Project 6 Report

Alfred Murabito

February 23, 2020

Abstract

This report details results for the following exercises from *Certified Security by Design Using Higher Order Logic*: 13.10.1, 13.10.2, 14.4.1. In these exercises, we defined new theorems using the Access Control logic definitions and theorems. In Ch 14, we build a sample Conops implemented with four theorems.

Acknowledgments: I received no assistance with this exercise.

Contents

1 Executive Summary

All requirements for this assignment have been satisfied. A description of each exercise is given in the corresponding sections of this report. These exercises focus on the material presented in Chapter 13 and 14 of the textbook, *Certified Security by Design Using Higher Order Logic*. In addition, pretty-printing was used for appropriate portions of this report.

Chapter 13

2 Exercise 13.10.1 Problem Statement

In this exercise we extend upon the *example1Theory* with a new theory *solutions1Theory*. It uses the definitions from example1Theory to build an inference rule represented by the theorem below.

```
⊢ (M,Oi,Os) sat Name Bob says prop go ⇒
  (M,Oi,Os) sat Name Alice says prop go ⇒
  (M,Oi,Os) sat Name Alice meet Name Bob says prop go
```

The inference rule is first proved using a forward proof by using the access control logic inference rules on the assumptions. Then *PROVE_TAC* is used alone with the *SPEC_ALL Conjunction* and *GSYM(SPEC_ALL And_Says_Eq)* theorems. Finally, we uses a combination of other tactics and *PROVE_TAC*.

3 Exercise 13.10.2 Problem Statement

This exercise further expands on *solutions1Theory* with the inference rule described with the theorem.

```
⊢ (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
```

We prove this theorem three consecutvie times as well using similar approaches to the problem statement above. The forward proof used ACL Inference rules *ACL_ASSUM CONTROLS ACL_MP SAYS*, to prove the theorem. The PROVE_TAC solved the theorem using the corresponding theorems for the above inference rules. And finally, it was proved using *PAT_ASSUM* and other tactics.

4 Chapter 13 Execution Transcript

```
-------------------------------------------------------------------
       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
-------------------------------------------------------------------
[extending loadPath with Holmakefile INCLUDES variable]
> > > > > > > > Loading example1Theory
> Loading acl_infRules

> > > > <<HOL message: Created theory "solutions1">>
Meson search level: .....
Meson search level: ..
Meson search level: ..
Meson search level: .......

Theory: solutions1

Parents:
    example1

Theorems:
    aclExercise1
      |- (M,Oi,Os) sat Name Bob says prop go
         (M,Oi,Os) sat Name Alice says prop go
         (M,Oi,Os) sat Name Alice meet Name Bob says prop go
    aclExercise1A
      |- (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
    aclExercise1B
      |- (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
    aclExercise2
       [...] |- (M,Oi,Os) sat Name Bob says prop launch
    aclExercise2A
      |- (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
    aclExercise2B
      |- (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
Exporting theory "solutions1" ... done.
Theory "solutions1" took 0.10774s to build
structure solutions1Script:
  sig

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

>
  end
val it = (): unit
>
```

```
*** Emacs/HOL command completed ***

Process HOL finished
```

Chapter 14

5 Exercise 14.4.1 Problem Statement

This exercise implements a launch CONOPS and an abort CONOS. This means the Commander role can issue a go or nogo command, which should correspond with the staff making the appropriate command to the Applications.

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

keyPrinc = Staff people | Role roles | Ap num

people = Alice | Bob

principals = PR keyPrinc | Key keyPrinc

roles = Commander | Operator | CA
```

Both conops are realized with the addition of the following four theroems

```
⊢ (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

 ⊢ (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

 ⊢ (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
```

6 Chapter 14 Execution Transcript

N

```
---------------------------------------------------------------------
       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
---------------------------------------------------------------------
[extending loadPath with Holmakefile INCLUDES variable]
> > > > > > > > Loading acl_infRules

> > > <<HOL message: Created theory "conops0Solution">>
<<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">>

Theory: conops0Solution

Parents:
    aclDrules

Type constants:
    commands 0
    keyPrinc 0
    people 0
    principals 0
    roles 0

