mult f2 f3)) =                                 *)
(*        nexpVal (Mult (Mult f1 f2) f3)                                   *)
(****************************************************************************)

val Mult_ASSOC =
TAC_PROOF(([],``!f1 f2 f3.
    nexpVal (Mult f1 (Mult f2 f3)) =
    nexpVal (Mult (Mult f1 f2) f3)``),
REPEAT STRIP_TAC THEN
PROVE_TAC[nexpVal_def, MULT_ASSOC]);

val _ = save_thm("Mult_ASSOC",Mult_ASSOC)
```

4.3 Execution Transcripts

```
--------------------------------------------------------------------                    1
> # <<HOL message: Defined type: "nexp">>
> # val nexp_one_one =
   |- (!(a :num) (a' :num). (Num a = Num a') <=> (a = a')) /\
   (!(a0 :nexp) (a1 :nexp) (a0' :nexp) (a1' :nexp).
      (Add a0 a1 = Add a0' a1') <=> (a0 = a0') /\ (a1 = a1')) /\
   (!(a0 :nexp) (a1 :nexp) (a0' :nexp) (a1' :nexp).
      (Sub a0 a1 = Sub a0' a1') <=> (a0 = a0') /\ (a1 = a1')) /\
   !(a0 :nexp) (a1 :nexp) (a0' :nexp) (a1' :nexp).
     (Mult a0 a1 = Mult a0' a1') <=> (a0 = a0') /\ (a1 = a1'):
   thm
> # val nexp_distinct_clauses =
   |- (!(a1 :nexp) (a0 :nexp) (a :num). Num a <> Add a0 a1) /\
   (!(a1 :nexp) (a0 :nexp) (a :num). Num a <> Sub a0 a1) /\
   (!(a1 :nexp) (a0 :nexp) (a :num). Num a <> Mult a0 a1) /\
   (!(a1' :nexp) (a1 :nexp) (a0' :nexp) (a0 :nexp).
      Add a0 a1 <> Sub a0' a1') /\
   (!(a1' :nexp) (a1 :nexp) (a0' :nexp) (a0 :nexp).
      Add a0 a1 <> Mult a0' a1') /\
   !(a1' :nexp) (a1 :nexp) (a0' :nexp) (a0 :nexp).
     Sub a0 a1 <> Mult a0' a1':
   thm
> # # # # # # Definition has been stored under "nexpVal_def"
val nexpVal_def =
   |- (!(f :num). nexpVal (Num f) = f) /\
   (!(f1 :nexp) (f2 :nexp).
      nexpVal (Add f1 f2) = nexpVal f1 + nexpVal f2) /\
   (!(f1 :nexp) (f2 :nexp).
      nexpVal (Sub f1 f2) = nexpVal f1 - nexpVal f2) /\
   !(f1 :nexp) (f2 :nexp).
     nexpVal (Mult f1 f2) = nexpVal f1 * nexpVal f2:
   thm
> # # # # val Add_0 =
   |- !(f :nexp). nexpVal (Add (Num (0 :num)) f) = nexpVal f:
   thm
> > # > # # # 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
> > > # # # # # # 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
> > > # # # # # Meson search level: ......................
val Mult_ASSOC =
   |- !(f1 :nexp) (f2 :nexp) (f3 :nexp).
     nexpVal (Mult f1 (Mult f2 f3)) = nexpVal (Mult (Mult f1 f2) f3):
   thm
> > >
```

4.3.1 Explanation of Results

The above results shows that the requirements are satisfied.

Chapter 5

Appendix A: Exercise 11.6.1 and 11.6.2

The following code is from the file exTypeScript.sml

```
(****************************************************************************)
(*  Exercise: Chapter 11                                                    *)
(*  Author: Bharath Karumudi                                                *)
(*  Date: Aug 10, 2019                                                      *)
(****************************************************************************)

structure exTypeScript = struct
open HolKernel Parse boolLib bossLib;
open listTheory TypeBase arithmeticTheory

(* ==== interactive mode ====
map load ["boolTheory","TypeBase","bexpTheory"];
open boolTheory TypeBase bexpTheory
 ==== end interactive mode ==== *)

val _ = new_theory "exType";

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


(****************************************************************************)
(*  Exercise: 11.6.1                                                        *)
(*        l 1 l 2 . LENGTH (APP l 1 l 2 ) = LENGTH l 1 + LENGTH l 2
*)
(****************************************************************************)

