Project 7

Jinhao Wei

5 March 2019

Abstract

This report is basically a summary on my attempts on Project 7, which includes proof by induction, function definition and datatype definiton. This report provides my solution on *exercise 11.6.1*, *11.6.2* and *11.6.3*. In addition, I had fine printed the corresponding datatypes and proofs and put the reports in *../HOL/HOL-Reports/exTypeReport.pdf* and *../HOL/HOLReports/nexpReport.pdf*.

Acknowledgments: This project follows the format and structure of *sampleTheory* provided by Professor Shiu-Kai Chin. To make it more accurate, this project mostly followed the format of one of my previous projects, which is project 5, and project 5 followed the sturcture of Professor Shiu-Kai Chin's *sampleTheory* project.

Contents

Chapter 1

Executive Summary

All requirements for this project are satisfied. In particular, we defined all the datatypes and proved all the theorems in this project, pretty printed the HOL theories, and made use of the *EmitTeX* structure to typeset HOL theorems in this report.

We gave definitions for the following functions or datatypes

[APP_def]

$\vdash (\forall l.\ \texttt{APP}\ [\,]\ l = l) \land \forall h\ l_1\ l_2.\ \texttt{APP}\ (h::l_1)\ l_2 = h::\texttt{APP}\ l_1\ l_2$

[Map_def]

$\vdash (\forall f.\ \texttt{Map}\ f\ [\,] = [\,]) \land \forall f\ h\ l.\ \texttt{Map}\ f\ (h::l) = f\ h::\texttt{Map}\ f\ l$

$nexp$ = Num num | Add nexp nexp | Sub nexp nexp | Mult nexp nexp

[nexpVal_def]

$$\begin{aligned}
\vdash\ & (\forall num.\ \texttt{nexpVal}\ (\texttt{Num}\ num) = num) \land \\
& (\forall f_1\ f_2.\ \texttt{nexpVal}\ (\texttt{Add}\ f_1\ f_2) = \texttt{nexpVal}\ f_1 + \texttt{nexpVal}\ f_2) \land \\
& (\forall f_1\ f_2.\ \texttt{nexpVal}\ (\texttt{Sub}\ f_1\ f_2) = \texttt{nexpVal}\ f_1 - \texttt{nexpVal}\ f_2) \land \\
& \forall f_1\ f_2.\ \texttt{nexpVal}\ (\texttt{Mult}\ f_1\ f_2) = \texttt{nexpVal}\ f_1 \times \texttt{nexpVal}\ f_2
\end{aligned}$$

and the following theorems are proved

[LENGTH_APP]

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

[Map_APP]

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

[Add_0]

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

[Add_SYM]

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

[Mult_ASSOC]

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

[Sub_0]

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

Reproducibility in ML and LaTeX

All ML and LaTeX source files compile well on the environment provided by this course.

Chapter 2

Exercise 11.6.1

2.1 Problem Statement

In this exercise, we will define a function named *APP*, according to the following formula

$\vdash$ `(`$\forall l$`. APP [] `l` = `l`)` $\land$ $\forall h\ l_1\ l_2$`. APP (`h`::`l_1`) `l_2` = `h`::APP `l_1` `l_2

and prove the theorem

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

Before we go through the following sections, we will need to print

```
open HolKernel Parse boolLib bossLib;
open arithmeticTheory listTheory;
```

in HOL session.

2.2 Definition of *APP*

2.2.1 Code for defining *APP*

We used the following code to define *APP*

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

2.2.2 Session Transcript

If we send the above code to HOL, we will see the transcript as below:

```
> # # # Definition has been stored under "APP_def"                1
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
```

2.3 Proof for *LENGTH_APP*

2.3.1 Code for Proving *LENGTH_APP*

```
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] THEN
ASM_REWRITE_TAC [APP_def, LENGTH, ADD_CLAUSES]
)
)
```

2.3.2 Session Transcript

The above code will give us transcript as below:

```
> # # # # # # # val LENGTH_APP =
   |- !(l1 :'a list) (l2 :'a list).
     LENGTH (APP l1 l2) = LENGTH l1 + LENGTH l2:
   thm
```
2

Chapter 3

Exercise 11.6.2

3.1 Problem Statement

In this exercise, we defined a function *Map*, using the following formula

$\vdash$ `(∀f. Map f [] = []) ∧ ∀f h l. Map f (h::l) = f h::Map f l`

and proved the theorem *Map_APP*:

$\vdash$ `Map f (APP l1 l2) = APP (Map f l1) (Map f l2)`

Before we go through the following sections, we will need to print

```
open HolKernel Parse boolLib bossLib;
open arithmeticTheory listTheory;
```

in HOL session.

