Project 10

Jinhao Wei

2 April 2019

Abstract

This report is basically a summary on my attempts on Project 10, which includes definitions propoties and theorems on several types of state machines. This report provides my solutions on *16.3.1* and *16.3.2*. In addition, I had pretty-printed the corresponding datatypes and proofs and put the reports in the corresponding folders */HOL/M1/HOLReports/* and */HOL/Counter/HOLReports/*

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. Besides, this project relies on */HOL/sm/smScript.sml* and */HOL/sminfRules.sml*, which are also provided by Professor Shiu-Kai Chin.

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.

For question *16.3.1*, we introduced the following datatypes

```
command = i0 | i1

output = o0 | o1

state = S0 | S1 | S2
```

and gave proofs on the following theorems or propoties:

[command_distinct_clauses]

```
⊢ i0 ≠ i1
```

[m1_rules]

```
⊢ (∀ins outs.
     TR i0 (CFG (i0::ins) S0 outs) (CFG ins S1 (o0::outs))) ∧
  (∀ins outs.
     TR i1 (CFG (i1::ins) S0 outs) (CFG ins S2 (o1::outs))) ∧
  (∀ins outs.
     TR i0 (CFG (i0::ins) S1 outs) (CFG ins S0 (o0::outs))) ∧
  (∀ins outs.
     TR i1 (CFG (i1::ins) S1 outs) (CFG ins S0 (o0::outs))) ∧
  (∀ins outs.
     TR i0 (CFG (i0::ins) S2 outs) (CFG ins S2 (o1::outs))) ∧
  ∀ins outs.
    TR i1 (CFG (i1::ins) S2 outs) (CFG ins S2 (o1::outs))
```

[M1ns_def]

```
⊢ (M1ns S0 i0 = S1) ∧ (M1ns S0 i1 = S2) ∧ (M1ns S1 i0 = S0) ∧
  (M1ns S1 i1 = S0) ∧ (M1ns S2 i0 = S2) ∧ (M1ns S2 i1 = S2)
```

[M1ns_ind]

```
⊢ ∀P.
    P S0 i0 ∧ P S0 i1 ∧ P S1 i0 ∧ P S1 i1 ∧ P S2 i0 ∧ P S2 i1 ⇒
    ∀v v₁. P v v₁
```

[M1out_def]

```
⊢ (M1out S0 i0 = o0) ∧ (M1out S0 i1 = o1) ∧
  (M1out S1 i0 = o0) ∧ (M1out S1 i1 = o0) ∧
  (M1out S2 i0 = o1) ∧ (M1out S2 i1 = o1)
```

[M1out_ind]

```
⊢ ∀P.
    P S0 i0 ∧ P S0 i1 ∧ P S1 i0 ∧ P S1 i1 ∧ P S2 i0 ∧ P S2 i1 ⇒
    ∀v v₁. P v v₁
```

[m1TR_clauses]

```
⊢ (∀x x1s s₁ out1s x2s out2s s₂.
      TR x (CFG x1s s₁ out1s) (CFG x2s s₂ out2s) ⟺
      ∃NS Out ins.
        (x1s = x::ins) ∧ (x2s = ins) ∧ (s₂ = NS s₁ x) ∧
        (out2s = Out s₁ x::out1s)) ∧
    ∀x x1s s₁ out1s x2s out2s.
      TR x (CFG x1s s₁ out1s)
        (CFG x2s (M1ns s₁ x) (M1out s₁ x::out2s)) ⟺
      ∃ins. (x1s = x::ins) ∧ (x2s = ins) ∧ (out2s = out1s)
```

[m1TR_rules]

```
⊢ ∀s x ins outs.
      TR x (CFG (x::ins) s outs)
        (CFG ins (M1ns s x) (M1out s x::outs))
```

[m1Trans_Equiv_TR]

```
⊢ TR x (CFG (x::ins) s outs)
      (CFG ins (M1ns s x) (M1out s x::outs)) ⟺
    Trans x s (M1ns s x)
```

[output_distinct_clauses]

```
⊢ o0 ≠ o1
```

[state_distinct_clauses]

```
⊢ S0 ≠ S1 ∧ S0 ≠ S2 ∧ S1 ≠ S2
```

For question *16.3.2*, we introduced the following datatypes

```
ctrcmd = load num | count | hold

ctrOut = DISPLAY num

ctrState = COUNT num
```

and proved the following theorems or propoties:

[ctr_rules]

```
⊢ (∀ins outs.
      TR (load new) (CFG (load new::ins) (COUNT n) outs)
        (CFG ins (COUNT new) (DISPLAY new::outs))) ∧
    (∀ins outs.
      TR count (CFG (count::ins) (COUNT n) outs)
        (CFG ins (COUNT (n - 1)) (DISPLAY (n - 1)::outs))) ∧
    ∀ins outs.
      TR hold (CFG (hold::ins) (COUNT n) outs)
        (CFG ins (COUNT n) (DISPLAY n::outs))
```

[ctrcmd_distinct_clauses]

```
⊢ (∀a. load a ≠ count) ∧ (∀a. load a ≠ hold) ∧ count ≠ hold
```

[ctrNS_def]

```
⊢ (ctrNS (COUNT n) (load k) = COUNT k) ∧
    (ctrNS (COUNT n) count = COUNT (n - 1)) ∧
    (ctrNS (COUNT n) hold = COUNT n)
```

[ctrNS_ind]

```
⊢ ∀P.
     (∀n k. P (COUNT n) (load k)) ∧ (∀n. P (COUNT n) count) ∧
     (∀n. P (COUNT n) hold) ⇒
     ∀v v1. P v v1
```

