

# Project 6

Bharath Karumudi

August 20, 2019

## **Abstract**

This project is to demonstrate the capabilities of implementing constructing and deconstructing HOL Terms using the tools and techniques - L<sup>A</sup>T<sub>E</sub>X, AcuTeX, emacs and ML.

Each chapter documents the given problems with a structure of:

1. Problem Statement
2. Relevant Code
3. Execution Transcripts
4. Explanation of results

**Acknowledgments:** Professor Marvine Hamner and Professor Shiu-Kai Chin who taught the Certified Security By Design.

---

# Contents

---

<b>1</b>	<b>Executive Summary</b>	<b>3</b>
<b>2</b>	<b>Exercise 13.10.1</b>	<b>5</b>
2.1	Problem Statement . . . . .	5
2.2	Relevant Code . . . . .	5
2.3	Execution Transcripts . . . . .	7
2.3.1	Explanation of Results . . . . .	7
<b>3</b>	<b>Exercise 13.10.2</b>	<b>8</b>
3.1	Problem Statement . . . . .	8
3.2	Relevant Code . . . . .	8
3.3	Execution Transcripts . . . . .	10
3.3.1	Explanation of Results . . . . .	10
<b>4</b>	<b>Exercise 14.4.1</b>	<b>11</b>
4.1	Problem Statement . . . . .	11
4.2	Relevant Code . . . . .	12
4.3	Execution Transcripts . . . . .	17
4.3.1	Explanation of Results . . . . .	18
<b>5</b>	<b>Appendix A: Exercise 13 - example1Theory</b>	<b>19</b>
<b>6</b>	<b>Appendix B: Exercise 13 - solutions1Theory</b>	<b>21</b>
<b>7</b>	<b>Appendix B: Exercise 14 - conops0SolutionTheory</b>	<b>25</b>

# Executive Summary

---

All requirements for this project are statisfied specifically, and by using HOL proved the below theorems:

### Exercise 13:

```
⊢ (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  
  
⊢ (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  
  
⊢ (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  
  
⊢ (M, Oi, Os) sat Name Bob says prop launch  
  
⊢ (M, Oi, Os) 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  
  
⊢ (M, Oi, Os) 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
```

### Exercise 14:

```
commands = go | nogo | launch | abort | activate | stand_down  
  
⊢ (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

⊢ (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 (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
prop abort

⊢ (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

⊢ (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

```

---

## Exercise 13.10.1

---

### 2.1 Problem Statement

```
⊢ (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

⊢ (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

⊢ (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

```
(*****
(* Exercise 13.10.1 *)
*****)

(*****
(* Part A: Forward proof *)
*****)

val aclExercise1=
let
val th1 = ACLASSUM term1;
val th2 = ACLASSUM term2;
val th3 = ACLCONJ 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)

(*****
(* Part B: Goal-oriented proof *)
*****)

val aclExercise1A=
```

---

```

TACPROOF(
  ([],
  ‘‘(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)

(*****
(* Part C: Goal-oriented proof *)
*****)

val aclExercise1B=
TACPROOF(([],
  ‘‘(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[] THEN
  PROVE_TAC[])

val _ = save_thm("aclExercise1B", aclExercise1B)

```

---



## 2.3 Execution Transcripts