3.2 Definition of *Map*

3.2.1 Code for Defining *Map*

We use the following code to define *Map*

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

3.2.2 Session Transcript

The above code will give us transcript as below:

```
> # # # <<HOL message: inventing new type variable names: 'b>>
Definition has been stored under "Map_def"
val Map_def =
   |- (!(f :'a -> 'b). Map f ([] :'a list) = ([] :'b list)) /\
   !(f :'a -> 'b) (h :'a) (l :'a list). Map f (h::l) = f h::Map f l:
   thm
```
1

3.3 Proof of *Map_APP*

3.3.1 Code for Proving *Map_APP*

We will use the following code to prove *Map_APP*.

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

3.3.2 Session Transcript

The above code will give us transcript as below:

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

2

Chapter 4

Exercise 11.6.3

4.1 Problem Statement

In this exercise, we will define our datatype *nexp*:

```
nexp = Num num | Add nexp nexp | Sub nexp nexp | Mult nexp nexp
```

and its semantic *nexpVal*

```
[nexpVal_def]
 ⊢ (∀num. nexpVal (Num num) = num) ∧
   (∀f1 f2. nexpVal (Add f1 f2) = nexpVal f1 + nexpVal f2) ∧
   (∀f1 f2. nexpVal (Sub f1 f2) = nexpVal f1 - nexpVal f2) ∧
   ∀f1 f2. nexpVal (Mult f1 f2) = nexpVal f1 × nexpVal f2
```

then we will prove several theorems concerning the datatype, including

```
[Add_0]
 ⊢ ∀f. nexpVal (Add (Num 0) f) = nexpVal f

[Add_SYM]
 ⊢ ∀f1 f2. nexpVal (Add f1 f2) = nexpVal (Add f2 f1)

[Mult_ASSOC]
 ⊢ ∀f1 f2 f3.
     nexpVal (Mult f1 (Mult f2 f3)) =
     nexpVal (Mult (Mult f1 f2) f3)

[Sub_0]
 ⊢ ∀f.
     (nexpVal (Sub (Num 0) f) = 0) ∧
     (nexpVal (Sub f (Num 0)) = nexpVal f)
```

Before we go through the following sections, we will need to enter the code below in HOL window.

```
open HolKernel Parse boolLib bossLib;
open TypeBase boolTheory arithmeticTheory
```

4.2 Definition of *nexp*

4.2.1 Code for Defining *nexp*

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

4.2.2 Session Transcript

```
> # <<HOL message: Defined type: "nexp">>
```

4.3 Definition of *nexpVal*

4.3.1 Code for Defining *nexpVal*

```
val nexpVal_def =
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))
`
```

4.3.2 Session Transcript

```
> # # # # # # # Definition has been stored under "nexpVal_def"
val nexpVal_def =
   |- (!(num :num). nexpVal (Num num) = num) /\
   (!(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
```

4.4 Proof of *Add_0*

4.4.1 Code for Proving *Add_0*

```
val Add_0 =
TAC_PROOF(
([], ``!f.nexpVal (Add (Num 0) f) = nexpVal f``),
Induct_on `f` THEN
ASM_REWRITE_TAC [ADD] THEN
REWRITE_TAC [nexpVal_def] THEN
REWRITE_TAC [ADD] THEN
REPEAT (PROVE_TAC [ADD, SUB, MULT, nexpVal_def])
);
```

4.4.2 Session Transcript

```
> # # # # # # # # val Add_0 =
   |- !(f :nexp). nexpVal (Add (Num (0 :num)) f) = nexpVal f:
   thm
```