[ctrOut_def]

```
⊢ (ctrOut (COUNT n) (load k) = DISPLAY k) ∧
  (ctrOut (COUNT n) count = DISPLAY (n - 1)) ∧
  (ctrOut (COUNT n) hold = DISPLAY n)
```

[ctrOut_ind]

```
⊢ ∀P.
     (∀n k. P (COUNT n) (load k)) ∧ (∀n. P (COUNT n) count) ∧
     (∀n. P (COUNT n) hold) ⇒
     ∀v v1. P v v1
```

[ctrOut_one_one]

```
⊢ ∀a a'. (DISPLAY a = DISPLAY a') ⇔ (a = a')
```

[ctrState_one_one]

```
⊢ ∀a a'. (COUNT a = COUNT a') ⇔ (a = a')
```

[ctrTR_clauses]

```
⊢ (∀x x1s s1 out1s x2s out2s s2.
      TR x (CFG x1s s1 out1s) (CFG x2s s2 out2s) ⇔
     ∃NS Out ins.
       (x1s = x::ins) ∧ (x2s = ins) ∧ (s2 = NS s1 x) ∧
       (out2s = Out s1 x::out1s)) ∧
  ∀x x1s s1 out1s x2s out2s.
    TR x (CFG x1s s1 out1s)
      (CFG x2s (ctrNS s1 x) (ctrOut s1 x::out2s)) ⇔
    ∃ins. (x1s = x::ins) ∧ (x2s = ins) ∧ (out2s = out1s)
```

[ctrTR_rules]

```
⊢ ∀s x ins outs.
    TR x (CFG (x::ins) s outs)
      (CFG ins (ctrNS s x) (ctrOut s x::outs))
```

[ctrTrans_Equiv_TR]

```
⊢ TR x (CFG (x::ins) s outs)
    (CFG ins (ctrNS s x) (ctrOut s x::outs)) ⇔
  Trans x s (ctrNS s x)
```

Reproducibility in ML and LaTeX

All ML and LaTeX source files compile well on the environment provided by this course. Please use the *Makefile* provided in subdirectory */HOL/* by typing command *make* or *make clean*.

Chapter 2

Exercise 16.3.1

In this exercise, we will define a state machine named *M1* and provide the corresponding datatypes, definitions or proofs:

Before we go through the following sections, we will need to enter the folder */M1/* and open up a new HOL session, then we will run

```
app load ["../sm/smTheory", "../sminfRules"];

open HolKernel Parse boolLib bossLib;
open TypeBase;

open sminfRules smTheory;
```

to make sure that all the following codes run.

2.1 Exercise 16.3.1.A

In this section, we will provide several datatypes and their properties.

2.1.1 Relevant Code

We will need to define three datatypes for our state machine *M1*, which are respectively *command*, *state* and *output*.

```
val _ =
Datatype
`command = i0 | i1`;

val _ =
Datatype `state = S0 | S1 | S2`;

val _ =
Datatype `output = o0 | o1`;
```

Also, we will need to prove the properties for these datatypes

```
val command_distinct_clauses = distinct_of ``:command``;
val _ = save_thm ("command_distinct_clauses", command_distinct_clauses);

val state_distinct_clauses = distinct_of ``:state``;
val _ = save_thm("state_distinct_clauses", state_distinct_clauses);

val output_distinct_clauses = distinct_of ``:output``;
val _ = save_thm("output_distinct_clauses", output_distinct_clauses);
```

2.1.2 Session Transcript

If we send the first code snippet that defined the datatypes to HOL session, we will see transcript as below:

```
> # # # <<HOL message: Defined type: "command">>                                                    1
> > # <<HOL message: Defined type: "state">>
> > # <<HOL message: Defined type: "output">>
```

Then we will send the second code snippet, which shows the properties of the datatypes, to the HOL session, we will see transcript as below (if we set the printing switches ”unicode” off and ”show types” on):

```
> # val command_distinct_clauses =                                                                   2
   |- i0 <> i1:
   thm
> > > val state_distinct_clauses =
   |- S0 <> S1 /\ S0 <> S2 /\ S1 <> S2:
   thm
> > > val output_distinct_clauses =
   |- o0 <> o1:
   thm
> >
```

2.2 Exercise 16.3.1.B

In this section, we will provide our definition of next state function *M1ns_def* and output function *M1out_def*, for our state machine *M1*

2.2.1 Relevant Code

We will use the following code to define our next state function *M1ns_def* and output function *M1out_def*.

```
val M1ns_def =
Define `(M1ns S0 i0 = S1) /\
(M1ns (S0: state) (i1: command) = (S2: state)) /\
(M1ns (S1: state) (i0: command) = (S0:state)) /\
(M1ns (S1: state) (i1: command) = (S0: state)) /\
(M1ns (S2: state) (i0: command)= (S2: state)) /\
(M1ns (S2: state) (i1: command) = (S2: state))`;


val M1out_def =
Define `(M1out S0 i0 = o0)/\(M1out S0 i1 = o1)/\(M1out S1 i0 = o0)/\(M1out S1
    i1 = o0)/\(M1out S2 i0 = o1)/\(M1out S2 i1 = o1)`;
```