Term constants:
    Alice             :people
    Ap                :num -> keyPrinc
    Bob               :people
    CA                :roles
    Commander         :roles
    Key               :keyPrinc -> principals
    Operator          :roles
    PR                :keyPrinc -> principals
    Role              :roles -> keyPrinc
    Staff             :people -> keyPrinc
    abort             :commands
    activate          :commands
    commands2num      :commands -> num
    commands_CASE     :commands ->  ->  ->  ->  ->  ->  ->
    commands_size     :commands -> num
    go                :commands
    keyPrinc_CASE     :keyPrinc ->
                       (people -> ) -> (roles -> ) -> (num -> ) ->
    keyPrinc_size     :keyPrinc -> num
    launch            :commands
    nogo              :commands
    num2commands      :num -> commands
    num2people        :num -> people
    num2roles         :num -> roles
    people2num        :people -> num
    people_CASE       :people ->  ->  ->
    people_size       :people -> num
    principals_CASE   :principals ->
                       (keyPrinc -> ) -> (keyPrinc -> ) ->
    principals_size   :principals -> num
    roles2num         :roles -> num
    roles_CASE        :roles ->  ->  ->  ->
    roles_size        :roles -> num
    stand_down        :commands

Definitions:
    @tempAlice_def
      |- Alice = num2people 0
    @tempBob_def
      |- Bob = num2people 1
    @tempCA_def
      |- CA = num2roles 2
    @tempCommander_def
      |- Commander = num2roles 0
    @tempOperator_def
      |- Operator = num2roles 1
    @tempabort_def
      |- abort = num2commands 3
    @tempactivate_def
      |- activate = num2commands 4
    @tempgo_def
      |- go = num2commands 0
    @templaunch_def
      |- launch = num2commands 2
    @tempnogo_def
      |- nogo = num2commands 1
    @tempstand_down_def
      |- stand_down = num2commands 5
    commands_BIJ
      |- (a. num2commands (commands2num a) = a)
         r. (n. n < 6) r  (commands2num (num2commands r) = r)
    commands_CASE
      |- x v0 v1 v2 v3 v4 v5.
           (case x of
              go => v0
```

```
        | nogo => v1
        | launch => v2
        | abort => v3
        | activate => v4
        | stand_down => v5) =
       (m.
          if m < 2 then if m = 0 then v0 else v1
          else if m < 3 then v2
          else if m < 4 then v3
          else if m = 4 then v4
          else v5) (commands2num x)
commands_TY_DEF
  |- rep. TYPE_DEFINITION (n. n < 6) rep
commands_size_def
  |- x. commands_size x = 0
keyPrinc_TY_DEF
  |- rep.
       TYPE_DEFINITION
         (a0.
            'keyPrinc' .
              (a0.
                 (a.
                    a0 =
                    (a.
                       ind_type$CONSTR 0 (a,ARB,ARB)
                         (n. ind_type$BOTTOM)) a)
                 (a.
                    a0 =
                    (a.
                       ind_type$CONSTR (SUC 0) (ARB,a,ARB)
                         (n. ind_type$BOTTOM)) a)
                 (a.
                    a0 =
                    (a.
                       ind_type$CONSTR (SUC (SUC 0)) (ARB,ARB,a)
                         (n. ind_type$BOTTOM)) a)
                 'keyPrinc' a0)
              'keyPrinc' a0) rep
keyPrinc_case_def
  |- (a f f1 f2. keyPrinc_CASE (Staff a) f f1 f2 = f a)
     (a f f1 f2. keyPrinc_CASE (Role a) f f1 f2 = f1 a)
     a f f1 f2. keyPrinc_CASE (Ap a) f f1 f2 = f2 a
keyPrinc_size_def
  |- (a. keyPrinc_size (Staff a) = 1 + people_size a)
     (a. keyPrinc_size (Role a) = 1 + roles_size a)
     a. keyPrinc_size (Ap a) = 1 + a
people_BIJ
```

