Homework 8

Chirag Sachdev

Week 8

Abstract

This project is a part of HW8 of Assurance Foundations. The homework deals with integration of ML and HOL to LaTeX. The goal of this report is to show reproducibility which is the groundwork for credibility that I have done this on my own without any external help. Every Chapter demonstrates the following sections:

- Problem Statement
- Relevant Code
- Test Results

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

My skills and my professional details can be found at `https://www.linkedin.in/in/chiragsachdev`.

Acknowledgments

I would gratefully acknowledge Dr. Shiu-Kai Chin and my other professors at Syracuse University and my Professors at Drexel University for being the wonderful mentors they are to guide me through my journey of obtaining a Master's Dregree.

Contents

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 13.10.1

2.1 Problem statement

Do a forward proof and a goal-oriented proof using only PROVE_TAC and goal-oriented proof using tactics in ACL.

$\vdash$ (*M*, *Oi*, *Os*) sat Name Alice says prop go ⇒
(*M*, *Oi*, *Os*) sat Name Bob says prop go ⇒
(*M*, *Oi*, *Os*) sat Name Alice meet Name Bob says prop go

2.2 Relevant Code

2.2.1 Forward Proof

```
val aclExercise1 =
let
 val th1 = ACL_ASSUM``((Name Alice) says (prop go)):(commands,staff,'d,'e)Form``
 val th2 = ACL_ASSUM``((Name Bob) says (prop go)):(commands,staff,'d,'e)Form``
 val th3 = ACL_CONJ th1 th2
 val th4 = AND_SAYS_RL th3
 val th5 = DISCH(hd(hyp th2)) th4
in
 DISCH(hd(hyp th1)) th5
end;
val _ = save_thm("aclExercise1",aclExercise1)
```

2.2.2 Goal Oriented Proof

```
val aclExercise1A =
TAC_PROOF(([],
``((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po), (Os :'e po)) sat
   Name Alice says (prop go) ==>
  (M,Oi,Os) sat Name Bob says (prop go) ==>
  (M,Oi,Os) sat (Name Alice) meet (Name Bob) says (prop go)``),
PROVE_TAC[Conjunction, And_Says_Eq])
val _ = save_thm("aclExercise1A",aclExercise1A)
```

2.2.3 Goal Oriented Proof using Tactics

```
val aclExercise1B =
TAC_PROOF(([],
```

```
``((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po), (Os :'e po)) sat
   Name Alice says (prop go) ==>
  (M,Oi,Os) sat Name Bob says (prop go) ==>
  (M,Oi,Os) sat (Name Alice) meet (Name Bob) says (prop go)``),
REPEAT STRIP_TAC THEN
ACL_AND_SAYS_RL_TAC THEN
ACL_CONJ_TAC THEN
PROVE_TAC[])
val _ = save_thm("aclExercise1B",aclExercise1B)
```

2.3 Session Transcript

2.3.1 Forward Proof

```
aclExercise1
```

1

```
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) sat
   Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Bob says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Alice meet Name Bob says
   (prop go :(commands, staff, 'd, 'e) Form)
```

2.3.2 Goal Oriented Proof

```
aclExercise1A
```

2

```
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) sat
   Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Bob says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Alice meet Name Bob says
   (prop go :(commands, staff, 'd, 'e) Form)
```

2.3.3 Goal Oriented Proof Using Tactics

```
aclExercise1B
```

3

```
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) sat
   Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Bob says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Alice meet Name Bob says
   (prop go :(commands, staff, 'd, 'e) Form)
```

Chapter 3

Excercise 13.10.2

3.1 Problem statement

Do a forward proof and a goal-oriented proof using only PROVE_TAC and goal-oriented proof using tactics in ACL.

$\vdash$ (M, Oi, Os) sat Name Alice says prop go $\Rightarrow$
(M, Oi, Os) sat Name Alice controls prop go $\Rightarrow$
(M, Oi, Os) sat prop go impf prop launch $\Rightarrow$
(M, Oi, Os) sat Name Bob says prop launch

3.2 Relevant Code

3.2.1 Forward Proof

```
val aclExercise2 =
let
 val th1 = ACL_ASSUM``((Name Alice) says (prop go)):(commands,staff,'d,'e)Form``
 val th2 = ACL_ASSUM``((Name Alice) controls (prop go)):(commands,staff,'d,'e)Form``
 val th3 = ACL_ASSUM``((prop go) impf (prop launch)):(commands,staff,'d,'e)Form``
 val th4 = CONTROLS th2 th1
 val th5 = ACL_MP th4 th3
 val th6 = SAYS ``(Name Bob):staff Princ`` th5
 val th7 = DISCH(hd(hyp th3)) th6
 val th8 = DISCH(hd(hyp th2)) th7
in
 DISCH(hd(hyp th1)) th8
end;
val _ = save_thm("aclExercise2",aclExercise2)
```

3.2.2 Goal Oriented Proof

```
val aclExercise2A =
TAC_PROOF(([],
``((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po), (Os :'e po)) sat
   Name Alice says (prop go) ==>
  (M,Oi,Os) sat Name Alice controls (prop go) ==>
  (M,Oi,Os) sat (prop go) impf (prop launch) ==>
  (M,Oi,Os) sat Name Bob says (prop launch)``),
PROVE_TAC[Controls, Modus_Ponens, Says]);
val _ = save_thm("aclExercise2A",aclExercise2A)
```