2.2.2 Session Transcript

If we send the above code to HOL, we will see the transcript as below(if we keep the ”unicode” off and ”show types” on):

```
> # # # # # # # <<HOL warning: GrammarDeltas.revise_data:                    3
  Grammar-deltas:
    overload_on("M1ns_tupled")
  invalidated by DelConstant(scratch$M1ns_tupled)>>
Equations stored under "M1ns_def".
Induction stored under "M1ns_ind".
val M1ns_def =
   |- (M1ns S0 i0 = S1) /\ (M1ns S0 i1 = S2) /\ (M1ns S1 i0 = S0) /\
   (M1ns S1 i1 = S0) /\ (M1ns S2 i0 = S2) /\ (M1ns S2 i1 = S2):
   thm
> > > # <<HOL warning: GrammarDeltas.revise_data:
  Grammar-deltas:
    overload_on("M1out_tupled")
  invalidated by DelConstant(scratch$M1out_tupled)>>
Equations stored under "M1out_def".
Induction stored under "M1out_ind".
val M1out_def =
   |- (M1out S0 i0 = o0) /\ (M1out S0 i1 = o1) /\ (M1out S1 i0 = o0) /\
   (M1out S1 i1 = o0) /\ (M1out S2 i0 = o1) /\ (M1out S2 i1 = o1):
   thm
>
*** Emacs/HOL command completed ***
```

The pretty-printed definitions should look like this:
M1ns_def:

$\vdash$ (M1ns S0 i0 = S1) $\wedge$ (M1ns S0 i1 = S2) $\wedge$ (M1ns S1 i0 = S0) $\wedge$
(M1ns S1 i1 = S0) $\wedge$ (M1ns S2 i0 = S2) $\wedge$ (M1ns S2 i1 = S2)

M1ns_ind:

$\vdash \forall P.$
P S0 i0 $\wedge$ P S0 i1 $\wedge$ P S1 i0 $\wedge$ P S1 i1 $\wedge$ P S2 i0 $\wedge$ P S2 i1 $\Rightarrow$
$\forall v\ v_1.\ P\ v\ v_1$

M1out_def:

$\vdash$ (M1out S0 i0 = o0) $\wedge$ (M1out S0 i1 = o1) $\wedge$
(M1out S1 i0 = o0) $\wedge$ (M1out S1 i1 = o0) $\wedge$
(M1out S2 i0 = o1) $\wedge$ (M1out S2 i1 = o1)

M1out_ind:

$\vdash \forall P.$
P S0 i0 $\wedge$ P S0 i1 $\wedge$ P S1 i0 $\wedge$ P S1 i1 $\wedge$ P S2 i0 $\wedge$ P S2 i1 $\Rightarrow$
$\forall v\ v_1.\ P\ v\ v_1$

2.3 Exercise 16.3.1.C

In this section, we will prove some theorems for our state machine *M1*.

2.3.1 Relevant Code

We used the following code on *m1TR_rules*.

```
val m1TR_rules = SPEC_TR ``M1ns`` ``M1out``;
val _ = save_thm("m1TR_rules", m1TR_rules);
```

As for *m1TR_clauses*, we have

```
val m1TR_clauses = SPEC_TR_clauses ``M1ns`` ``M1out``;
val _ = save_thm("m1TR_clauses", m1TR_clauses);
```

We will use the following code to prove *m1Trans_Equiv_TR*

```
val m1Trans_Equiv_TR = SPEC_Trans_Equiv_TR ``M1ns`` ``M1out``;
val _ = save_thm("m1Trans_Equiv_TR", m1Trans_Equiv_TR);
```

We can prove *m1_rules* with the following code

```
val th1 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL[``S0``, ``i0``] m1TR_rules);

val th2 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S0``, ``i1``] m1TR_rules);

val th3 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S1``, ``i0``] m1TR_rules);

val th4 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S1``, ``i1``] m1TR_rules);

val th5 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S2``, ``i0``] m1TR_rules);

val th6 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S2``, ``i1``] m1TR_rules);


val m1_rules = LIST_CONJ[th1, th2, th3, th4, th5, th6];
val _ = save_thm("m1_rules", m1_rules);
```

2.3.2 Session Transcript

If we send all the codes above to HOL, we will see the transcript as below (still, "unicode" is off and "show types" is on):