```
      |- (a. num2people (people2num a) = a)
         r. (n. n < 2) r  (people2num (num2people r) = r)
    people_CASE
      |- x v0 v1.
           (case x of Alice => v0 | Bob => v1) =
           (m. if m = 0 then v0 else v1) (people2num x)
    people_TY_DEF
      |- rep. TYPE_DEFINITION (n. n < 2) rep
    people_size_def
      |- x. people_size x = 0
    principals_TY_DEF
      |- rep.
           TYPE_DEFINITION
             (a0.
                'principals' .
                  (a0.
                     (a.
                        a0 =
                        (a. ind_type$CONSTR 0 a (n. ind_type$BOTTOM))
                          a)
                     (a.
                        a0 =
                        (a.
                           ind_type$CONSTR (SUC 0) a
                             (n. ind_type$BOTTOM)) a)
                     'principals' a0)
                  'principals' a0) rep
    principals_case_def
      |- (a f f1. principals_CASE (PR a) f f1 = f a)
         a f f1. principals_CASE (Key a) f f1 = f1 a
    principals_size_def
      |- (a. principals_size (PR a) = 1 + keyPrinc_size a)
         a. principals_size (Key a) = 1 + keyPrinc_size a
    roles_BIJ
      |- (a. num2roles (roles2num a) = a)
         r. (n. n < 3) r  (roles2num (num2roles r) = r)
    roles_CASE
      |- x v0 v1 v2.
           (case x of Commander => v0 | Operator => v1 | CA => v2) =
           (m. if m < 1 then v0 else if m = 1 then v1 else v2)
             (roles2num x)
    roles_TY_DEF
      |- rep. TYPE_DEFINITION (n. n < 3) rep
    roles_size_def
      |- x. roles_size x = 0

Theorems:
```

```
ApRuleActive_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
commands2num_11
  |- a a’. (commands2num a = commands2num a’)  (a = a’)
commands2num_ONTO
  |- r. r < 6  a. r = commands2num a
commands2num_num2commands
  |- r. r < 6  (commands2num (num2commands r) = r)
commands2num_thm
  |- (commands2num go = 0)  (commands2num nogo = 1)
     (commands2num launch = 2)  (commands2num abort = 3)
     (commands2num activate = 4)  (commands2num stand_down = 5)
commands_Axiom
  |- x0 x1 x2 x3 x4 x5.
       f.
         (f go = x0)  (f nogo = x1)  (f launch = x2)
         (f abort = x3)  (f activate = x4)  (f stand_down = x5)
commands_EQ_commands
  |- a a’. (a = a’)  (commands2num a = commands2num a’)
commands_case_cong
```

```
  |- M M’ v0 v1 v2 v3 v4 v5.
       (M = M’)  ((M’ = go)  (v0 = v0’))
       ((M’ = nogo)  (v1 = v1’))  ((M’ = launch)  (v2 = v2’))
       ((M’ = abort)  (v3 = v3’))
       ((M’ = activate)  (v4 = v4’))
       ((M’ = stand_down)  (v5 = v5’))
       ((case M of
           go => v0
         | nogo => v1
         | launch => v2
         | abort => v3
         | activate => v4
         | stand_down => v5) =
        case M’ of
          go => v0’
        | nogo => v1’
        | launch => v2’
        | abort => v3’
        | activate => v4’
        | stand_down => v5’)
commands_case_def
  |- (v0 v1 v2 v3 v4 v5.
        (case go of
           go => v0
         | nogo => v1
         | launch => v2
         | abort => v3
         | activate => v4
         | stand_down => v5) =
        v0)
     (v0 v1 v2 v3 v4 v5.
        (case nogo of
           go => v0
         | nogo => v1
         | launch => v2
         | abort => v3
         | activate => v4
         | stand_down => v5) =
        v1)
     (v0 v1 v2 v3 v4 v5.
        (case launch of
           go => v0
         | nogo => v1
         | launch => v2
         | abort => v3
         | activate => v4
         | stand_down => v5) =
        v2)
     (v0 v1 v2 v3 v4 v5.
        (case abort of
           go => v0
         | nogo => v1
         | launch => v2
         | abort => v3
         | activate => v4
         | stand_down => v5) =
        v3)
     (v0 v1 v2 v3 v4 v5.
        (case activate of
           go => v0
         | nogo => v1
         | launch => v2
         | abort => v3
         | activate => v4
         | stand_down => v5) =
        v4)
     v0 v1 v2 v3 v4 v5.
       (case stand_down of
          go => v0
        | nogo => v1
        | launch => v2
        | abort => v3
        | activate => v4
        | stand_down => v5) =
       v5
commands_distinct
  |- go  nogo  go  launch  go  abort  go  activate
     go  stand_down  nogo  launch  nogo  abort
     nogo  activate  nogo  stand_down  launch  abort
     launch  activate  launch  stand_down  abort  activate
     abort  stand_down  activate  stand_down
commands_induction
  |- P.
       P abort  P activate  P go  P launch  P nogo
       P stand_down
       a. P a
commands_nchotomy
  |- a.
       (a = go)  (a = nogo)  (a = launch)  (a = abort)
       (a = activate)  (a = stand_down)
datatype_commands
  |- DATATYPE (commands go nogo launch abort activate stand_down)
datatype_keyPrinc
  |- DATATYPE (keyPrinc Staff Role Ap)
```