```

-----
HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]
-----
For introductory HOL help, type: help "hol";
To exit type <Control>-D
-----
> > > # # # # # ** types trace now on
> # # # # # ** Unicode trace now off
> val princTerm =
  'Name Alice':
  term
> val term1 =
  'Name Alice says (prop go :(commands, staff, 'd, 'e) Form)':
  term
> val term2 =
  'Name Bob says (prop go :(commands, staff, 'd, 'e) Form)':
  term
> # val term3 =
  'Name Alice meet Name Bob says
  (prop launch :(commands, staff, 'd, 'e) Form)':
  term
> # # # # # val aclExercise1 =
  []
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(O1 :'d po),
    (Os :'e po)) sat
Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
(M,O1,Os) sat
Name Bob says (prop go :(commands, staff, 'd, 'e) Form) ==>
(M,O1,Os) sat
Name Alice meet Name Bob says
(prop go :(commands, staff, 'd, 'e) Form):
thm
> # # # # # Meson search level: ....
val aclExercise1A =
  []
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(O1 :'d po),
    (Os :'e po)) sat
Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
(M,O1,Os) sat
Name Bob says (prop go :(commands, staff, 'd, 'e) Form) ==>
(M,O1,Os) sat
Name Alice meet Name Bob says
(prop go :(commands, staff, 'd, 'e) Form):
thm
> # # # # # Meson search level: ..
Meson search level: ..
val aclExercise1B =
  []
|- ((M :(commands, 'b, staff, 'd, 'e) Kripke),(O1 :'d po),
    (Os :'e po)) sat
Name Alice says (prop go :(commands, staff, 'd, 'e) Form) ==>
(M,O1,Os) sat
Name Bob says (prop go :(commands, staff, 'd, 'e) Form) ==>
(M,O1,Os) sat
Name Alice meet Name Bob says
(prop go :(commands, staff, 'd, 'e) Form):
thm
>
*** Emacs/HOL command completed ***

```

### 2.3.1 Explanation of Results

The above results shows that the requirements are satisfied.

## Exercise 13.10.2

### 3.1 Problem Statement

In this exercise we need to prove the theorem:

```

⊢ (M, Oi, Os) sat Name Bob says prop launch

⊢ (M, Oi, Os) 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

⊢ (M, Oi, Os) 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

```

### 3.2 Relevant Code

```

(*****
(* Exercise 13.10.2
*****)

(*****
(* Part A: Forward proof
*****)

val term1 = “((Name Alice) says (prop go)):(commands, staff, 'd, 'e)Form ‘ ‘;
val term2 = “((Name Alice) controls (prop go)):(commands, staff, 'd, 'e)Form ‘ ‘;
val term3 = “((prop go) impf (prop launch)):(commands, staff, 'd, 'e)Form ‘ ‘;
val term4 = “((Name Bob) says (prop launch)):(commands, staff, 'd, 'e)Form ‘ ‘;

val aclExercise2 =
let
val thm1 = ACLASSUM term1
val thm2 = ACLASSUM term2
val thm3 = ACLASSUM term3
val thm4 = ACLASSUM term4

val thm5 = CONTROLS thm2 thm1
val thm6 = ACLMP thm5 thm3
in
SAYS ‘ ‘Name Bob‘ ‘ thm6
end;

```

---

```

val _ = save_thm("aclExercise2", aclExercise2)

(*****
(* Part B: Goal-oriented proof
*)
*****)

val aclExercise2A=
TACPROOF([
  ‘‘(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)

(*****
(* Part C: Goal-oriented proof
*)
*****)

val aclExercise2B =
TACPROOF([
  ‘‘(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
  ACLSAYS_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 th3 =>
  (PAT_ASSUM
  ‘‘(M,Oi,Os) sat (prop go) impf (prop launch) ‘‘
  (fn th4 => ASSUME_TAC(ACLMP th3 th4)))))) THEN
  ASM_REWRITE_TAC[])

val _ = save_thm("aclExercise2B",aclExercise2B)

```

---

### 3.3 Execution Transcripts

```

-----
HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]
-----
For introductory HOL help, type: help "hol";
To exit type <Control>-D
-----
> > > # # # # # ** types trace now on
> # # # # # ** Unicode trace now off
> val term1 =
  ``Name Alice says (prop go :(commands, staff, 'd, 'e) Form)``:
  term
> val term2 =
  ``Name Alice controls (prop go :(commands, staff, 'd, 'e) Form)``:
  term
> val term3 =
  ``(prop go :(commands, staff, 'd, 'e) Form) impf
  (prop launch :(commands, staff, 'd, 'e) Form)``:
  term
> val term4 =
  ``Name Bob says (prop launch :(commands, staff, 'd, 'e) Form)``:
  term
>
*** Emacs/HOL command completed ***