4

```
> val m1TR_rules =
   |- !(s :state) (x :command) (ins :command list) (outs :output list).
     TR x (CFG (x::ins) s outs) (CFG ins (M1ns s x) (M1out s x::outs)):
   thm
val m1TR_clauses =
   |- (!(x :'input) (x1s :'input list) (s1 :'state) (out1s :'output list)
       (x2s :'input list) (out2s :'output list) (s2 :'state).
      TR x (CFG x1s s1 out1s) (CFG x2s s2 out2s) <=>
      ?(NS :'state -> 'input -> 'state)
         (Out :'state -> 'input -> 'output) (ins :'input list).
        (x1s = x::ins) /\ (x2s = ins) /\ (s2 = NS s1 x) /\
        (out2s = Out s1 x::out1s)) /\
   !(x :command) (x1s :command list) (s1 :state) (out1s :output list)
      (x2s :command list) (out2s :output list).
     TR x (CFG x1s s1 out1s)
       (CFG x2s (M1ns s1 x) (M1out s1 x::out2s)) <=>
     ?(ins :command list).
       (x1s = x::ins) /\ (x2s = ins) /\ (out2s = out1s):
   thm
val m1Trans_Equiv_TR =
   |- TR (x :command)
     (CFG (x::(ins :command list)) (s :state) (outs :output list))
     (CFG ins (M1ns s x) (M1out s x::outs)) <=> Trans x s (M1ns s x):
   thm
val th1 =
   |- !(ins :command list) (outs :output list).
     TR i0 (CFG (i0::ins) S0 outs) (CFG ins S1 (o0::outs)):
   thm
val th2 =
   |- !(ins :command list) (outs :output list).
     TR i1 (CFG (i1::ins) S0 outs) (CFG ins S2 (o1::outs)):
   thm
val th3 =
   |- !(ins :command list) (outs :output list).
     TR i0 (CFG (i0::ins) S1 outs) (CFG ins S0 (o0::outs)):
   thm
val th4 =
   |- !(ins :command list) (outs :output list).
     TR i1 (CFG (i1::ins) S1 outs) (CFG ins S0 (o0::outs)):
   thm
val th5 =
   |- !(ins :command list) (outs :output list).
     TR i0 (CFG (i0::ins) S2 outs) (CFG ins S2 (o1::outs)):
   thm
val th6 =
   |- !(ins :command list) (outs :output list).
     TR i1 (CFG (i1::ins) S2 outs) (CFG ins S2 (o1::outs)):
   thm
val m1_rules =
   |- (!(ins :command list) (outs :output list).
      TR i0 (CFG (i0::ins) S0 outs) (CFG ins S1 (o0::outs))) /\
   (!(ins :command list) (outs :output list).
      TR i1 (CFG (i1::ins) S0 outs) (CFG ins S2 (o1::outs))) /\
   (!(ins :command list) (outs :output list).
      TR i0 (CFG (i0::ins) S1 outs) (CFG ins S0 (o0::outs))) /\
   (!(ins :command list) (outs :output list).
      TR i1 (CFG (i1::ins) S1 outs) (CFG ins S0 (o0::outs))) /\
   (!(ins :command list) (outs :output list).
      TR i0 (CFG (i0::ins) S2 outs) (CFG ins S2 (o1::outs))) /\
   !(ins :command list) (outs :output list).
     TR i1 (CFG (i1::ins) S2 outs) (CFG ins S2 (o1::outs)):
   thm
val it = (): unit
>
```

Chapter 3

Exercise 16.3.2

In this exercise, we will define a state machine named *Counter* and provide the corresponding datatypes, definitions or proofs:

Before we go through the following sections, we will need to enter the folder *Counter* and open up a new HOL session, then we will run

```
app load ["../sm/smTheory", "../sminfRules"];

open HolKernel Parse boolLib bossLib;
open TypeBase smTheory sminfRules listTheory;
```

so all the following codes will work.

3.1 Exercise 16.3.2.A

In this section, we will provide several datatypes and their properties, for our state machine *Counter*.

3.1.1 Relevant Code

We will need to define three datatypes for our state machine *Counter*, which are respectively *ctrcmd*, *ctrState* and *ctrOut*.

```
val _ =
Datatype `ctrcmd = load num | count | hold`;

val _ =
Datatype `ctrState = COUNT num`;

val _ =
Datatype `ctrOut = DISPLAY num`;
```

Also, we will need to prove the properties for these datatypes

```
val ctrcmd_distinct_clauses = distinct_of ``:ctrcmd``;

val ctrState_one_one = one_one_of ``:ctrState``;

val ctrOut_one_one = one_one_of ``:ctrOut``;
```

3.1.2 Session Transcript

If we send the first code snippet that defined the datatypes to HOL session, we will see transcript as below (if we keep the printing switches "unicode" off and "show types" on):

```
> # <<HOL message: Defined type: "ctrcmd">>                                    5
> > # <<HOL message: Defined type: "ctrState">>
> > # <<HOL message: Defined type: "ctrOut">>
```

Then we will send the second code snippet, which shows the properties of the datatypes, to the HOL session, we will see transcript as below :

```
> val ctrcmd_distinct_clauses =                                          6
   |- (!(a :num). load a <> count) /\ (!(a :num). load a <> hold) /\
   count <> hold:
   thm
> > val ctrState_one_one =
   |- !(a :num) (a' :num). (COUNT a = COUNT a') <=> (a = a'):
   thm
> > val ctrOut_one_one =
   |- !(a :num) (a' :num). (DISPLAY a = DISPLAY a') <=> (a = a'):
   thm
```

3.2 Exercise 16.3.2.B

In this section, we will provide our definition of next state function *ctrNS_def* and output function *ctrOut_def*, for our state machine *Counter*.

3.2.1 Relevant Code

We will use the following code to define our next state function *ctrNS_def* and output function *ctrOut_def*.

```
val ctrNS_def =
Define
`(ctrNS (COUNT n) (load k) = (COUNT k)) /\
(ctrNS (COUNT n) count = (COUNT (n-1)))/\
(ctrNS (COUNT n) (hold) =  (COUNT n))`


val ctrOut_def =
Define
`(ctrOut (COUNT n) (load k) = (DISPLAY k)) /\
 (ctrOut (COUNT n) (count) = (DISPLAY (n-1))) /\
 (ctrOut (COUNT n) (hold) = (DISPLAY n))`
```

3.2.2 Session Transcript

If we send the above code to HOL, we will see the transcript as below(still, the "show types" is on while "unicode" is off):

```
> # # # # # # # # # # # <<HOL warning: GrammarDeltas.revise_data:                 7
  Grammar-deltas:
    overload_on("ctrNS_tupled")
  invalidated by DelConstant(scratch$ctrNS_tupled)>>
Equations stored under "ctrNS_def".
Induction stored under "ctrNS_ind".
<<HOL warning: GrammarDeltas.revise_data:
  Grammar-deltas:
    overload_on("ctrOut_tupled")
  invalidated by DelConstant(scratch$ctrOut_tupled)>>
Equations stored under "ctrOut_def".
Induction stored under "ctrOut_ind".
val ctrNS_def =
   |- !(n :num) (k :num).
     (ctrNS (COUNT n) (load k) = COUNT k) /\
     (ctrNS (COUNT n) count = COUNT (n - (1 :num))) /\
     (ctrNS (COUNT n) hold = COUNT n):
   thm
val ctrOut_def =
   |- !(n :num) (k :num).
     (ctrOut (COUNT n) (load k) = DISPLAY k) /\
     (ctrOut (COUNT n) count = DISPLAY (n - (1 :num))) /\
     (ctrOut (COUNT n) hold = DISPLAY n):
   thm
>
```