```
datatype_people
  |- DATATYPE (people Alice Bob)
datatype_principals
  |- DATATYPE (principals PR Key)
datatype_roles
  |- DATATYPE (roles Commander Operator CA)
keyPrinc_11
  |- (a a'. (Staff a = Staff a')  (a = a'))
     (a a'. (Role a = Role a')  (a = a'))
     a a'. (Ap a = Ap a')  (a = a')
keyPrinc_Axiom
  |- f0 f1 f2.
       fn.
         (a. fn (Staff a) = f0 a)  (a. fn (Role a) = f1 a)
         a. fn (Ap a) = f2 a
keyPrinc_case_cong
  |- M M' f f1 f2.
       (M = M')  (a. (M' = Staff a)  (f a = f' a))
       (a. (M' = Role a)  (f1 a = f1' a))
       (a. (M' = Ap a)  (f2 a = f2' a))
       (keyPrinc_CASE M f f1 f2 = keyPrinc_CASE M' f' f1' f2')
keyPrinc_distinct
  |- (a' a. Staff a  Role a')  (a' a. Staff a  Ap a')
     a' a. Role a  Ap a'
keyPrinc_induction
  |- P.
       (p. P (Staff p))  (r. P (Role r))  (n. P (Ap n))
       k. P k
keyPrinc_nchotomy
  |- kk. (p. kk = Staff p)  (r. kk = Role r)  n. kk = Ap n
num2commands_11
  |- r r'.
       r < 6
       r' < 6
       ((num2commands r = num2commands r')  (r = r'))
num2commands_ONTO
  |- a. r. (a = num2commands r)  r < 6
num2commands_commands2num
  |- a. num2commands (commands2num a) = a
num2commands_thm
  |- (num2commands 0 = go)  (num2commands 1 = nogo)
     (num2commands 2 = launch)  (num2commands 3 = abort)
     (num2commands 4 = activate)  (num2commands 5 = stand_down)
num2people_11
  |- r r'.
       r < 2  r' < 2  ((num2people r = num2people r')  (r = r'))
num2people_ONTO
```

```
  |- a. r. (a = num2people r)  r < 2
num2people_people2num
  |- a. num2people (people2num a) = a
num2people_thm
  |- (num2people 0 = Alice)  (num2people 1 = Bob)
num2roles_11
  |- r r'.
       r < 3  r' < 3  ((num2roles r = num2roles r')  (r = r'))
num2roles_ONTO
  |- a. r. (a = num2roles r)  r < 3
num2roles_roles2num
  |- a. num2roles (roles2num a) = a
num2roles_thm
  |- (num2roles 0 = Commander)  (num2roles 1 = Operator)
     (num2roles 2 = CA)
people2num_11
  |- a a'. (people2num a = people2num a')  (a = a')
people2num_ONTO
  |- r. r < 2  a. r = people2num a
people2num_num2people
  |- r. r < 2  (people2num (num2people r) = r)
people2num_thm
  |- (people2num Alice = 0)  (people2num Bob = 1)
people_Axiom
  |- x0 x1. f. (f Alice = x0)  (f Bob = x1)
people_EQ_people
  |- a a'. (a = a')  (people2num a = people2num a')
people_case_cong
  |- M M' v0 v1.
       (M = M')  ((M' = Alice)  (v0 = v0'))
       ((M' = Bob)  (v1 = v1'))
       ((case M of Alice => v0 | Bob => v1) =
        case M' of Alice => v0' | Bob => v1')
people_case_def
  |- (v0 v1. (case Alice of Alice => v0 | Bob => v1) = v0)
     v0 v1. (case Bob of Alice => v0 | Bob => v1) = v1
people_distinct
  |- Alice  Bob
people_induction
  |- P. P Alice  P Bob  a. P a
people_nchotomy
  |- a. (a = Alice)  (a = Bob)
principals_11
  |- (a a'. (PR a = PR a')  (a = a'))
     a a'. (Key a = Key a')  (a = a')
principals_Axiom
  |- f0 f1. fn. (a. fn (PR a) = f0 a)  a. fn (Key a) = f1 a
```

