Project 5

Alfred Murabito

February 15, 2020

Abstract

This report details results for the following exercises from *Certified Security by Design Using Higher Order Logic*: 11.6.1, 11.6.2, 11.6.3 The exercises from Ch 11 focus on defining new types in HOL and defining the properties and semantics of them. We then used our new type definitions to develop new theories that can be imported in further HOL proofs. All of these theories are built using the Holmake program. Each of the exercises includes a problem statement, relevant code, test cases as appropriate and test results.

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 11 is included in the following chapters.

A HOLReports subfolder is in the HOL directory with all the HOL generated Latex macros used in the report, the HOL directory contains the exTypeScript.sml and nexpScript.sml files used by Holmake to generate the theorem object files, and the LaTex directory contains the project report LaTex document.

The theories are shown pretty printed in the Ch. 11 Theory section, and the source code has been included in the Appendix.

2 Chapter 11 Exercises

2.1 Problem Statement

Exercise 11.6.1 Theorem was answered using structural indcution on the inital goal then rewriting using the APP function definition, the ADD_CLAUSES theorem from arithmeticTheory, and the LENGTH theorem.

Exercise 11.6.2 Theorem used our APP definition along with our Map definition to prove the Map_APP theorem. Exercise 11.6.3 proofs use our new datatype nexp along with its semantic value definition as well as a few theories from arithmeticTheory.

2.2 Test Results

Exercise 11.6.1:

```
> # # # Definition has been stored under "APP_def"
val APP_def =
   |- (l. APP [] l = l)  h l1 l2. APP (h::l1) l2 = h::APP l1 l2:
   thm

> val LENGTH_APP =
TAC_PROOF (
([], ``!(l1:'a list)(l2:'a list). LENGTH (APP l1 l2) = LENGTH l1 + LENGTH l2``),
Induct_on `l1` THEN
STRIP_TAC THEN
ASM_REWRITE_TAC [APP_def, ADD_CLAUSES, LENGTH]);
# # # # # <<HOL message: more than one resolution of overloading was possible>>
val LENGTH_APP =
   |- l1 l2. LENGTH (APP l1 l2) = LENGTH l1 + LENGTH l2:
   thm
>
```

Exercise 11.6.2:

```
> # # # Definition has been stored under "APP_def"
val APP_def =
   |- (l. APP [] l = l)  h l1 l2. APP (h::l1) l2 = h::APP l1 l2:
   thm

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

Exercise 11.6.3:

```
> val _ = Datatype
`nexp = Num num | Add nexp nexp | Sub nexp nexp | Mult nexp nexp`;
# <<HOL message: Defined type: "nexp">>
> val nexpVal_def =
Define
`(nexpVal (Num num) = num) /\
 (nexpVal (Add n1 n2) = (nexpVal n1) + (nexpVal n2)) /\
 (nexpVal (Sub n1 n2) = (nexpVal n1) - (nexpVal n2)) /\
 (nexpVal (Mult n1 n2) = (nexpVal n1) * (nexpVal n2))`;
# # # # # <<HOL message: more than one resolution of overloading was possible>>
Definition has been stored under "nexpVal_def"
val nexpVal_def =
   |- (num. nexpVal (Num num) = num)
   (n1 n2. nexpVal (Add n1 n2) = nexpVal n1 + nexpVal n2)
   (n1 n2. nexpVal (Sub n1 n2) = nexpVal n1  nexpVal n2)
   n1 n2. nexpVal (Mult n1 n2) = nexpVal n1 * nexpVal n2:
   thm
>
> val Add_0 =
TAC_PROOF (
([],``!(n:nexp).nexpVal(Add (Num 0) n) = nexpVal(n)``),
Induct_on`n` THEN
ASM_REWRITE_TAC[nexpVal_def, ADD]
);
# # # # # val Add_0 =
   |- n. nexpVal (Add (Num 0) n) = nexpVal n:
   thm
> val Add_SYM =
TAC_PROOF (
([],``!(n1:nexp)(n2:nexp).nexpVal(Add n1 n2) = nexpVal(Add n2 n1)``),
PROVE_TAC [nexpVal_def, ADD_SYM]
);
# # # # Meson search level: ..........
val Add_SYM =
   |- n1 n2. nexpVal (Add n1 n2) = nexpVal (Add n2 n1):
   thm
> val Sub_0 =
```