The pretty-printed definitions should look like this:
ctrNS_def:

$\vdash$ `(ctrNS (COUNT` n`) (load` k`) = COUNT` k`)` $\wedge$
`(ctrNS (COUNT` n`) count = COUNT (`n `- 1))` $\wedge$
`(ctrNS (COUNT` n`) hold = COUNT` n`)`

ctrNS_ind:

$\vdash \forall P.$
$(\forall n\ k.\ P$ `(COUNT` n`) (load` k`))` $\wedge$ $(\forall n.\ P$ `(COUNT` n`) count)` $\wedge$
$(\forall n.\ P$ `(COUNT` n`) hold)` $\Rightarrow$
$\forall v\ v_1.\ P\ v\ v_1$

ctrOut_def:

$\vdash$ `(ctrOut (COUNT` n`) (load` k`) = DISPLAY` k`)` $\wedge$
`(ctrOut (COUNT` n`) count = DISPLAY (`n `- 1))` $\wedge$
`(ctrOut (COUNT` n`) hold = DISPLAY` n`)`

ctrOut_ind:

$\vdash \forall P.$
$(\forall n\ k.\ P$ `(COUNT` n`) (load` k`))` $\wedge$ $(\forall n.\ P$ `(COUNT` n`) count)` $\wedge$
$(\forall n.\ P$ `(COUNT` n`) hold)` $\Rightarrow$
$\forall v\ v_1.\ P\ v\ v_1$

3.3 Exercise 16.3.2.C

In this section, we will prove some theorems for our state machine *Counter*.

3.3.1 Relevant Code

We used the following code on *ctrTR_rules.*

```
val ctrTR_rules = SPEC_TR ``ctrNS`` ``ctrOut``;
```

As for *ctrTR_clauses*, we have

```
val ctrTR_clauses = SPEC_TR_clauses ``ctrNS`` ``ctrOut``;
```

We will use the following code to prove *ctrTrans_Equiv_TR*

```
val ctrTrans_Equiv_TR = SPEC_Trans_Equiv_TR ``ctrNS`` ``ctrOut``;
```

We can prove *ctr_rules* with the following code

```
val th1 =
REWRITE_RULE
[ctrNS_def, ctrOut_def]
(SPECL[``COUNT n``, ``load new``] ctrTR_rules);

val th2 =
REWRITE_RULE
[ctrNS_def, ctrOut_def]
(SPECL[``COUNT n``, ``count``] ctrTR_rules);

val th3 =
REWRITE_RULE
[ctrNS_def, ctrOut_def]
(SPECL[``COUNT n``, ``hold``] ctrTR_rules);

val ctr_rules = LIST_CONJ [th1, th2, th3];
val _ = save_thm("ctr_rules", ctr_rules);
```

3.3.2 Session Transcript

If we send all the codes above to HOL, we will see the transcript as below (still, ”unicode” is off and ”show types” is on):

8

```
> val ctrTR_rules =
   |- !(s :ctrState) (x :ctrcmd) (ins :ctrcmd list) (outs :ctrOut list).
     TR x (CFG (x::ins) s outs) (CFG ins (ctrNS s x) (ctrOut s x::outs)):
   thm
val ctrTR_clauses =
   |- (!(x :'input) (x1s :'input list) (s1 :'state) (out1s :'output list)
       (x2s :'input list) (out2s :'output list) (s2 :'state).
      TR x (CFG x1s s1 out1s) (CFG x2s s2 out2s) <=>
      ?(NS :'state -> 'input -> 'state)
         (Out :'state -> 'input -> 'output) (ins :'input list).
        (x1s = x::ins) /\ (x2s = ins) /\ (s2 = NS s1 x) /\
        (out2s = Out s1 x::out1s)) /\
   !(x :ctrcmd) (x1s :ctrcmd list) (s1 :ctrState) (out1s :ctrOut list)
      (x2s :ctrcmd list) (out2s :ctrOut list).
     TR x (CFG x1s s1 out1s)
       (CFG x2s (ctrNS s1 x) (ctrOut s1 x::out2s)) <=>
     ?(ins :ctrcmd list).
       (x1s = x::ins) /\ (x2s = ins) /\ (out2s = out1s):
   thm
val ctrTrans_Equiv_TR =
   |- TR (x :ctrcmd)
     (CFG (x::(ins :ctrcmd list)) (s :ctrState) (outs :ctrOut list))
     (CFG ins (ctrNS s x) (ctrOut s x::outs)) <=> Trans x s (ctrNS s x):
   thm
val th1 =
   |- !(ins :ctrcmd list) (outs :ctrOut list).
     TR (load (new :num)) (CFG (load new::ins) (COUNT (n :num)) outs)
       (CFG ins (COUNT new) (DISPLAY new::outs)):
   thm
<<HOL message: more than one resolution of overloading was possible>>
val th2 =
   |- !(ins :ctrcmd list) (outs :ctrOut list).
     TR count (CFG (count::ins) (COUNT (n :num)) outs)
       (CFG ins (COUNT (n - (1 :num))) (DISPLAY (n - (1 :num))::outs)):
   thm
val th3 =
   |- !(ins :ctrcmd list) (outs :ctrOut list).
     TR hold (CFG (hold::ins) (COUNT (n :num)) outs)
       (CFG ins (COUNT n) (DISPLAY n::outs)):
   thm
val ctr_rules =
   |- (!(ins :ctrcmd list) (outs :ctrOut list).
      TR (load (new :num)) (CFG (load new::ins) (COUNT (n :num)) outs)
        (CFG ins (COUNT new) (DISPLAY new::outs))) /\
   (!(ins :ctrcmd list) (outs :ctrOut list).
      TR count (CFG (count::ins) (COUNT n) outs)
        (CFG ins (COUNT (n - (1 :num)))
           (DISPLAY (n - (1 :num))::outs))) /\
   !(ins :ctrcmd list) (outs :ctrOut list).
     TR hold (CFG (hold::ins) (COUNT n) outs)
       (CFG ins (COUNT n) (DISPLAY n::outs)):
   thm
val it = (): unit
>
```