3.2.3 Goal Oriented Proof using Tactics

```
val aclExercise2B =
TAC_PROOF(([],
``((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po), (Os :'e po)) sat
   Name Alice says (prop go) ==>
  (M,Oi,Os) sat Name Alice controls (prop go) ==>
  (M,Oi,Os) sat (prop go) impf (prop launch) ==>
  (M,Oi,Os) sat Name Bob says (prop launch)``),
REPEAT STRIP_TAC THEN
ACL_SAYS_TAC THEN
PAT_ASSUM
``(M,Oi,Os) sat Name Alice says (prop go)``
(fn th1 =>
(PAT_ASSUM
``(M,Oi,Os) sat Name Alice controls (prop go)``
(fn th2 => ASSUME_TAC(CONTROLS th2 th1)))) THEN
PAT_ASSUM
``(M,Oi,Os) sat (prop go)``
(fn th1 =>
(PAT_ASSUM
``(M,Oi,Os) sat (prop go) impf (prop launch)``
(fn th2 => PROVE_TAC[(ACL_MP th1 th2)])))))
val _ = save_thm("aclExercise2B",aclExercise2B)
```

3.3 Session Transcript

3.3.1 Forward Proof

```
aclExercise2
```

4

```
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) sat
   Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Alice controls
   (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   (prop go :(commands, staff, 'd, 'e) Form) impf
   (prop launch :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Bob says (prop launch :(commands, staff, 'd, 'e) Form)
```

3.3.2 Goal Oriented Proof

```
aclExercise2A
```

5

```
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) sat
   Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Alice controls
   (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   (prop go :(commands, staff, 'd, 'e) Form) impf
   (prop launch :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Bob says (prop launch :(commands, staff, 'd, 'e) Form)
```

3.3.3 Goal Oriented Proof Using Tactics

aclExercise2B

6

```
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) sat
   Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Alice controls
   (prop go :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   (prop go :(commands, staff, 'd, 'e) Form) impf
   (prop launch :(commands, staff, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name Bob says (prop launch :(commands, staff, 'd, 'e) Form)
```

Chapter 4

Excercise 14.4.1

4.1 Problem statement

Prove the launch and abort CONOPS by proving the following theorems:

[ApRuleActivate_thm]

```
⊢ (M,Oi,Os) sat
  Name (PR (Role Operator)) controls prop launch ⇒
  (M,Oi,Os) sat
  reps (Name (PR (Staff Bob))) (Name (PR (Role Operator)))
    (prop launch) ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
  prop launch ⇒
  (M,Oi,Os) sat prop launch impf prop activate ⇒
  (M,Oi,Os) sat
  Name (Key (Role CA)) speaks_for Name (PR (Role CA)) ⇒
  (M,Oi,Os) sat
  Name (Key (Role CA)) says
  Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob)) ⇒
  (M,Oi,Os) sat
  Name (PR (Role CA)) controls
  Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob)) ⇒
  (M,Oi,Os) sat prop activate
```

[ApRuleStandDown_thm]

```
⊢ (M,Oi,Os) sat Name (PR (Role Operator)) controls prop abort ⇒
  (M,Oi,Os) sat
  reps (Name (PR (Staff Bob))) (Name (PR (Role Operator)))
    (prop abort) ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
  prop abort ⇒
  (M,Oi,Os) sat prop abort impf prop stand_down ⇒
  (M,Oi,Os) sat
  Name (Key (Role CA)) speaks_for Name (PR (Role CA)) ⇒
  (M,Oi,Os) sat
  Name (Key (Role CA)) says
  Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob)) ⇒
  (M,Oi,Os) sat
  Name (PR (Role CA)) controls
  Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob)) ⇒
  (M,Oi,Os) sat prop stand_down
```

[OpRuleAbort_thm]

```
⊢ (M,Oi,Os) sat Name (PR (Role Commander)) controls prop nogo ⇒
  (M,Oi,Os) sat
  reps (Name (PR (Staff Alice))) (Name (PR (Role Commander)))
```

```
    (prop nogo) ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) quoting
  Name (PR (Role Commander)) says prop nogo ⇒
  (M,Oi,Os) sat prop nogo impf prop abort ⇒
  (M,Oi,Os) sat
  Name (Key (Role CA)) speaks_for Name (PR (Role CA)) ⇒
  (M,Oi,Os) sat
  Name (Key (Role CA)) says
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)) ⇒
  (M,Oi,Os) sat
  Name (PR (Role CA)) controls
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)) ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
  prop abort
```

[OpRuleLaunch_thm]

```
⊢ (M,Oi,Os) sat Name (PR (Role Commander)) controls prop go ⇒
  (M,Oi,Os) sat
  reps (Name (PR (Staff Alice))) (Name (PR (Role Commander)))
    (prop go) ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) quoting
  Name (PR (Role Commander)) says prop go ⇒
  (M,Oi,Os) sat prop go impf prop launch ⇒
  (M,Oi,Os) sat
  Name (Key (Role CA)) speaks_for Name (PR (Role CA)) ⇒
  (M,Oi,Os) sat
  Name (Key (Role CA)) says
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)) ⇒
  (M,Oi,Os) sat
  Name (PR (Role CA)) controls
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)) ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
  prop launch
```