> # # # # # val aclExercise2 =

[(M :(commands, 'b, staff, 'd, 'e) Kripke), (Oi : 'd po), (Os : 'e po)) sat
Name Alice controls (prop go :(commands, staff, 'd, 'e) Form),
(M :(commands, 'b, staff, 'd, 'e) Kripke), (Oi : 'd po), (Os : 'e po)) sat
(prop go :(commands, staff, 'd, 'e) Form) impf
(prop launch :(commands, staff, 'd, 'e) Form),
(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 :(commands, 'b, staff, 'd, 'e) Kripke), (Oi : 'd po),
    (Os : 'e po)) sat
    Name Bob says (prop launch :(commands, staff, 'd, 'e) Form):
    thm
> # # # # # Meson search level: .....
val aclExercise2A =
  []
|- ((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):
    thm
>
*** Emacs/HOL command completed ***

val aclExercise2B =
  []
|- ((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):
    thm
val it = (): unit
>
*** Emacs/HOL command completed ***

```

#### 3.3.1 Explanation of Results

The above results shows that the requirements are satisfied.

# Exercise 14.4.1

## 4.1 Problem Statement

In this exercise we need to prove the following:

```

commands = go | nogo | launch | abort | activate | stand_down

⊢ (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

⊢ (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 (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
  prop abort

⊢ (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

⊢ (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

```

## 4.2 Relevant Code

```

(*****)
(* Exercise: 14.4.1 *)
(* Author:   Bharath Karumudi *)
(*          Prof. Shiu-Kai Chin (Ref: Async) *)
(* Date:     Aug 16 2019 *)
(*****)

structure conops0SolutionScript = struct

open HolKernel boolLib Parse bossLib;
open acl_infRules acrulesTheory aclDrulesTheory

val _ = new_theory "conops0Solution";

(*****)
(* Defining instructions – go, nogo, launch, abort, activate, stand_down *)
(*****)
val _ =
Hol_datatype
‘commands = go | nogo | launch | abort | activate | stand_down‘

```

---

```

(*****)
(* Defining Principals - Alice and Bob *)
(*****)
val _ =
Hol_datatype
‘people = Alice | Bob‘

(*****)
(* Define roles *)
(*****)
val _ =
Hol_datatype
‘roles = Commander | Operator | CA‘

(*****)
(* Define principals that will have keys *)
(*****)
val _ =
Hol_datatype
‘keyPrinc = Staff of people | Role of roles | Ap of num‘

(*****)
(* Define principals as keyPrinc and keys *)
(*****)
val _ =
Hol_datatype
‘principals = PR of keyPrinc | Key of keyPrinc ‘

(*****)
(* Proof for OpRuleLaunch *)
(*****)
val OpRuleLaunch_thm =
let
  val th1 = ACLASSUM ‘((Name (PR(Role Commander))) controls (prop go)):(commands, principals)
  val th2 = ACLASSUM ‘(reps (Name(PR (Staff Alice)))(Name(PR(Role Commander))) (prop go)):(c
  val th3 = ACLASSUM ‘((Name(Key(Staff Alice))) quoting (Name((PR(Role Commander)))) says c
  val th4 = ACLASSUM ‘((prop go) impf (prop launch)):(commands, principals ,’d,’e)Form‘ ‘
  val th5 = ACLASSUM ‘((Name(Key(Role CA))) speaks_for (Name(PR(Role CA)))):(commands, prin
  val th6 = ACLASSUM ‘((Name(Key(Role CA))) says ((Name(Key(Staff Alice))) speaks_for (Name
  val th7 = ACLASSUM ‘((Name(PR(Role CA))) controls ((Name(Key(Staff Alice))) speaks_for (N
  val th8 = SPEAKS_FOR th5 th6
  val th9 = CONTROLS th7 th8
  val th10 = IDEMP_SPEAKS_FOR ‘(Name((PR(Role Commander)))) ‘ ‘
  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 = ACLMP th14 th4
  val th16 = SAYS ‘((Name(Key(Staff Bob))) quoting (Name(PR(Role Operator)))): principals P
  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)

```