Appendix A

Source Code for smScript.sml

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

```
(******************************)
(* State machine theory       *)
(* Author: Shiu-Kai Chin      *)
(* Date: 01 January 2014,     *)
(* Modified 06 August 2015    *)
(******************************)

structure smScript = struct

(* ==== Interactive mode ====
app load ["TypeBase","listTheory","smTheory"];
open TypeBase listTheory smTheory
 ==== end interactive mode ==== *)

open HolKernel boolLib Parse bossLib
open TypeBase listTheory
(*********************)
(* create a new theory *)
(*********************)
val _ = new_theory "sm";

(**********************************************************************
    *)
(* State-based transition relation
    *)
(*
    *)
(*
    *)
(**********************************************************************
    *)
val (Trans_rules, Trans_ind, Trans_cases) =
Hol_reln
`!NS (s:'state) (x:'input).
  Trans x s ((NS:'state -> 'input -> 'state) s x)`

(***********************)
(* Define configurations *)
(***********************)
val _ =
Datatype
`configuration = CFG ('input list) 'state ('output list)`
```

```
(*
   ***********************************************************************
   *)
(* Note: configuration_11, configuration_induction, and configuration_nchotomy
    *)
(* are proved and available when fsmTheory is loaded and opened
                    *)
(*
   ***********************************************************************
   *)
val configuration_11 = one_one_of ``:('input,'state,'output)configuration``
val configuration_one_one = one_one_of ``:('input,'state,'output)configuration
   ``

val _ = save_thm("configuration_11",configuration_11)
val _ = save_thm("configuration_one_one",configuration_one_one)

(**********************************************)
(* Define transition relation among configurations *)
(* This definition is parameterized in terms of    *)
(* next state transition and output relations      *)
(**********************************************)
val (TR_rules, TR_ind, TR_cases) =
Hol_reln
`!NS Out (s:'state) (x:'input) (ins:'input list) (outs:'output list).
  TR x (CFG (x::ins) s outs)(CFG ins (NS s x) ((Out s x)::outs))`

val lemma1 =
ISPECL [``x:'input``,``CFG (x1s:'input list) (s1:'state) (out1s:'output list)
    ``,
        ``CFG (x2s:'input list) (s2:'state) (out2s:'output list)``] TR_cases

val lemma2 =
TAC_PROOF(
([],
``!x x1s s1 out1s x2s out2s s2.
  TR (x:'input) (CFG (x1s:'input list) (s1:'state) (out1s:'output list)) (CFG
      (x2s:'input list) (s2:'state) (out2s:'output list)) <=>
   ?NS Out ins.(x1s = x::ins) /\ (x2s = ins) /\ (s2 = NS s1 x) /\ (out2s = (Out
       s1 x)::out1s)``),
REWRITE_TAC[lemma1,configuration_11,list_11] THEN
REPEAT GEN_TAC THEN
EQ_TAC THEN
REPEAT STRIP_TAC THEN
EXISTS_TAC ``NS:'state -> 'input -> 'state`` THEN
EXISTS_TAC ``Out:'state -> 'input -> 'output`` THEN
ASM_REWRITE_TAC[] THENL
[(EXISTS_TAC``ins:'input list`` THEN PROVE_TAC []),
 ALL_TAC] THEN
EXISTS_TAC``s1:'state`` THEN
EXISTS_TAC``ins:'input list`` THEN
EXISTS_TAC``out1s:'output list`` THEN
REWRITE_TAC[])

val lemma3 =
ISPECL [``x:'input``,``CFG (x1s:'input list) (s1:'state) (out1s:'output list)
    ``,
        ``CFG
          (x2s:'input list)
          ((NS:'state -> 'input -> 'state) s1 x)
          ((Out:'state -> 'input -> 'output) s1 x::out2s)``] TR_cases

val lemma4 =
TAC_PROOF(([],
``!(NS:'state -> 'input -> 'state)(Out:'state -> 'input -> 'output)(x:'input)(
    x1s:'input list)(s1:'state)out1s x2s out2s.
  TR (x:'input) (CFG (x1s:'input list) (s1:'state) (out1s:'output list)) (CFG
      (x2s:'input list) (NS s1 x) (Out s1 x::out2s)) <=>
  ?ins.(x1s = x::ins) /\ (x2s = ins) /\ (out2s = out1s)``),
REWRITE_TAC[lemma3,configuration_11,list_11] THEN
REPEAT GEN_TAC THEN
EQ_TAC THEN
REPEAT STRIP_TAC THENL
[(EXISTS_TAC``ins:'input list`` THEN
  ASM_REWRITE_TAC[]),
 (EXISTS_TAC ``NS:'state -> 'input -> 'state`` THEN
  EXISTS_TAC ``Out:'state -> 'input -> 'output`` THEN
  EXISTS_TAC ``s1:'state`` THEN
  EXISTS_TAC ``ins:'input list`` THEN
  EXISTS_TAC ``out1s:'output list`` THEN
  ASM_REWRITE_TAC[])])

val TR_clauses = CONJ lemma2 lemma4
val _ = save_thm("TR_clauses",TR_clauses)


(*********************************)
(* Proof that TR is deterministic *)