4.2 Relevant Code

4.2.1 Data types

```
val _ = Datatype `commands = go | nogo | launch | abort | activate | stand_down`

val _ = Datatype `people = Alice | Bob`

val _ = Datatype `roles = Commander | Operator | CA`

val _ = Datatype `keyPrinc = Staff people | Role roles | Ap num`

val _ = Datatype `principals = PR keyPrinc | Key keyPrinc`
```

4.2.2 Forward proof of OpRuleLaunch_thm

```
val OpRuleLaunch_thm =
let
 val th1 =
  ACL_ASSUM ``((Name (PR (Role Commander))) controls (prop go))
    : (commands,principals,'d,'e)Form``
 val th2 =
  ACL_ASSUM ``(reps(Name (PR (Staff Alice))) (Name (PR (Role Commander))) (prop go))
    : (commands,principals,'d,'e)Form``
 val th3 =
  ACL_ASSUM ``((Name (Key (Staff Alice))) quoting (Name (PR (Role Commander))) says (prop g
    : (commands,principals,'d,'e)Form``
 val th4 =
  ACL_ASSUM ``((prop go) impf (prop launch)) : (commands,principals,'d,'e)Form``
 val th5 =
  ACL_ASSUM ``((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))) : (commands,princip
 val th6 =
  ACL_ASSUM ``((Name (Key (Role CA))) says ((Name (Key (Staff Alice))) speaks_for (Name (PR
    : (commands,principals,'d,'e)Form``
 val th7 =
  ACL_ASSUM ``((Name (PR (Role CA))) controls ((Name (Key (Staff Alice))) speaks_for (Name
    : (commands,principals,'d,'e)Form``
 val th8 = SPEAKS_FOR th5 th6
 val th9 = CONTROLS th7 th8
 val th10 = IDEMP_SPEAKS_FOR ``Name (PR (Role Commander)): principals Princ``
 val th11 = INST_TYPE [``:'a`` |-> ``:commands``] th10
 val th12 = MONO_SPEAKS_FOR th9 th11
 val th13 = SPEAKS_FOR th12 th3
 val th14 = REPS th2 th13 th1
 val th15 = ACL_MP th14 th4
 val th16 = SAYS ``(Name (Key (Staff Bob))) quoting (Name (PR (Role Operator))) :principals
 val th17 = DISCH(hd(hyp th7)) th16
 val th18 = DISCH(hd(hyp th6)) th17
 val th19 = DISCH(hd(hyp th5)) th18
 val th20 = DISCH(hd(hyp th4)) th19
 val th21 = DISCH(hd(hyp th3)) th20
 val th22 = DISCH(hd(hyp th2)) th21
in
 DISCH(hd(hyp th1)) th22
end;
```

4.2.3 Proof of OpRuleAbort_thm

```
val OpRuleAbort_thm =
let
 val th1 =
  ACL_ASSUM ``((Name (PR (Role Commander))) controls (prop nogo))
    : (commands,principals,'d,'e)Form``
 val th2 =
  ACL_ASSUM ``(reps(Name (PR (Staff Alice))) (Name (PR (Role Commander))) (prop nogo))
    : (commands,principals,'d,'e)Form``
 val th3 =
```

```
  ACL_ASSUM ``((Name (Key (Staff Alice))) quoting (Name (PR (Role Commander))) says (prop r
    : (commands,principals,'d,'e)Form``
 val th4 =
  ACL_ASSUM ``((prop nogo) impf (prop abort)) : (commands,principals,'d,'e)Form``
 val th5 =
  ACL_ASSUM ``((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))) : (commands,princip
 val th6 =
  ACL_ASSUM ``((Name (Key (Role CA))) says ((Name (Key (Staff Alice))) speaks_for (Name (PR
    : (commands,principals,'d,'e)Form``
 val th7 =
  ACL_ASSUM ``((Name (PR (Role CA))) controls ((Name (Key (Staff Alice))) speaks_for (Name
    : (commands,principals,'d,'e)Form``
 val th8 = SPEAKS_FOR th5 th6
 val th9 = CONTROLS th7 th8
 val th10 = IDEMP_SPEAKS_FOR ``Name (PR (Role Commander)): principals Princ``
 val th11 = INST_TYPE [``:'a`` |-> ``:commands``] th10
 val th12 = MONO_SPEAKS_FOR th9 th11
 val th13 = SPEAKS_FOR th12 th3
 val th14 = REPS th2 th13 th1
 val th15 = ACL_MP th14 th4
 val th16 = SAYS ``(Name (Key (Staff Bob))) quoting (Name (PR (Role Operator))) :principals
 val th17 = DISCH(hd(hyp th7)) th16
 val th18 = DISCH(hd(hyp th6)) th17
 val th19 = DISCH(hd(hyp th5)) th18
 val th20 = DISCH(hd(hyp th4)) th19
 val th21 = DISCH(hd(hyp th3)) th20
 val th22 = DISCH(hd(hyp th2)) th21
in
 DISCH(hd(hyp th1)) th22
end;
```