```
principals_case_cong
  |- M M' f f1.
       (M = M')  (a. (M' = PR a)  (f a = f' a))
       (a. (M' = Key a)  (f1 a = f1' a))
       (principals_CASE M f f1 = principals_CASE M' f' f1')
principals_distinct
  |- a' a. PR a  Key a'
principals_induction
  |- P. (k. P (PR k))  (k. P (Key k))  p. P p
principals_nchotomy
  |- pp. (k. pp = PR k)  k. pp = Key k
roles2num_11
  |- a a'. (roles2num a = roles2num a')  (a = a')
roles2num_ONTO
  |- r. r < 3  a. r = roles2num a
roles2num_num2roles
  |- r. r < 3  (roles2num (num2roles r) = r)
roles2num_thm
  |- (roles2num Commander = 0)  (roles2num Operator = 1)
     (roles2num CA = 2)
roles_Axiom
  |- x0 x1 x2.
       f. (f Commander = x0)  (f Operator = x1)  (f CA = x2)
roles_EQ_roles
  |- a a'. (a = a')  (roles2num a = roles2num a')
roles_case_cong
  |- M M' v0 v1 v2.
       (M = M')  ((M' = Commander)  (v0 = v0'))
       ((M' = Operator)  (v1 = v1'))  ((M' = CA)  (v2 = v2'))
       ((case M of Commander => v0 | Operator => v1 | CA => v2) =
        case M' of Commander => v0' | Operator => v1' | CA => v2')
roles_case_def
  |- (v0 v1 v2.
        (case Commander of
           Commander => v0
         | Operator => v1
         | CA => v2) =
        v0)
     (v0 v1 v2.
        (case Operator of
           Commander => v0
         | Operator => v1
         | CA => v2) =
        v1)
     v0 v1 v2.
       (case CA of Commander => v0 | Operator => v1 | CA => v2) = v2
roles_distinct
```

```
      |- Commander  Operator  Commander  CA  Operator  CA
    roles_induction
      |- P. P CA  P Commander  P Operator  a. P a
    roles_nchotomy
      |- a. (a = Commander)  (a = Operator)  (a = CA)
Exporting theory "conops0Solution" ... done.
Theory "conops0Solution" took 1.2s to build
structure conops0SolutionScript:
   sig

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

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

Process HOL finished
```

A Source for solutions1Script.sml

The following code is from *HOL/solutions1Script.sml*

```
(*********************************************)
(* Project 6:  solutions1Script.sml          *)
(* Alfred Murabito                           *)
(* Date: 22 February 2020                     *)
(*********************************************)

(* Exercise 13.10.1 [Alice says gp, Bob says go] -| Alice & Bob says go *)

structure solutions1Script = struct

(* only necessary when working interactively
app load ["acl_infRules","aclrulesTheory","aclDrulesTheory","solutions1Theory"];
open acl_infRules aclrulesTheory aclDrulesTheory example1Theory solutions1Theory
*)

open HolKernel boolLib Parse bossLib example1Theory
open acl_infRules aclrulesTheory aclDrulesTheory example1Theory

val _ = new_theory "solutions1"

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 th1)) th4
val th6 = DISCH(hd(hyp th2)) th5

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 th1)) th4
in
   DISCH(hd(hyp th2)) th5
end;

val _ = save_thm("aclExercise1", aclExercise1)

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[SPEC_ALL Conjunction, GSYM(SPEC_ALL And_Says_Eq)])

val _ = save_thm("aclExercise1A", aclExercise1A)