```

(*****
(* Proof for OpRuleAbort
*****)
val OpRuleAbort_thm =
let
  val th1 = ACLASSUM '((Name (PR(Role Commander))) controls (prop nego)): (commands, principals)
  val th2 = ACLASSUM ' (reps (Name (PR (Staff Alice))) (Name (PR (Role Commander))) (prop nego))
  val th3 = ACLASSUM ' ((Name (Key (Staff Alice))) quoting (Name (PR (Role Commander)))) says
  val th4 = ACLASSUM ' ((prop nego) impf (prop abort)): (commands, principals, 'd, 'e) Form
  val th5 = ACLASSUM ' ((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))): (commands, principals)
  val th6 = ACLASSUM ' ((Name (Key (Role CA))) says ((Name (Key (Staff Alice))) speaks_for (Name
  val th7 = ACLASSUM ' ((Name (PR (Role CA))) controls ((Name (Key (Staff Alice))) speaks_for (Name
  val th8 = SPEAKS_FOR th5 th6
  val th9 = CONTROLS th7 th8
  val th10 = IDEMP_SPEAKS_FOR ' (Name (PR (Role Commander))) '
  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 = ACLMP th14 th4
  val th16 = SAYS ' ((Name (Key (Staff Bob))) quoting (Name (PR (Role Operator)))): principals
  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("OpRuleAbort_thm", OpRuleAbort_thm)

```

```

(*****
(* Proof for ApRuleActivate_thm
*****)
val ApRuleActivate_thm =
let
  val th1 = ACLASSUM ' ((Name (PR (Role Operator))) controls (prop launch)): (commands, principals)
  val th2 = ACLASSUM ' (reps (Name (PR (Staff Bob))) (Name (PR (Role Operator))) (prop launch)):
  val th3 = ACLASSUM ' ((Name (Key (Staff Bob))) quoting (Name (PR (Role Operator)))) says (pro

```



---

```

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, principals, 'd, 'e)Form
val th6 = ACL_ASSUM ' ((Name(Key(Role CA))) says ((Name(Key(Staff Bob)) speaks_for (Name(PR(Role CA)))):(commands, principals, 'd, 'e)Form
val th7 = ACL_ASSUM ' ((Name(PR(Role CA))) controls ((Name(Key(Staff Bob)) speaks_for (Name(PR(Role CA)))):(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)))) ' '
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)

```

```

(*****
(* 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 abort)):(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, principals, 'd, 'e)Form
  val th6 = ACL_ASSUM ' ((Name(Key(Role CA))) says ((Name(Key(Staff Bob)) speaks_for (Name(PR(Role CA)))):(commands, principals, 'd, 'e)Form
  val th7 = ACL_ASSUM ' ((Name(PR(Role CA))) controls ((Name(Key(Staff Bob)) speaks_for (Name(PR(Role CA)))):(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)))) ' '
  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;

```

---

```

end;

val _ = save_thm("ApRuleStandDown_thm", ApRuleStandDown_thm)

(*****)
(* Exporting Theory *)
(*****)

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

end (* structure *)

```

## 4.3 Execution Transcripts

```

-----
HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]
-----
For introductory HOL help, type: help "hol";
To exit type <Control>-D
-----
> # # <<HOL message: Defined type: "commands">>
> # # <<HOL message: Defined type: "people">>
> # # <<HOL message: Defined type: "roles">>
> # # <<HOL message: Defined type: "keyPrinc">>
> # # <<HOL message: Defined type: "principals">>
> val 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:
  thm
val it = (): unit
>
*** Emacs/HOL command completed ***

> val OpRuleAbort_thm =

[(M,Oi,Os) sat
  Name (PR (Role CA)) controls
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))]
|- (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 (Key (Staff Bob)) quoting Name (PR (Role Operator)) says
  prop abort:
  thm
val it = (): unit
>
*** Emacs/HOL command completed ***

```

```

> val 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:
  thm
val it = (): unit
>
*** Emacs/HOL command completed ***

> val 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:
  thm
val it = (): unit
>
*** Emacs/HOL command completed ***

```

### 4.3.1 Explanation of Results

The above results shows that the requirements are satisfied.

## Appendix A: Exercise 13 - example1Theory

The following code is from the file example1Script.sml