4.2.4 Proof for ApRuleActivate_thm

```
val ApRuleActivate_thm =
let
 val th1 =
  ACL_ASSUM ``((Name (PR (Role Operator))) controls (prop launch))
    : (commands,principals,'d,'e)Form``
 val th2 =
  ACL_ASSUM ``(reps(Name (PR (Staff Bob))) (Name (PR (Role Operator))) (prop launch))
    : (commands,principals,'d,'e)Form``
 val th3 =
  ACL_ASSUM ``((Name (Key (Staff Bob))) quoting (Name (PR (Role Operator))) says (prop laun
    : (commands,principals,'d,'e)Form``
 val th4 =
  ACL_ASSUM ``((prop launch) impf (prop activate)) : (commands,principals,'d,'e)Form``
 val th5 =
  ACL_ASSUM ``((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))) : (commands,princip
 val th6 =
  ACL_ASSUM ``((Name (Key (Role CA))) says ((Name (Key (Staff Bob))) speaks_for (Name (PR
    : (commands,principals,'d,'e)Form``
 val th7 =
```

```
  ACL_ASSUM ``((Name (PR (Role CA))) controls ((Name (Key (Staff Bob))) speaks_for (Name (P
    : (commands, principals, 'd, 'e)Form``
 val th8 = SPEAKS_FOR th5 th6
 val th9 = CONTROLS th7 th8
 val th10 = IDEMP_SPEAKS_FOR ``Name (PR (Role Operator)): principals Princ``
 val th11 = INST_TYPE [``:'a`` |-> ``:commands``] th10
 val th12 = MONO_SPEAKS_FOR th9 th11
 val th13 = SPEAKS_FOR th12 th3
 val th14 = REPS th2 th13 th1
 val th15 = ACL_MP th14 th4
 val th16 = DISCH(hd(hyp th7)) th15
 val th17 = DISCH(hd(hyp th6)) th16
 val th18 = DISCH(hd(hyp th5)) th17
 val th19 = DISCH(hd(hyp th4)) th18
 val th20 = DISCH(hd(hyp th3)) th19
 val th21 = DISCH(hd(hyp th2)) th20
in
 DISCH(hd(hyp th1)) th21
end;
```

4.2.5 Proof for ApRuleStandDown_thm

```
val ApRuleStandDown_thm =
let
 val th1 =
  ACL_ASSUM ``((Name (PR (Role Operator))) controls (prop abort))
    : (commands, principals, 'd, 'e)Form``
 val th2 =
  ACL_ASSUM ``(reps(Name (PR (Staff Bob))) (Name (PR (Role Operator))) (prop abort))
    : (commands, principals, 'd, 'e)Form``
 val th3 =
  ACL_ASSUM ``((Name (Key (Staff Bob))) quoting (Name (PR (Role Operator))) says (prop abo
    : (commands, principals, 'd, 'e)Form``
 val th4 =
  ACL_ASSUM ``((prop abort) impf (prop stand_down)) : (commands, principals, 'd, 'e)Form``
 val th5 =
  ACL_ASSUM ``((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))) : (commands, princip
 val th6 =
  ACL_ASSUM ``((Name (Key (Role CA))) says ((Name (Key (Staff Bob))) speaks_for (Name (PR
    : (commands, principals, 'd, 'e)Form``
 val th7 =
  ACL_ASSUM ``((Name (PR (Role CA))) controls ((Name (Key (Staff Bob))) speaks_for (Name (P
    : (commands, principals, 'd, 'e)Form``
 val th8 = SPEAKS_FOR th5 th6
 val th9 = CONTROLS th7 th8
 val th10 = IDEMP_SPEAKS_FOR ``Name (PR (Role Operator)): principals Princ``
 val th11 = INST_TYPE [``:'a`` |-> ``:commands``] th10
 val th12 = MONO_SPEAKS_FOR th9 th11
 val th13 = SPEAKS_FOR th12 th3
 val th14 = REPS th2 th13 th1
 val th15 = ACL_MP th14 th4
 val th16 = DISCH(hd(hyp th7)) th15
```

```
 val th17 = DISCH(hd(hyp th6)) th16
 val th18 = DISCH(hd(hyp th5)) th17
 val th19 = DISCH(hd(hyp th4)) th18
 val th20 = DISCH(hd(hyp th3)) th19
 val th21 = DISCH(hd(hyp th2)) th20
in
 DISCH(hd(hyp th1)) th21
end;
```

4.3 Session Transcripts

4.3.1 Data types

7

```
  <<HOL warning: GrammarDeltas.revise_data:
   Grammar-deltas:
     overload_on("commands2num"), overload_on("num2commands"), overload_on("go"), overload_on("nogo"), overload_on("launch"), overload_on("abort"
   invalidated by NewTypeOp(scratch$commands)>>
 <<HOL message: Defined type: "commands">>
 <<HOL warning: GrammarDeltas.revise_data:
   Grammar-deltas:
     overload_on("people2num"), overload_on("num2people"), overload_on("Alice"), overload_on("Bob"), overload_on("people_size"), overload_on("peo
   invalidated by NewTypeOp(scratch$people)>>
 <<HOL message: Defined type: "people">>
 <<HOL warning: GrammarDeltas.revise_data:
   Grammar-deltas:
     overload_on("roles2num"), overload_on("num2roles"), overload_on("Commander"), overload_on("Operator"), overload_on("CA"), overload_on("roles
   invalidated by NewTypeOp(scratch$roles)>>
 <<HOL message: Defined type: "roles">>
 <<HOL warning: GrammarDeltas.revise_data:
   Grammar-deltas:
     overload_on("Ap"), overload_on("keyPrinc_size"), overload_on("PR"), overload_on("Key"), overload_on("principals_CASE"), overload_on("case")
   invalidated by NewTypeOp(scratch$keyPrinc)>>
 <<HOL message: Defined type: "keyPrinc">>
 <<HOL warning: GrammarDeltas.revise_data:
   Grammar-deltas:
     overload_on("principals_size")
   invalidated by NewTypeOp(scratch$principals)>>
 <<HOL message: Defined type: "principals">>
```