(*********************************)

val lemma1 =
TAC_PROOF(([],
``!(NS:'state -> 'input -> 'state)(Out:'state -> 'input -> 'output).
   (TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2)))
      ==>
   (TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2'(NS s1 x1) ((Out s1 x1)::outs2'))
     ) ==>
   (ins2 = ins2') /\ (outs2 = outs2')``),
REWRITE_TAC[TR_clauses] THEN
REPEAT STRIP_TAC THEN
ASM_REWRITE_TAC [] THEN
IMP_RES_TAC list_11 THEN
PROVE_TAC[])

val lemma2 =
TAC_PROOF(([],
``!(NS:'state -> 'input -> 'state)(Out:'state -> 'input -> 'output).
   (TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2)))
       ==>
   (TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2'(NS s1 x1) ((Out s1 x1)::outs2'))
      ) ==>
   ((CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2)) = (CFG ins2'(NS s1 x1) ((Out s1
        x1)::outs2')))``),
REWRITE_TAC[configuration_11,list_11] THEN
REPEAT STRIP_TAC THEN

IMP_RES_TAC lemma1)

val lemma3 =
TAC_PROOF(([],
``!(NS:'state -> 'input -> 'state)(Out:'state -> 'input -> 'output).
   ((CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2)) = (CFG ins2'(NS s1 x1) ((Out s1
        x1)::outs2'))) /\
   (TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2)))
       ==>
   ((TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2))
      ) /\
    (TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2'(NS s1 x1) ((Out s1 x1)::outs2')
        )))``),
PROVE_TAC[])

val TR_deterministic =
TAC_PROOF(([],
``!(NS:'state -> 'input -> 'state)(Out:'state -> 'input -> 'output) x1 ins1 s1
     outs1 ins2 ins2' outs2 outs2'.
   ((TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2))
      ) /\
    (TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2'(NS s1 x1) ((Out s1 x1)::outs2')
        ))) =
   (((CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2)) = (CFG ins2'(NS s1 x1) ((Out
      s1 x1)::outs2'))) /\
    (TR x1 (CFG (x1::ins1) s1 outs1)(CFG ins2 (NS s1 x1) ((Out s1 x1)::outs2))
        ))``),
PROVE_TAC[lemma2, lemma3])

val _ = save_thm("TR_deterministic",TR_deterministic)

(****************************************)
(* Proof that TR is completely specified *)
(* if NS and Out are total functions.    *)
(****************************************)
val TR_complete =
TAC_PROOF(([],
``!(s:'state)(x:'input)(ins:'input list)(outs:'output list).?(s':'state)(out:'
   output).
      (TR (x:'input) (CFG (x::ins) (s:'state) (outs:'output list))(CFG ins (s
          ':'state) (out::outs)))``),
REPEAT STRIP_TAC THEN
```

```
REWRITE_TAC[TR_cases] THEN
EXISTS_TAC``(NS:'state -> 'input -> 'state) s x`` THEN
EXISTS_TAC``(Out:'state -> 'input -> 'output) s x`` THEN
EXISTS_TAC``(NS:'state -> 'input -> 'state)`` THEN
EXISTS_TAC``(Out:'state -> 'input -> 'output)`` THEN
EXISTS_TAC``s:'state`` THEN
EXISTS_TAC``ins:'input list`` THEN
EXISTS_TAC``outs:'output list`` THEN
REWRITE_TAC[])

val _ = save_thm("TR_complete",TR_complete)

(***************************************************************************
    *)
(* Show trans and TR are equivalent
    *)
(*
    *)
(*
    *)
(***************************************************************************
    *)
val Trans_TR_lemma =
TAC_PROOF(([],``(Trans (x:'input) (s:'state) (NS s x)) ==>
(TR x (CFG (x::ins) s (outs:'output list))(CFG ins (NS s x) ((Out s x)::outs))
    )``),
STRIP_TAC THEN
PROVE_TAC[TR_rules])

val _ = save_thm("Trans_TR_lemma",Trans_TR_lemma)

val TR_Trans_lemma =
TAC_PROOF(([],``(TR (x:'input) (CFG (x::ins) (s:'state)(outs:'output list))(
    CFG ins (NS s x)((Out s x)::outs))) ==>
        (Trans (x:'input) (s:'state) (NS s x))``),
STRIP_TAC THEN
IMP_RES_TAC TR_cases THEN
PAT_ASSUM
``CFG (x::ins) s outs = CFG (x::ins') s' outs'``
(fn th => ASSUME_TAC(REWRITE_RULE[configuration_11,list_11]th)) THEN
PROVE_TAC[Trans_rules])

val _ = save_thm("TR_Trans_lemma",TR_Trans_lemma)

val Trans_Equiv_TR =
TAC_PROOF(([],``(TR (x:'input) (CFG (x::ins) (s:'state)(outs:'output list))(
    CFG ins (NS s x)((Out s x)::outs))) =
        (Trans (x:'input) (s:'state) (NS s x))``),
PROVE_TAC[TR_Trans_lemma,Trans_TR_lemma])

val _ = save_thm("Trans_Equiv_TR",Trans_Equiv_TR)

(* ==== start here ====
 ==== end here ==== *)

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

end (* structure *)
```