```
(*****)
(*   File: example1Script.sml                               *)
(*   Author: Bharath Karumudi                               *)
(*           Prof. Shiu-Kai Chin (Ref. Async)                *)
(*   Date: Aug 16, 2019                                       *)
(*****)
```

```
structure example1Script = struct
```

```
open HolKernel boolLib Parse bossLib (* used by Holmake, not in interactive *)
open acl_infRules aclrulesTheory aclDrulesTheory (* used by Holmake and interactive mode *)
```

```
(*****)
(* Create New Theory                                           *)
(*****)
```

```
val _ = new_theory "example1";
val _ =
  Datatype
  'commands = go | nogo | launch | abort'
```

```
(*****)
(* Define some names of people who will be principals *)
(*****)
```

```
val _ =
  Datatype
  'staff = Alice | Bob | Carol | Dan'
```

```
val commandProp = '(prop go):(commands,staff,'d','e)Form'';
val xProposition = '(prop x):('a,'c,'d,'e)Form''
val x = 'x:( 'a,'c,'d,'e)Form''
val princTerm = 'Name Alice'';
val term1 = '((Name Alice) says (prop go)):(commands,staff,'d','e)Form'';
val term2 = '((Name Alice) controls (prop go)):(commands,staff,'d','e)Form'';
val term3 =
  '((Name Alice) meet (Name Bob) says (prop launch)):(commands,staff,'d','e)Form'';
val term4 =
  '((Name Carol) quoting (Name Dan) says (prop nogo)):(commands,staff,'d','e)Form'';
val term5 =
  '((Name Dan) speaks_for (Name Carol)):(commands,staff,'d','e)Form'';
```

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

---

## Appendix B: Exercise 13 - solutions1Theory

---

The following code is from the file solutions1Script.sml

```
(*****)
(* Exercise: Chapter 13 *)
(* Author: Bharath Karumudi *)
(* Date: Aug 16 2019 *)
(*****)

structure solutions1Script = struct

open HolKernel boolLib Parse bossLib;
open acl_infRules acrulesTheory aclDrulesTheory example1Theory

val _ = new_theory "solutions1";

(* ===== Commented to avoid duplicate =====
val _ =
Datatype
'commands = go | nogo | launch | abort '

(*****)
(* Define some names of people who will be principals *)
(*****)
val _ =
Datatype
'staff = Alice | Bob '
===== Comment End ===== *)

val princTerm = 'Name Alice ' ;

(* Principals make statements *)
val term1 = '((Name Alice) says (prop go)):(commands, staff, 'd, 'e)Form ' ;

(* Principals make statements *)
val term2 = '((Name Bob) says (prop go)):(commands, staff, 'd, 'e)Form ' ;

(* Alice with Bob says <go> *)
val term3 =
'((Name Alice) meet (Name Bob) says (prop launch)):(commands, staff, 'd, 'e)Form ' ;

(*****)
```

---

```

(* Exercise 13.10.1 *)
(*****)

(*****)
(* Part A: Forward proof *)
(*****)

val aclExercise1=
let
val th1 = ACLASSUM term1;
val th2 = ACLASSUM term2;
val th3 = ACLCONJ 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)

(*****)
(* Part B: 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)

(*****)
(* Part C: Goal-oriented proof *)
(*****)

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[] THEN
PROVE_TAC[]))

```

---



---

```
val _ = save_thm("aclExercise1B", aclExercise1B)
```

```
(*****  
(* Exercise 13.10.2 *)  
(*****)
```

```
(*****  
(* Part A: Forward proof *)  
(*****)
```

```
val term1 = ‘‘((Name Alice) says (prop go)):(commands, staff, 'd, 'e)Form ‘‘;  
val term2 = ‘‘((Name Alice) controls (prop go)):(commands, staff, 'd, 'e)Form ‘‘;  
val term3 = ‘‘((prop go) impf (prop launch)):(commands, staff, 'd, 'e)Form ‘‘;  
val term4 = ‘‘((Name Bob) says (prop launch)):(commands, staff, 'd, 'e)Form ‘‘;
```

