Project 8 Report

Kyle Peppe

March 4, 2020

Abstract

This project demonstrated my ability to create machine states, defining datatypes, and using the previous 2 things to prove the theroems provided for the many theorems for the problems 16.3.1 and 16.3.2. 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

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:

Reproducibility in ML and LaTeX
The ML and LaTeX source files compile with no errors.

Chapter 2

Excercise 16.3.1

2.1 Problem statement

This problem had 3 different sections, in the first section I had to define datatypes for the inputs, states and outputs. For the 2nd part I defined the next state and output functions, followed by proving 4 theories about the m1.

2.2 16-3-1A: Definitions and Theorems of Datatypes

```
(* Setting the datatypes                              *)
(* Part A                                             *)
val _ = Datatype`command = i0 | i1`
val command_distinct_clauses = distinct_of ``:command``
(* Save the Theorem                                   *)
val _ = save_thm("command_distinct_clauses",command_distinct_clauses)

val _ = Datatype`state = S0 | S1 | S2`
val state_distinct_clauses = distinct_of ``:state``
(* Save the Theorem                                   *)
val _ = save_thm("state_distinct_clauses",state_distinct_clauses)

val _ = Datatype`output = o0 | o1`
val output_distinct_clauses = distinct_of ``:output``
(* Save the Theorem                                   *)
val _ = save_thm("output_distinct_clauses",output_distinct_clauses)

> val _ = Datatype`command = i0 | i1`;
val command_distinct_clauses = distinct_of ``:command``;
<<HOL message: Defined type: "command">>
> val command_distinct_clauses =
   |- i0 <> i1:
   thm
> val _ = Datatype`state = S0 | S1 | S2`;
val state_distinct_clauses = distinct_of ``:state``;
<<HOL message: Defined type: "state">>
> val state_distinct_clauses =
   |- S0 <> S1 /\ S0 <> S2 /\ S1 <> S2:
   thm
> val _ = Datatype`output = o0 | o1`;
val output_distinct_clauses = distinct_of ``:output``;
<<HOL message: Defined type: "output">>
> val output_distinct_clauses =
   |- o0 <> o1:
   thm
```

2.3 16-3-1B: Definitions of M1ns and M1out

```
(* Part B                                                    *)
val M1ns_def =
    Define `(M1ns S0 i0 = S1) /\ (M1ns S0 i1 = S2) /\
       (M1ns S1 i0 = S0) /\ (M1ns S1 i1 = S0) /\
       (M1ns S2 i0 = S2) /\ (M1ns S2 i1 = S2)`;


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

2.4 16-3-1C: Proofs of m1TR_rules, m1TR_clauses, m1Trans_Equiv_-TR, and m1_rules

```
(* Part C                                                    *)
val m1TR_rules = SPEC_TR ``M1ns`` ``M1out``

(* Save the Theorem                                          *)
val _ = save_thm("m1TR_rules",m1TR_rules)
```

```
val m1TR_clauses = SPEC_TR_clauses ``M1ns`` ``M1out``

(* Save the Theorem                                      *)
val _ = save_thm("m1TR_clauses",m1TR_clauses)

val m1Trans_Equiv_TR = SPEC_Trans_Equiv_TR ``M1ns`` ``M1out``

(* Save the Theorem                                      *)
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]

(* Save the Theorem                                 *)
val _ = save_thm("m1_rules",m1_rules)

> val m1TR_rules = SPEC_TR ``M1ns`` ``M1out``;
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 = SPEC_TR_clauses ``M1ns`` ``M1out``;
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 = SPEC_Trans_Equiv_TR ``M1ns`` ``M1out``;
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 = 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 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
>
```

Chapter 3

Exercise 16.3.2

3.1 Problem statement

This problem had 3 different sections, in the first section I had to define datatypes for the inputs, states and outputs. For the 2nd part I defined the next state and output functions, followed by proving 4 theories about the counter.

3.2 16-3-2A: Definitions and Theorems of Datatypes

```
(* Setting the datatypes                                    *)
(* Part A                                                   *)
val _ = Datatype `ctrcmd = load num | count | hold`
val ctrcmd_distinct_clauses = distinct_of ``:ctrcmd``
(* Save the Theorem                                         *)
val _ = save_thm("ctrcmd_distinct_clauses",ctrcmd_distinct_clauses)

val _ = Datatype `ctrState = COUNT num`
val ctrState_one_one = one_one_of ``:ctrState``
(* Save the Theorem                                         *)
val _ = save_thm("ctrState_one_one",ctrState_one_one)

val _ = Datatype `ctrOut = DISPLAY num`
val ctrOut_one_one = one_one_of ``:ctrOut``
(* Save the Theorem                                         *)
val _ = save_thm("ctrOut_one_one",ctrOut_one_one)

> val _ = Datatype `ctrcmd = load num | count | hold`;
val ctrcmd_distinct_clauses = distinct_of ``:ctrcmd``;
<<HOL message: Defined type: "ctrcmd">>
> val ctrcmd_distinct_clauses =
   |- (!(a :num). load a <> count) /\ (!(a :num). load a <> hold) /\
   count <> hold:
   thm
> val _ = Datatype `ctrState = COUNT num`;
val ctrState_one_one = one_one_of ``:ctrState``;
<<HOL message: Defined type: "ctrState">>
> val ctrState_one_one =
   |- !(a :num) (a' :num). (COUNT a = COUNT a') <=> (a = a'):
   thm
> val _ = Datatype `ctrOut = DISPLAY num`;
val ctrOut_one_one = one_one_of ``:ctrOut``;
<<HOL message: Defined type: "ctrOut">>
> val ctrOut_one_one =
   |- !(a :num) (a' :num). (DISPLAY a = DISPLAY a') <=> (a = a'):
```