4.5 Proof of *Add_SYM*

4.5.1 Code for Proving *Add_SYM*

```
val Add_SYM =
TAC_PROOF(
([], ``!f1 f2. nexpVal (Add f1 f2) = nexpVal (Add f2 f1)``),
REWRITE_TAC [ADD, nexpVal_def] THEN
REWRITE_TAC [Once ADD_COMM]
);
```

4.5.2 Session Transcript

```
> # # # # # val Add_SYM =
   |- !(f1 :nexp) (f2 :nexp). nexpVal (Add f1 f2) = nexpVal (Add f2 f1):
   thm
```

4.6 Proof of *Sub_0*

4.6.1 Code for Proving *Sub_0*

```
val Sub_0 =
TAC_PROOF(
([], ``!f. (nexpVal (Sub (Num 0) f ) = 0)/\(nexpVal (Sub f (Num 0)) = nexpVal
    f)``),
REWRITE_TAC [SUB, nexpVal_def] THEN
REWRITE_TAC [SUB_0]
);
```

4.6.2 Session Transcript

```
> # # # # # val Sub_0 =
   |- !(f :nexp).
     (nexpVal (Sub (Num (0 :num)) f) = (0 :num)) /\
     (nexpVal (Sub f (Num (0 :num))) = nexpVal f):
   thm
```

4.7 Proof of *Mult_ASSOC*

4.7.1 Code for Proving *Mult_ASSOC*

```
val Mult_ASSOC =
TAC_PROOF(
([], ``!f1 f2 f3. nexpVal (Mult f1 (Mult f2 f3)) = nexpVal (Mult (Mult f1 f2)
    f3)``),
REWRITE_TAC [nexpVal_def, MULT, MULT_ASSOC]
);
```

4.7.2 Session Transcript

```
> # # # # 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 for exTypeScript.sml

The following code is from *exTypeScript.sml*, which is located in directory "../HOL/"

```
structure exTypeScript = struct

open HolKernel Parse boolLib bossLib;
open arithmeticTheory listTheory;


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))`;

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] THEN
ASM_REWRITE_TAC [APP_def, LENGTH, ADD_CLAUSES]
)
)

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


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

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

val _ = export_theory();
end
```

Appendix B

Source Code for nexpScript.sml

The following code is from *nexpScript.sml*, which is located in directory "../HOL/"

```
structure nexpScript = struct

open HolKernel Parse boolLib bossLib;
open TypeBase boolTheory arithmeticTheory

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

val Add_0 =
TAC_PROOF(
([], ``!f.nexpVal (Add (Num 0) f) = nexpVal f``),
Induct_on `f` THEN
ASM_REWRITE_TAC [ADD] THEN
REWRITE_TAC [nexpVal_def] THEN
REWRITE_TAC [ADD] THEN
REPEAT (PROVE_TAC [ADD, SUB, MULT, nexpVal_def])
);

val Add_SYM =
TAC_PROOF(
([], ``!f1 f2. nexpVal (Add f1 f2) = nexpVal (Add f2 f1)``),
REWRITE_TAC [ADD, nexpVal_def] THEN
REWRITE_TAC [Once ADD_COMM]
);

val Sub_0 =
TAC_PROOF(
([], ``!f. (nexpVal (Sub (Num 0) f ) = 0)/\(nexpVal (Sub f (Num 0)) = nexpVal
   f)``),
REWRITE_TAC [SUB, nexpVal_def] THEN
```

```
REWRITE_TAC [SUB_0]
);

val Mult_ASSOC =
TAC_PROOF(
([], ``!f1 f2 f3. nexpVal (Mult f1 (Mult f2 f3)) = nexpVal (Mult (Mult f1 f2)
    f3)``),
REWRITE_TAC [nexpVal_def, MULT, MULT_ASSOC]
);

val _ = save_thm("Add_0", Add_0);
val _ = save_thm("Add_SYM", Add_SYM);
val _ = save_thm("Sub_0", Sub_0);
val _ = save_thm("Mult_ASSOC", Mult_ASSOC);

val _ = export_theory();

end
```