```
TAC_PROOF (
([],``!(n:nexp).(nexpVal(Sub (Num 0) n) = 0) /\
                (nexpVal(Sub n (Num 0)) = nexpVal n)``),
ASM_REWRITE_TAC[nexpVal_def, SUB_0]
);
# # # # # val Sub_0 =
   |- n.
     (nexpVal (Sub (Num 0) n) = 0)
     (nexpVal (Sub n (Num 0)) = nexpVal n):
   thm

> val Mult_ASSOC =
TAC_PROOF (
([],``!(n1:nexp)(n2:nexp)(n3:nexp).nexpVal(Mult n1 (Mult n2 n3)) =
                                          nexpVal(Mult (Mult n1 n2) n3)``),
ASM_REWRITE_TAC [nexpVal_def, MULT_ASSOC]
);
# # # # # val Mult_ASSOC =
   |- n1 n2 n3.
     nexpVal (Mult n1 (Mult n2 n3)) = nexpVal (Mult (Mult n1 n2) n3):
   thm
>
```

2.3 Exercise 11 Theory

Built: 15 February 2020
Parent Theories: indexedLists, patternMatches

[APP_ASSOC]

$\vdash \forall l_1\ l_2\ l_3.$ `APP (APP` $l_1\ l_2$`)` l_3 `= APP` l_1 `(APP` $l_2\ l_3$`)`

[LENGTH_APP]

$\vdash \forall l_1\ l_2.$ `LENGTH (APP` $l_1\ l_2$`) = LENGTH` l_1 `+ LENGTH` l_2

[Map_APP]

$\vdash$ `Map` f `(APP` $l_1\ l_2$`) = APP (Map` $f\ l_1$`) (Map` $f\ l_2$`)`

[Add_0]

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

[Add_SYM]

$\vdash \forall n_1\ n_2.$ `nexpVal (Add` $n_1\ n_2$`) = nexpVal (Add` $n_2\ n_1$`)`

[Mult_ASSOC]

$\vdash \forall n_1\ n_2\ n_3.$
    `nexpVal (Mult` n_1 `(Mult` $n_2\ n_3$`)) =`
    `nexpVal (Mult (Mult` $n_1\ n_2$`)` n_3`)`

[Sub_0]

$\vdash \forall n.$
    `(nexpVal (Sub (Num 0)` n`) = 0)` $\land$
    `(nexpVal (Sub` n `(Num 0)) = nexpVal` n`)`

A Source Code for exTypeTheory

The following code is from *HOL/exTypeScript.sml*

```
structure exTypeScript = struct

(* === Interactive mode: within a comment so it isn't executed by Holmake ===
map load ["listTheory","TypeBase","exTypeTheory"];
open listTheory TypeBase exTypeTheory arithmeticTheory
=== end Interactive mode ==========================================*)

open HolKernel boolLib Parse bossLib
open listTheory TypeBase arithmeticTheory

val _ = new_theory "exType";

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

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

(* === start here ===
set_goal([] ,
``!(l1:'a list)(l2:'a list)(l3:'a list).
  (APP(APP l1 l2) l3) = (APP l1 (APP l2 l3))``)
Induct_on `l1` THEN
ASM_REWRITE_TAC [APP_def]
=== end here === *)

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
ASM_REWRITE_TAC [APP_def])

val _ = save_thm("APP_ASSOC", APP_ASSOC)

(* Exercise 11_6_1 LENGTH_APP thm *)

(* === start here ===
```

```
set_goal(
([], ``!(l1:'a list)(l2:'a list). LENGTH (APP l1 l2) = LENGTH l1 + LENGTH l2``))
Induct_on `l1` THEN
STRIP_TAC THEN
ASM_REWRITE_TAC [APP_def, ADD_CLAUSES, LENGTH]
=== end here === *)

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

val _ = save_thm("LENGTH_APP", LENGTH_APP)

(* Exercise 11_6_2 Map_APP thm *)

(* === start here ===
val Map_def =
Define
 `(Map f [] = []) /\
 (Map f (h::(l1:'a list)) = (f h)::(Map f l1))`
 === end here === *)

val Map_def =
Define
 `(Map f [] = []) /\
 (Map f (h::(l1:'a list)) = (f h)::(Map f l1))`

(* === start here ===
 set_goal(
 ([], ``Map f (APP l1 l2) = APP (Map f l1) (Map f l2)``))
 Induct_on `l1` THEN
 ASM_REWRITE_TAC [App_def, Map_def]
 === end here === *)