4.3.2 OpRuleLaunch

8

```
 val OpRuleLaunch_thm =
    |- ((M :(commands, 'b, principals, 'd, 'e) Kripke),(Oi :'d po),
     (Os :'e po)) sat
    Name (PR (Role Commander)) controls
    (prop go :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    reps (Name (PR (Staff Alice))) (Name (PR (Role Commander)))
      (prop go :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    Name (Key (Staff Alice)) quoting Name (PR (Role Commander)) says
    (prop go :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    (prop go :(commands, principals, 'd, 'e) Form) impf
    (prop launch :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    ((Name (Key (Role CA)) speaks_for Name (PR (Role CA)))
       :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    Name (Key (Role CA)) says
    ((Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)))
       :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    Name (PR (Role CA)) controls
    ((Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)))
       :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    Name (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
    (prop launch :(commands, principals, 'd, 'e) Form):
    thm
```

4.3.3 OpRuleAbort_thm

```
val OpRuleAbort_thm =
```

9

```
   |- ((M :(commands, 'b, principals, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) sat
   Name (PR (Role Commander)) controls
   (prop nogo :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   reps (Name (PR (Staff Alice))) (Name (PR (Role Commander)))
     (prop nogo :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name (Key (Staff Alice)) quoting Name (PR (Role Commander)) says
   (prop nogo :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   (prop nogo :(commands, principals, 'd, 'e) Form) impf
   (prop abort :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   ((Name (Key (Role CA)) speaks_for Name (PR (Role CA)))
      :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name (Key (Role CA)) says
   ((Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)))
      :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name (PR (Role CA)) controls
   ((Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)))
      :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
   (prop abort :(commands, principals, 'd, 'e) Form):
   thm
```

4.3.4 ApRuleAbort_thm

```
val ApRuleActivate_thm =
```

10

```
   |- ((M :(commands, 'b, principals, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) sat
   Name (PR (Role Operator)) controls
   (prop launch :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   reps (Name (PR (Staff Bob))) (Name (PR (Role Operator)))
     (prop launch :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
   (prop launch :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   (prop launch :(commands, principals, 'd, 'e) Form) impf
   (prop activate :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   ((Name (Key (Role CA)) speaks_for Name (PR (Role CA)))
      :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name (Key (Role CA)) says
   ((Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob)))
      :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat
   Name (PR (Role CA)) controls
   ((Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob)))
      :(commands, principals, 'd, 'e) Form) ==>
   (M,Oi,Os) sat (prop activate :(commands, principals, 'd, 'e) Form):
   thm
```

4.3.5 ApRuleStandDown_thm

```
val ApRuleStandDown_thm =
    |- ((M :(commands, 'b, principals, 'd, 'e) Kripke),(Oi :'d po),                    11
     (Os :'e po)) sat
    Name (PR (Role Operator)) controls
    (prop abort :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    reps (Name (PR (Staff Bob))) (Name (PR (Role Operator)))
      (prop abort :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    Name (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
    (prop abort :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    (prop abort :(commands, principals, 'd, 'e) Form) impf
    (prop stand_down :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    ((Name (Key (Role CA)) speaks_for Name (PR (Role CA)))
       :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    Name (Key (Role CA)) says
    ((Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob)))
       :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat
    Name (PR (Role CA)) controls
    ((Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob)))
       :(commands, principals, 'd, 'e) Form) ==>
    (M,Oi,Os) sat (prop stand_down :(commands, principals, 'd, 'e) Form):
    thm
```