```
val aclExercise2 =  
let  
val thm1 = ACLASSUM term1  
val thm2 = ACLASSUM term2  
val thm3 = ACLASSUM term3  
val thm4 = ACLASSUM term4
```

```
val thm5 = CONTROLS thm2 thm1  
val thm6 = ACLMP thm5 thm3  
in  
SAYS ‘‘Name Bob‘‘ thm6  
end;
```

```
val _ = save_thm("aclExercise2", aclExercise2)
```

```
(*****  
(* Part B: Goal-oriented proof *)  
(*****)
```

```
val aclExercise2A =  
TACPROOF([  
‘‘((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)
```

```
(*****  
(* Part C: Goal-oriented proof *)  
(*****)
```

---

```

(*****)

val aclExercise2B =
TACPROOF([
  ‘‘(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 th3 =>
      (PAT_ASSUM
        ‘‘(M,Oi,Os) sat (prop go) impf (prop launch)‘‘
        (fn th4 => ASSUME_TAC(ACLMP th3 th4)))))) THEN
  ASM_REWRITE_TAC[])

val _ = save_thm("aclExercise2B",aclExercise2B)

(*****)
(* Exporting Theory *)
(*****)
val _ = print_theory "-";
val _ = export_theory();

end (* Structure *)

```

# Appendix B: Exercise 14 - conops0SolutionTheory

The following code is from the file conops0SolutionScript.sml

```
(*****)
(* Exercise: 14.4.1 *)
(* Author:   Bharath Karumudi *)
(*          Prof. Shiu-Kai Chin (Ref: Async) *)
(* Date:     Aug 16 2019 *)
(*****)

structure conops0SolutionScript = struct

open HolKernel boolLib Parse bossLib;
open acl_infRules aclrulesTheory aclDrulesTheory

val _ = new_theory "conops0Solution";

(*****)
(* Defining instructions – go, nogo, launch, abort, activate, stand_down *)
(*****)
val _ =
Hol_datatype
‘commands = go | nogo | launch | abort | activate | stand_down‘

(*****)
(* Defining Principals – Alice and Bob *)
(*****)
val _ =
Hol_datatype
‘people = Alice | Bob‘

(*****)
(* Define roles *)
(*****)
val _ =
Hol_datatype
‘roles = Commander | Operator | CA‘

(*****)
(* Define principals that will have keys *)
(*****)
val _ =
```

---

```

Hol_datatype
'keyPrinc = Staff of people | Role of roles | Ap of num'

(* ***** *)
(* Define principals as keyPrinc and keys *)
(* ***** *)
val _ =
Hol_datatype
'principals = PR of keyPrinc | Key of keyPrinc'

(* ***** *)
(* Proof for OpRuleLaunch *)
(* ***** *)
val OpRuleLaunch_thm =
let
  val th1 = ACLASSUM '((Name (PR (Role Commander))) controls (prop go)):(commands, principals)
  val th2 = ACLASSUM ' (reps (Name (PR (Staff Alice))) (Name (PR (Role Commander))) (prop go)):(c
  val th3 = ACLASSUM '((Name (Key (Staff Alice))) quoting (Name (PR (Role Commander)))) says (
  val th4 = ACLASSUM '((prop go) impf (prop launch)):(commands, principals, 'd, 'e)Form '
  val th5 = ACLASSUM '((Name (Key (Role CA))) speaks_for (Name (PR (Role CA)))):(commands, prin
  val th6 = ACLASSUM '((Name (Key (Role CA))) says ((Name (Key (Staff Alice))) speaks_for (Name
  val th7 = ACLASSUM '((Name (PR (Role CA))) controls ((Name (Key (Staff Alice))) speaks_for (N
  val th8 = SPEAKS_FOR th5 th6
  val th9 = CONTROLS th7 th8
  val th10 = IDEMP_SPEAKS_FOR ' (Name (PR (Role Commander))) '
  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 = ACLMP th14 th4
  val th16 = SAYS ' ((Name (Key (Staff Bob))) quoting (Name (PR (Role Operator)))): principals P
  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)