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[ADD_CLAUSES, APP_def,LENGTH])

val _ = save_thm("LENGTH_APP", LENGTH_APP);


(****************************************************************************)
```

```
(* Exercise: 11.6.2
*)
(* ` Map f (APP l 1 l 2 ) = APP (Map f l 1 ) (Map f l 2 )                    *)
(*************************************************************************)

val Map_def =
Define
`(Map f [] = []) /\ (Map f (x::f1) = f x::Map f (f1))`;

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

val _ = save_thm("Map_APP", Map_APP);

(*************************************************************************)
(* Exporting Theory                                                      *)
(*************************************************************************)

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

end (* Structure *)
```

Chapter 6

Appendix B: Exercise 11.6.3

The following code is from the file nexpScript.sml

```
(***********************************************************************)
(*   Exercise: Chapter 11.6.3                                          *)
(*   Author: Bharath Karumudi                                          *)
(*   Derived from: bexpScript.sml                                      *)
(*   Date: Aug 10, 2019                                                *)
(***********************************************************************)

structure nexpScript = struct
open HolKernel Parse boolLib bossLib;
open TypeBase boolTheory arithmeticTheory

(* ==== interactive mode ====
map load ["boolTheory","TypeBase","nexpTheory"];
open boolTheory TypeBase nexpTheory
 ==== end interactive mode ==== *)

val _ = new_theory "nexp";

(***********************************************************************)
(* Introduce the syntax of natural number expression nexp             *)
(***********************************************************************)
val _ = Datatype
`nexp = Num num | Add nexp nexp | Sub nexp nexp | Mult nexp nexp`;

(***********************************************************************)
(* Prove that identical nexps have identical components               *)
(***********************************************************************)
val nexp_one_one = one_one_of ``:nexp``
val _ = save_thm("nexp_one_one",nexp_one_one)

(***********************************************************************)
(* Prove that the different forms of bexp expressions are distinct    *)
(***********************************************************************)
val nexp_distinct_clauses = distinct_of ``:nexp``
val _ = save_thm("nexp_distinct_clauses",nexp_distinct_clauses)

(***********************************************************************)
(* Define the semantics of nexp expressions                           *)
(***********************************************************************)
val nexpVal_def =
Define
`(nexpVal (Num f) = f )/\
```

```
(nexpVal (Add f1 f2) = (nexpVal f1) + (nexpVal f2)) /\
(nexpVal (Sub f1 f2) = (nexpVal f1) - (nexpVal f2)) /\
(nexpVal (Mult f1 f2) = (nexpVal f1) * (nexpVal f2))`


(*********************************************************************)
(* Prove  nexpVal (Add (Num 0) f ) = nexpVal f                      *)
(*********************************************************************)

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

val _ = save_thm("Add_0",Add_0)


(*********************************************************************)
(* Prove nexpVal (Add f1 f2) = nexpVal (Add f2 f1)                   *)
(*********************************************************************)

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

val _ = save_thm("Add_SYM",Add_SYM)


(*********************************************************************)
(* Prove (nexpVal (Sub (Num 0) f ) = 0)
*)
(*       (nexpVal (Sub f (Num 0)) = nexpVal f )                      *)
(*********************************************************************)

val Sub_0 =
TAC_PROOF(([],
``!f.(nexpVal (Sub (Num 0) f ) = 0)
  (nexpVal (Sub f (Num 0)) = nexpVal f )``),
STRIP_TAC THEN
ASM_REWRITE_TAC[nexpVal_def] THEN
PROVE_TAC[nexpVal_def, SUB_0]);

val _ = save_thm("Sub_0",Sub_0)

(*********************************************************************)
(* Prove   f1 f2 f3.
*)
(*       nexpVal (Mult f1 (Mult f2 f3)) =                            *)
(*       nexpVal (Mult (Mult f1 f2) f3)                              *)
(*********************************************************************)
```

```
val Mult_ASSOC =
TAC_PROOF(([],``!f1 f2 f3.
   nexpVal (Mult f1 (Mult f2 f3)) =
   nexpVal (Mult (Mult f1 f2) f3)``),
REPEAT STRIP_TAC THEN
PROVE_TAC[nexpVal_def, MULT_ASSOC]);

val _ = save_thm("Mult_ASSOC",Mult_ASSOC)


(***********************************************************************)
(* Exporting Theory                                                    *)
(***********************************************************************)

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

end (* Structure *)
```