```
thm
```

3.3 16-3-2B: Definitions of ctrNS_def and ctrOut_def

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

3.4 16-3-2C: Proofs of ctrTR_rules, ctrTR_clauses, ctrTrans_Equiv_TR, and ctr_rules

```
(* Part C                                              *)
val ctrTR_rules = SPEC_TR ``ctrNS`` ``ctrOut``

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

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

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

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

(* Save the Theorem                                    *)
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]

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

> val ctrTR_rules = SPEC_TR ``ctrNS`` ``ctrOut``;
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 = SPEC_TR_clauses ``ctrNS`` ``ctrOut``;
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 = SPEC_Trans_Equiv_TR ``ctrNS`` ``ctrOut``;
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 = REWRITE_RULE [ctrNS_def,ctrOut_def] (SPECL[``COUNT n``,``load new``] ctrTR_rul
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 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
>
```

Appendix A

Source Code for smScript.sml

```
(*****************************)
(* 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 
  ?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 li
  TR (x:'input) (CFG (x1s:'input list) (s1:'state) (out1s:'output list)) (CFG (x2s:'input 
  ?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 
   ((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
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 
      (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
      (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

```
(*************************************************************************)
(* Name: Kyle Peppe                                                       *)
(* Exercise 16.3.1                                                        *)
(* Date: 3/1/20                                                           *)
(*************************************************************************)

(* Beginning structure and open commands                      *)
structure m1Script = struct
open HolKernel Parse boolLib bossLib;
open TypeBase smTheory sminfRules

(* Set new theory                                             *)
val _ = new_theory "m1";

(* Setting the datatypes                                      *)
(* Part A                                                     *)
val _ = Datatype`command = i0 | i1`
val command_distinct_clauses = distinct_of ``:command``
(* Save the Theorem                                           *)
val _ = save_thm("command_distinct_clauses",command_distinct_clauses)

val _ = Datatype`state = S0 | S1 | S2`
val state_distinct_clauses = distinct_of ``:state``
(* Save the Theorem                                           *)
val _ = save_thm("state_distinct_clauses",state_distinct_clauses)

val _ = Datatype`output = o0 | o1`
val output_distinct_clauses = distinct_of ``:output``
(* Save the Theorem                                           *)
val _ = save_thm("output_distinct_clauses",output_distinct_clauses)

(* Part B                                                     *)
val M1ns_def =
    Define`(M1ns S0 i0 = S1) /\ (M1ns S0 i1 = S2) /\
       (M1ns S1 i0 = S0) /\ (M1ns S1 i1 = S0) /\
       (M1ns S2 i0 = S2) /\ (M1ns S2 i1 = S2)`;


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)`;
```

```
(* Part C                                                    *)
val m1TR_rules = SPEC_TR ``M1ns`` ``M1out``

(* Save the Theorem                                          *)
val _ = save_thm("m1TR_rules",m1TR_rules)

val m1TR_clauses = SPEC_TR_clauses ``M1ns`` ``M1out``

(* Save the Theorem                                          *)
val _ = save_thm("m1TR_clauses",m1TR_clauses)

val m1Trans_Equiv_TR = SPEC_Trans_Equiv_TR ``M1ns`` ``M1out``

(* Save the Theorem                                          *)
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]

(* Save the Theorem                                    *)
val _ = save_thm("m1_rules",m1_rules)

(* Print and Export theory                             *)
val _ = export_theory();
val _ = print_theory "-";

end
```

Appendix C

Source Code for counterScript.sml

```
(****************************************************************************)
(* Name: Kyle Peppe                                                         *)
(* Exercise 16.3.2                                                          *)
(* Date: 3/1/20                                                             *)
(****************************************************************************)

(* Beginning structure and open commands                    *)
structure counterScript = struct
open HolKernel Parse boolLib bossLib;
open TypeBase smTheory sminfRules

(* Set new theory                                           *)
val _ = new_theory "counter";

(* Setting the datatypes                                    *)
(* Part A                                                   *)
val _ = Datatype `ctrcmd = load num | count | hold`
val ctrcmd_distinct_clauses = distinct_of ``:ctrcmd``
(* Save the Theorem                                         *)
val _ = save_thm("ctrcmd_distinct_clauses",ctrcmd_distinct_clauses)

val _ = Datatype `ctrState = COUNT num`
val ctrState_one_one = one_one_of ``:ctrState``
(* Save the Theorem                                         *)
val _ = save_thm("ctrState_one_one",ctrState_one_one)

val _ = Datatype `ctrOut = DISPLAY num`
val ctrOut_one_one = one_one_of ``:ctrOut``
(* Save the Theorem                                         *)
val _ = save_thm("ctrOut_one_one",ctrOut_one_one)

(* Part B                                                   *)
(* Defining variables                                       *)
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))`;
```

```
(* Part C                                                    *)
val ctrTR_rules = SPEC_TR ``ctrNS`` ``ctrOut``

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

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

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

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

(* Save the Theorem                                          *)
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]

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

(* Export and print the theory                               *)
val _ = export_theory();
val _ = print_theory "-";

end
```