Appendix A

Source code: Ex 13.10.1 and Ex 13.10.2

```
structure solutions1Theory :> solutions1Theory =
struct
  val _ = if !Globals.print_thy_loads then TextIO.print "Loading solutions1Theory ... " els
  open Type Term Thm
  infixr -->

  fun C s t ty = mk_thy_const{Name=s,Thy=t,Ty=ty}
  fun T s t A = mk_thy_type{Tyop=s, Thy=t,Args=A}
  fun V s q = mk_var(s,q)
  val U     = mk_vartype
  (* Parents and ML dependencies *)
  local open example1Theory
  in end;
  val _ = Theory.link_parents
          ("solutions1",
          Arbnum.fromString "1553033132",
          Arbnum.fromString "794835")
          [("example1",
           Arbnum.fromString "1553033126",
           Arbnum.fromString "662190")];
  val _ = Theory.incorporate_types "solutions1" [];

  val idvector =
    let fun ID(thy,oth) = {Thy = thy, Other = oth}
    in Vector.fromList
  [ID("aclfoundation", "Kripke"), ID("example1", "staff"),
   ID("example1", "commands"), ID("aclfoundation", "po"), ID("pair", ","),
   ID("min", "fun"), ID("pair", "prod"), ID("min", "==>"),
   ID("min", "bool"), ID("example1", "Alice"), ID("example1", "Bob"),
   ID("aclfoundation", "Name"), ID("aclfoundation", "Princ"),
   ID("aclfoundation", "controls"), ID("aclfoundation", "Form"),
   ID("example1", "go"), ID("aclfoundation", "impf"),
   ID("example1", "launch"), ID("aclfoundation", "meet"),
   ID("aclfoundation", "prop"), ID("aclrules", "sat"),
   ID("aclfoundation", "says")]
  end;
  local open SharingTables
  in
  val tyvector = build_type_vector idvector
  [TYV "'e", TYV "'d", TYOP [1], TYV "'b", TYOP [2],
   TYOP [0, 4, 3, 2, 1, 0], TYOP [3, 1], TYOP [3, 0], TYOP [6, 6, 7],
   TYOP [6, 5, 8], TYOP [5, 8, 9], TYOP [5, 5, 10], TYOP [5, 7, 8],
   TYOP [5, 6, 12], TYOP [8], TYOP [5, 14, 14], TYOP [5, 14, 15],
```

```
 TYOP [12, 2], TYOP [5, 2, 17], TYOP [14, 4, 2, 1, 0], TYOP [5, 19, 19],
 TYOP [5, 17, 20], TYOP [5, 19, 20], TYOP [5, 17, 17], TYOP [5, 17, 23],
 TYOP [5, 4, 19], TYOP [5, 19, 14], TYOP [5, 9, 26]]
end
val _ = Theory.incorporate_consts "solutions1" tyvector [];

local open SharingTables
in
val tmvector = build_term_vector idvector tyvector
[TMV("M", 5), TMV("Oi", 6), TMV("Os", 7), TMC(4, 11), TMC(4, 13),
 TMC(7, 16), TMC(9, 2), TMC(10, 2), TMC(11, 18), TMC(13, 21), TMC(15, 4),
 TMC(16, 22), TMC(17, 4), TMC(18, 24), TMC(19, 25), TMC(20, 27),
 TMC(21, 21)]
end
structure ThmBind = struct
  val DT = Thm.disk_thm
  val read = Term.read_raw tmvector
end
fun op aclExercise1 x = x
  val op aclExercise1 =
  ThmBind.DT((((("solutions1",0),
               [("aclDrules",[7]),("aclrules",[62]),
                ("bool",[57])]),["DISK_THM"]),
              [ThmBind.read"%5%15%3%0@%4%1@%2@3%16%8%6@2%14%10@4%5%15%3%0@%4%1@%2@3%16%8%
fun op aclExercise1A x = x
  val op aclExercise1A =
  ThmBind.DT((((("solutions1",1),
               [("aclDrules",[7]),("aclrules",[62]),
                ("bool",[25,26,46,47,52,53,62,92,93,95]),
                ("sat",[1,3,5,6,7,11,12,13,14,15])]),["DISK_THM"]),
              [ThmBind.read"%5%15%3%0@%4%1@%2@3%16%8%6@2%14%10@4%5%15%3%0@%4%1@%2@3%16%8%
fun op aclExercise1B x = x
  val op aclExercise1B =
  ThmBind.DT((((("solutions1",2),
               [("aclDrules",[7]),("aclrules",[62]),
                ("bool",[25,26,46,47,52,53,57,62]),
                ("sat",[1,3,5,6,7,11,14,15])]),["DISK_THM"]),
              [ThmBind.read"%5%15%3%0@%4%1@%2@3%16%8%6@2%14%10@4%5%15%3%0@%4%1@%2@3%16%8%
fun op aclExercise2 x = x
  val op aclExercise2 =
  ThmBind.DT((((("solutions1",3),
               [("aclDrules",[3]),("aclrules",[23,24])]),["DISK_THM"]),
              [ThmBind.read"%5%15%3%0@%4%1@%2@3%16%8%6@2%14%10@4%5%15%3%0@%4%1@%2@3%9%8%6
fun op aclExercise2A x = x
  val op aclExercise2A =
  ThmBind.DT((((("solutions1",4),
               [("aclDrules",[3]),("aclrules",[23,24]),
                ("bool",[25,26,46,47,52,53,62,92,93,95]),
                ("sat",[1,3,5,6,7,11,12,13,14,15])]),["DISK_THM"]),
              [ThmBind.read"%5%15%3%0@%4%1@%2@3%16%8%6@2%14%10@4%5%15%3%0@%4%1@%2@3%9%8%6
fun op aclExercise2B x = x
  val op aclExercise2B =
  ThmBind.DT((((("solutions1",5),
```

```
                  [("aclDrules",[3]),("aclrules",[23,24]),
                   ("bool",[25,26,46,47,52,53,62]),
                   ("sat",[1,3,5,6,7,11,14,15])]),["DISK_THM"]),
                 [ThmBind.read"%5%15%3%0@%4%1@%2@3%16%8%6@2%14%10@4%5%15%3%0@%4%1@%2@3%9%8%6
```

```
  val _ = DB.bindl "solutions1"
  [("aclExercise1",aclExercise1,DB.Thm),
   ("aclExercise1A",aclExercise1A,DB.Thm),
   ("aclExercise1B",aclExercise1B,DB.Thm),
   ("aclExercise2",aclExercise2,DB.Thm),
   ("aclExercise2A",aclExercise2A,DB.Thm),
   ("aclExercise2B",aclExercise2B,DB.Thm)]

  local open GrammarSpecials Parse
    fun UTOFF f = Feedback.trace("Parse.unicode_trace_off_complaints",0)f
  in
  val solutions1_grammars = merge_grammars ["example1"]
  local
  val (tyUDs, tmUDs) = GrammarDeltas.thy_deltas{thyname="solutions1"}
  val addtmUDs = term_grammar.add_deltas tmUDs
  val addtyUDs = type_grammar.apply_deltas tyUDs
  in
  val solutions1_grammars =
    Portable.## (addtyUDs,addtmUDs) solutions1_grammars
  val _ = Parse.grammarDB_insert("solutions1",solutions1_grammars)
  val _ = Parse.temp_set_grammars (addtyUDs (Parse.type_grammar()), addtmUDs (Parse.term_gr
  end (* addUDs local *)
  end

val _ = if !Globals.print_thy_loads then TextIO.print "done\n" else ()
val _ = Theory.load_complete "solutions1"
end
```