(* ***** *)
(* Proof for OpRuleAbort *)
(* ***** *)
val OpRuleAbort_thm =
let
  val th1 = ACLASSUM '((Name (PR (Role Commander))) controls (prop nogo)):(commands, principals)
  val th2 = ACLASSUM ' (reps (Name (PR (Staff Alice))) (Name (PR (Role Commander))) (prop nogo))

```

---

---

```

val th3 = ACL_ASSUM ' ((Name(Key(Staff Alice))) quoting (Name((PR(Role Commander)))) says (
val th4 = ACL_ASSUM ' ((prop nego) impf (prop abort)):(commands, principals, 'd, 'e)Form '
val th5 = ACL_ASSUM ' ((Name(Key(Role CA))) speaks_for (Name(PR(Role CA)))):(commands, prin
val th6 = ACL_ASSUM ' ((Name(Key(Role CA))) says ((Name(Key(Staff Alice)) speaks_for (Name
val th7 = ACL_ASSUM ' ((Name(PR(Role CA))) controls ((Name(Key(Staff Alice)) speaks_for (N
val th8 = SPEAKS_FOR th5 th6
val th9 = CONTROLS th7 th8
val th10 = IDEMP_SPEAKS_FOR ' ((Name((PR(Role Commander)))) '
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 P
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("OpRuleAbort_thm", OpRuleAbort_thm)

```

```

(*****
(* Proof for ApRuleActivate_thm *)
(*****)
val ApRuleActivate_thm =
let
  val th1 = ACL_ASSUM ' ((Name (PR(Role Operator))) controls (prop launch)):(commands, princip
  val th2 = ACL_ASSUM ' (reps(Name(PR (Staff Bob)))(Name(PR(Role Operator))) (prop launch)):(
  val th3 = ACL_ASSUM ' ((Name(Key(Staff Bob))) quoting (Name((PR(Role Operator)))) says (pro
  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, prin
  val th6 = ACL_ASSUM ' ((Name(Key(Role CA))) says ((Name(Key(Staff Bob)) speaks_for (Name(PR
  val th7 = ACL_ASSUM ' ((Name(PR(Role CA))) controls ((Name(Key(Staff Bob)) speaks_for (Nam
  val th8 = SPEAKS_FOR th5 th6
  val th9 = CONTROLS th7 th8
  val th10 = IDEMP_SPEAKS_FOR ' ((Name((PR(Role Operator)))) '
  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)

```

```

(* ***** *)
(* Proof for ApRuleStandDown_thm *)
(* ***** *)
val ApRuleStandDown_thm =
let
  val th1 = ACLASSUM' '((Name (PR(Role Operator))) controls (prop abort)):(commands, principals, 'd, 'e)Form
  val th2 = ACLASSUM' ' (reps (Name(PR (Staff Bob)))(Name(PR(Role Operator))) (prop abort)):(commands, principals, 'd, 'e)Form
  val th3 = ACLASSUM' ' ((Name(Key(Staff Bob))) quoting (Name((PR(Role Operator)))) says (prop abort)):(commands, principals, 'd, 'e)Form
  val th4 = ACLASSUM' ' ((prop abort) impf (prop stand_down)):(commands, principals, 'd, 'e)Form
  val th5 = ACLASSUM' ' ((Name(Key(Role CA))) speaks_for (Name(PR(Role CA)))):(commands, principals, 'd, 'e)Form
  val th6 = ACLASSUM' ' ((Name(Key(Role CA))) says ((Name(Key(Staff Bob)) speaks_for (Name(PR(Role CA)))))):(commands, principals, 'd, 'e)Form
  val th7 = ACLASSUM' ' ((Name(PR(Role CA))) controls ((Name(Key(Staff Bob)) speaks_for (Name(PR(Role CA)))))):(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)))) ' '
  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 = ACLMP 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)

```

```

(* ***** *)
(* Exporting Theory *)
(* ***** *)

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

end (* structure *)

```