Appendix B

Source Code for m1Script.sml

The following code is from *m1Script.sml*, which is located in directory "../HOL/M1"

```
app load ["../sm/smTheory", "../sminfRules"];
structure m1Script = struct

open HolKernel Parse boolLib bossLib;
open TypeBase;

open sminfRules smTheory;


val _ = new_theory "m1";

val _ =
Datatype
`command = i0 | i1 `;

val _ =
Datatype `state = S0 | S1 | S2`;

val _ =
Datatype `output = o0 | o1`;

val command_distinct_clauses = distinct_of ``:command``;
val _ = save_thm ("command_distinct_clauses", command_distinct_clauses);

val state_distinct_clauses = distinct_of ``:state``;
val _ = save_thm("state_distinct_clauses", state_distinct_clauses);

val output_distinct_clauses = distinct_of ``:output``;
val _ = save_thm("output_distinct_clauses", output_distinct_clauses);


val M1ns_def =
Define `(M1ns S0 i0 = S1) /\
(M1ns (S0: state) (i1: command) = (S2: state)) /\
(M1ns (S1: state) (i0: command) = (S0:state)) /\
(M1ns (S1: state) (i1: command) = (S0: state)) /\
(M1ns (S2: state) (i0: command)= (S2: state)) /\
(M1ns (S2: state) (i1: command) = (S2: state))`;


val M1out_def =
Define `(M1out S0 i0 = o0)/\(M1out S0 i1 = o1)/\(M1out S1 i0 = o0)/\(M1out S1
    i1 = o0)/\(M1out S2 i0 = o1)/\(M1out S2 i1 = o1)`;

val m1TR_rules = SPEC_TR ``M1ns`` ``M1out``;
val _ = save_thm("m1TR_rules", m1TR_rules);

val m1TR_clauses = SPEC_TR_clauses ``M1ns`` ``M1out``;
val _ = save_thm("m1TR_clauses", m1TR_clauses);

val m1Trans_Equiv_TR = SPEC_Trans_Equiv_TR ``M1ns`` ``M1out``;
val _ = save_thm("m1Trans_Equiv_TR", m1Trans_Equiv_TR);

val th1 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL[``S0``, ``i0``] m1TR_rules);

val th2 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S0``, ``i1``] m1TR_rules);

val th3 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S1``, ``i0``] m1TR_rules);

val th4 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S1``, ``i1``] m1TR_rules);

val th5 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S2``, ``i0``] m1TR_rules);

val th6 =
REWRITE_RULE
[M1ns_def, M1out_def]
(SPECL [``S2``, ``i1``] m1TR_rules);


val m1_rules = LIST_CONJ[th1, th2, th3, th4, th5, th6];
val _ = save_thm("m1_rules", m1_rules);

val _ = export_theory();

end
```

Appendix C

Source Code for counterScript.sml

The following code is from *m1Script.sml*, which is located in directory "../HOL/Counter"

```
app load ["../sm/smTheory", "../sminfRules"];
structure counterScript = struct

open HolKernel Parse boolLib bossLib;
open TypeBase smTheory sminfRules listTheory;

val _ = new_theory "counter";

val _ =
Datatype `ctrcmd = load num | count | hold`;

val _ =
Datatype `ctrState = COUNT num`;

val _ =
Datatype `ctrOut = DISPLAY num`;


val ctrcmd_distinct_clauses = distinct_of ``:ctrcmd``;
val _ = save_thm("ctrcmd_distinct_clauses", ctrcmd_distinct_clauses);

val ctrState_one_one = one_one_of ``:ctrState``;
val _ = save_thm("ctrState_one_one", ctrState_one_one);

val ctrOut_one_one = one_one_of ``:ctrOut``;
val _ = save_thm("ctrOut_one_one", ctrOut_one_one);

val ctrNS_def =
Define
`(ctrNS (COUNT n) (load k) = (COUNT k)) /\
(ctrNS (COUNT n) count = (COUNT (n-1)))/\
(ctrNS (COUNT n) (hold) =  (COUNT n))`


val ctrOut_def =
Define
`(ctrOut (COUNT n) (load k) = (DISPLAY k)) /\
 (ctrOut (COUNT n) (count) = (DISPLAY (n-1))) /\
 (ctrOut (COUNT n) (hold) = (DISPLAY n))`

val ctrTR_rules = SPEC_TR ``ctrNS`` ``ctrOut``;
val _ = save_thm("ctrTR_rules", ctrTR_rules);

val ctrTR_clauses = SPEC_TR_clauses ``ctrNS`` ``ctrOut``;
val _ = save_thm("ctrTR_clauses", ctrTR_clauses);

val ctrTrans_Equiv_TR = SPEC_Trans_Equiv_TR ``ctrNS`` ``ctrOut``;
val _ = save_thm("ctrTrans_Equiv_TR", ctrTrans_Equiv_TR);

val th1 =
REWRITE_RULE
[ctrNS_def, ctrOut_def]
(SPECL[``COUNT n``, ``load new``] ctrTR_rules);

val th2 =
REWRITE_RULE
[ctrNS_def, ctrOut_def]
(SPECL[``COUNT n``, ``count``] ctrTR_rules);

val th3 =
REWRITE_RULE
[ctrNS_def, ctrOut_def]
(SPECL[``COUNT n``, ``hold``] ctrTR_rules);

val ctr_rules = LIST_CONJ [th1, th2, th3];
val _ = save_thm("ctr_rules", ctr_rules);

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