Appendix B

Source code: Ex 14.4.1

```
(*****************************************************************************)
(* Chirag Sachdev                                                            *)
(* Exercise: 14.4.1                                                          *)
(*****************************************************************************)

structure conops0SolutionScript = struct

open HolKernel Parse boolLib bossLib
open acl_infRules aclrulesTheory aclDrulesTheory

val _ = new_theory "conops0Solution";

val _ = Datatype `commands = go | nogo | launch | abort | activate | stand_down`

val _ = Datatype `people = Alice | Bob`

val _ = Datatype `roles = Commander | Operator | CA`

val _ = Datatype `keyPrinc = Staff people | Role roles | Ap num`

val _ = Datatype `principals = PR keyPrinc | Key keyPrinc`

val OpRuleLaunch_thm =
let
 val th1 =
  ACL_ASSUM ``((Name (PR (Role Commander))) controls (prop go))
    : (commands, principals, 'd, 'e)Form``
 val th2 =
  ACL_ASSUM ``(reps(Name (PR (Staff Alice))) (Name (PR (Role Commander))) (prop go))
    : (commands, principals, 'd, 'e)Form``
 val th3 =
  ACL_ASSUM ``((Name (Key (Staff Alice))) quoting (Name (PR (Role Commander))) says (prop g
    : (commands, principals, 'd, 'e)Form``
 val th4 =
  ACL_ASSUM ``((prop go) impf (prop launch)) : (commands, principals, 'd, 'e)Form``
 val th5 =
  ACL_ASSUM ``((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))) : (commands, princip
 val th6 =
  ACL_ASSUM ``((Name (Key (Role CA))) says ((Name (Key (Staff Alice))) speaks_for (Name (PR
    : (commands, principals, 'd, 'e)Form``
 val th7 =
  ACL_ASSUM ``((Name (PR (Role CA))) controls ((Name (Key (Staff Alice))) speaks_for (Name
```

```
    : (commands, principals, 'd, 'e)Form``
 val th8 = SPEAKS_FOR th5 th6
 val th9 = CONTROLS th7 th8
 val th10 = IDEMP_SPEAKS_FOR ``Name (PR (Role Commander)): principals Princ``
 val th11 = INST_TYPE [``:'a`` |-> ``:commands``] th10
 val th12 = MONO_SPEAKS_FOR th9 th11
 val th13 = SPEAKS_FOR th12 th3
 val th14 = REPS th2 th13 th1
 val th15 = ACL_MP th14 th4
 val th16 = SAYS ``(Name (Key (Staff Bob))) quoting (Name (PR (Role Operator))) :principals
 val th17 = DISCH(hd(hyp th7)) th16
 val th18 = DISCH(hd(hyp th6)) th17
 val th19 = DISCH(hd(hyp th5)) th18
 val th20 = DISCH(hd(hyp th4)) th19
 val th21 = DISCH(hd(hyp th3)) th20
 val th22 = DISCH(hd(hyp th2)) th21
in
 DISCH(hd(hyp th1)) th22
end;

val _ = save_thm("OpRuleLaunch_thm",OpRuleLaunch_thm)

val OpRuleAbort_thm =
let
 val th1 =
  ACL_ASSUM ``((Name (PR (Role Commander))) controls (prop nogo))
    : (commands, principals, 'd, 'e)Form``
 val th2 =
  ACL_ASSUM ``(reps(Name (PR (Staff Alice))) (Name (PR (Role Commander))) (prop nogo))
    : (commands, principals, 'd, 'e)Form``
 val th3 =
  ACL_ASSUM ``((Name (Key (Staff Alice))) quoting (Name (PR (Role Commander))) says (prop n
    : (commands, principals, 'd, 'e)Form``
 val th4 =
  ACL_ASSUM ``((prop nogo) impf (prop abort)) : (commands, principals, 'd, 'e)Form``
 val th5 =
  ACL_ASSUM ``((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))) : (commands, princip
 val th6 =
  ACL_ASSUM ``((Name (Key (Role CA))) says ((Name (Key (Staff Alice))) speaks_for (Name (PR
    : (commands, principals, 'd, 'e)Form``
 val th7 =
  ACL_ASSUM ``((Name (PR (Role CA))) controls ((Name (Key (Staff Alice))) speaks_for (Name
    : (commands, principals, 'd, 'e)Form``
 val th8 = SPEAKS_FOR th5 th6
 val th9 = CONTROLS th7 th8
 val th10 = IDEMP_SPEAKS_FOR ``Name (PR (Role Commander)): principals Princ``
 val th11 = INST_TYPE [``:'a`` |-> ``:commands``] th10
 val th12 = MONO_SPEAKS_FOR th9 th11