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

val _ = save_thm("Map_APP", Map_APP)


val _ = export_theory ();
```

```
val _ = print_theory "-";

end
```

B Source Code for nexpTheory

The following code is from *HOL/nexpScript.sml*

```
(**************************************************************************)
(* Syntax and semantics of natural expressions
*)
(* Author: Alfred Murabito
*)
(* Date: 14 February 2020
*)
(**************************************************************************)
structure nexpScript = struct
open HolKernel Parse boolLib bossLib;
open TypeBase boolTheory arithmeticTheory

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

val _ = new_theory "nexp";

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

val nexpVal_def =
Define
`(nexpVal (Num num) = num) /\
 (nexpVal (Add n1 n2) = (nexpVal n1) + (nexpVal n2)) /\
 (nexpVal (Sub n1 n2) = (nexpVal n1) - (nexpVal n2)) /\
 (nexpVal (Mult n1 n2) = (nexpVal n1) * (nexpVal n2))`

(**************************************************************************)
(* Prove nexpVal (Add (Num 0) n) = nexpVal(n)
*)
(**************************************************************************)
(* ==== start here ====
set_goal([],``!(n:nexp).nexpVal(Add (Num 0) n) = nexpVal(n)``)
ASM_REWRITE_TAC[nexpVal_def, ADD]
 ==== end here ==== *)

val Add_0 =
```

```
TAC_PROOF (
([],``!(n:nexp).nexpVal(Add (Num 0) n) = nexpVal(n)``),
Induct_on`n` THEN
ASM_REWRITE_TAC[nexpVal_def, ADD]
)

val _ = save_thm("Add_0", Add_0)

(*****************************************************************************)
(* Prove nexpVal (Add n1 n2) = nexpVal (Add n2 n1)
*)
(*****************************************************************************)
(* ==== start here ====
set_goal([],``!(n1:nexp)(n2:nexp).nexpVal(Add n1 n2) = nexpVal(Add n2 n1)``)
PROVE_TAC [nexpVal_def, ADD_SYM]
 ==== end here ==== *)

val Add_SYM =
TAC_PROOF (
([],``!(n1:nexp)(n2:nexp).nexpVal(Add n1 n2) = nexpVal(Add n2 n1)``),
PROVE_TAC [nexpVal_def, ADD_SYM]
)

val _ = save_thm("Add_SYM", Add_SYM)



(*****************************************************************************)
(*   Prove (nexpVal (Sub (Num 0) n) = 0) /\
*)
(*            (nexpVal (Sub n (Num 0)) = nexpVal n)
*)
(*****************************************************************************)

(* ==== start here ====
set_goal([],``!(n:nexp).(nexpVal(Sub (Num 0) n) = 0) /\
                        (nexpVal(Sub n (Num 0)) = nexpVal n)``)
ASM_REWRITE_TAC[nexpVal_def, SUB_0]
 ==== end here ==== *)

val Sub_0 =
TAC_PROOF (
([],``!(n:nexp).(nexpVal(Sub (Num 0) n) = 0) /\
                (nexpVal(Sub n (Num 0)) = nexpVal n)``),
ASM_REWRITE_TAC[nexpVal_def, SUB_0]
)
```

```
val _ = save_thm("Sub_0", Sub_0)

(*****************************************************************************)
(*  Prove nexpVal (Mult n1 (Mult n2 n3)) =
*)
(*        nexpVal (Mult (Mult n1 n2) n3)
*)
(*****************************************************************************)

(* ==== start here ====
set_goal([], ``!(n1:nexp)(n2:nexp)(n3:nexp).nexpVal(Mult n1 (Mult n2 n3)) =
                                    nexpVal(Mult (Mult n1 n2) n3)``)
ASM_REWRITE_TAC [nexpVal_def, MULT_ASSOC]
 ==== end here ==== *)

val Mult_ASSOC =
TAC_PROOF (
([], ``!(n1:nexp)(n2:nexp)(n3:nexp).nexpVal(Mult n1 (Mult n2 n3)) =
                                    nexpVal(Mult (Mult n1 n2) n3)``),
ASM_REWRITE_TAC [nexpVal_def, MULT_ASSOC]
)

val _ = save_thm("Mult_ASSOC", Mult_ASSOC)

val _ = export_theory();

end
```