 val th13 = SPEAKS_FOR th12 th3
 val th14 = REPS th2 th13 th1
 val th15 = ACL_MP th14 th4
 val th16 = SAYS ``(Name (Key (Staff Bob))) quoting (Name (PR (Role Operator))) :principals
 val th17 = DISCH(hd(hyp th7)) th16
```

```
 val th18 = DISCH(hd(hyp th6)) th17
 val th19 = DISCH(hd(hyp th5)) th18
 val th20 = DISCH(hd(hyp th4)) th19
 val th21 = DISCH(hd(hyp th3)) th20
 val th22 = DISCH(hd(hyp th2)) th21
in
 DISCH(hd(hyp th1)) th22
end;

val _ = save_thm("OpRuleAbort_thm",OpRuleAbort_thm)

val ApRuleActivate_thm =
let
 val th1 =
  ACL_ASSUM ``((Name (PR (Role Operator))) controls (prop launch))
    : (commands,principals,'d,'e)Form``
 val th2 =
  ACL_ASSUM ``(reps(Name (PR (Staff Bob))) (Name (PR (Role Operator))) (prop launch))
    : (commands,principals,'d,'e)Form``
 val th3 =
  ACL_ASSUM ``((Name (Key (Staff Bob))) quoting (Name (PR (Role Operator))) says (prop laun
    : (commands,principals,'d,'e)Form``
 val th4 =
  ACL_ASSUM ``((prop launch) impf (prop activate)) : (commands,principals,'d,'e)Form``
 val th5 =
  ACL_ASSUM ``((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))) : (commands,princip
 val th6 =
  ACL_ASSUM ``((Name (Key (Role CA))) says ((Name (Key (Staff Bob))) speaks_for (Name (PR
    : (commands,principals,'d,'e)Form``
 val th7 =
  ACL_ASSUM ``((Name (PR (Role CA))) controls ((Name (Key (Staff Bob))) speaks_for (Name (P
    : (commands,principals,'d,'e)Form``
 val th8 = SPEAKS_FOR th5 th6
 val th9 = CONTROLS th7 th8
 val th10 = IDEMP_SPEAKS_FOR ``Name (PR (Role Operator)): principals Princ``
 val th11 = INST_TYPE [``:'a`` |-> ``:commands``] th10
 val th12 = MONO_SPEAKS_FOR th9 th11
 val th13 = SPEAKS_FOR th12 th3
 val th14 = REPS th2 th13 th1
 val th15 = ACL_MP th14 th4
 val th16 = DISCH(hd(hyp th7)) th15
 val th17 = DISCH(hd(hyp th6)) th16
 val th18 = DISCH(hd(hyp th5)) th17
 val th19 = DISCH(hd(hyp th4)) th18
 val th20 = DISCH(hd(hyp th3)) th19
 val th21 = DISCH(hd(hyp th2)) th20
in
 DISCH(hd(hyp th1)) th21
end;

val _ = save_thm("ApRuleActivate_thm",ApRuleActivate_thm)

val ApRuleStandDown_thm =
```

```
let
 val th1 =
  ACL_ASSUM ``((Name (PR (Role Operator))) controls (prop abort))
     : (commands,principals,'d,'e)Form``
 val th2 =
  ACL_ASSUM ``(reps(Name (PR (Staff Bob))) (Name (PR (Role Operator))) (prop abort))
     : (commands,principals,'d,'e)Form``
 val th3 =
  ACL_ASSUM ``((Name (Key (Staff Bob))) quoting (Name (PR (Role Operator))) says (prop abo
     : (commands,principals,'d,'e)Form``
 val th4 =
  ACL_ASSUM ``((prop abort) impf (prop stand_down)) : (commands,principals,'d,'e)Form``
 val th5 =
  ACL_ASSUM ``((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))) : (commands,princi
 val th6 =
  ACL_ASSUM ``((Name (Key (Role CA))) says ((Name (Key (Staff Bob))) speaks_for (Name (PR
     : (commands,principals,'d,'e)Form``
 val th7 =
  ACL_ASSUM ``((Name (PR (Role CA))) controls ((Name (Key (Staff Bob))) speaks_for (Name (
     : (commands,principals,'d,'e)Form``
 val th8 = SPEAKS_FOR th5 th6
 val th9 = CONTROLS th7 th8
 val th10 = IDEMP_SPEAKS_FOR ``Name (PR (Role Operator)): principals Princ``
 val th11 = INST_TYPE [``:'a`` |-> ``:commands``] th10
 val th12 = MONO_SPEAKS_FOR th9 th11
 val th13 = SPEAKS_FOR th12 th3
 val th14 = REPS th2 th13 th1
 val th15 = ACL_MP th14 th4
 val th16 = DISCH(hd(hyp th7)) th15
 val th17 = DISCH(hd(hyp th6)) th16
 val th18 = DISCH(hd(hyp th5)) th17
 val th19 = DISCH(hd(hyp th4)) th18
 val th20 = DISCH(hd(hyp th3)) th19
 val th21 = DISCH(hd(hyp th2)) th20
in
 DISCH(hd(hyp th1)) th21
end;

val _ = save_thm("ApRuleStandDown_thm",ApRuleStandDown_thm)

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