(* interactive mode ===
set_goal([],
``((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)``);
e(REPEAT STRIP_TAC);
e(ACL_AND_SAYS_RL_TAC);
e(ACL_CONJ_TAC THEN PROVE_TAC []);
=== end interactive mode*)

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)

(* Exercise 13.10.2
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 = CONTROLS th2 th1;
val th4 = ACL_ASSUM``((prop go) impf (prop launch)):(commands,staff,'d,'e)Form``
val th5 = ACL_MP th3 th4;
val th6 = SAYS ``Name Bob`` th5;
*)

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)Fo
   val th3 = CONTROLS th2 th1
   val th4 = ACL_ASSUM``((prop go) impf (prop launch)):(commands,staff,'d,'e)Form
   val th5 = ACL_MP th3 th4
in
  SAYS ``Name Bob`` th5
```

```
end;

val _ = save_thm(”aclExercise2”, aclExercise2)

(* interactive mode
set_goal([],
‘‘((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)‘‘);
end interactive mode *)


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[Says, Controls, Modus_Ponens]);

val _ = save_thm(”aclExercise2A”, aclExercise2A)

(* interactive mode
set_goal([],
‘‘((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 => ASSUME_TAC(ACL_MP th1 th2)))) THEN
   ASM_REWRITE_TAC []
end interactive mode *)
```

```
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 => ASSUME_TAC(ACL_MP th1 th2)))) THEN
   ASM_REWRITE_TAC []
);

val _ = save_thm("aclExercise2B", aclExercise2B)


val _ = print_theory "-";

val _ = export_theory();

end (* structure *)
```

B Source for conops0SolutionScript.sml

The following code is from *HOL/conops0SolutionScript.sml*

```
(***************************************************)
(* Project 6:  conops0SolutionScript.sml              *)
(* Alfred Murabito                                  *)
(* Date: 2 February 2020                            *)
(***************************************************)

structure conops0SolutionScript = struct

(* only necessary when working interactively
app load ["acl_infRules","aclrulesTheory","aclDrulesTheory","conops0SolutionTheo
```

```
open acl_infRules aclrulesTheory aclDrulesTheory conops0SolutionTheory
*)

open HolKernel boolLib Parse 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 conops0Solution$people | Role conops0Solution$roles | Ap num`

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

(* interactive mode

set_goal
([],
``(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``
);

REPEAT STRIP_TAC THEN
ACL_SAYS_TAC THEN
PAT_ASSUM ``(M,Oi,Os) sat
            Name (Key (Role CA)) speaks_for Name (PR (Role CA))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Role CA)) says
      Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role CA)) controls
            Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (PR (Role CA)) says
      Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
    (fn th2 => ASSUME_TAC(CONTROLS th1 th2)))) THEN

(* Derive Key Alice says Commander says go *)
PAT_ASSUM ``(M,Oi,Os) sat Name (Key (Staff Alice)) quoting
            Name (PR (Role Commander)) says prop go``
  (fn th => ASSUME_TAC (QUOTING_LR th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Staff Alice)) says
      Name (PR (Role Commander)) says (prop go)``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            Name (PR (Staff Alice)) says Name (PR (Role Commander)) says
            (prop go)``
  (fn th => ASSUME_TAC (QUOTING_RL th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
           reps (Name (PR (Staff Alice))) (Name (PR (Role Commander)))(prop go)`
  (fn th1 =>
    (PAT_ASSUM ``(M,Oi,Os) sat
    Name (PR (Staff Alice)) quoting Name (PR (Role Commander)) says
    (prop go)``
      (fn th2 =>
        (PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role Commander)) controls (prop go)``
          (fn th3 => ASSUME_TAC (REPS th1 th2 th3)))))) THEN

PAT_ASSUM ``(M,Oi,Os) sat (prop go)`` (fn th1 =>
  (PAT_ASSUM ``(M,Oi,Os) sat prop go impf prop launch``
    (fn th2 => ASSUME_TAC (ACL_MP th1 th2)))) THEN

ASM_REWRITE_TAC[]
*)

val OpRuleLaunch_thm =
TAC_PROOF(
([], ``(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``),
REPEAT STRIP_TAC THEN
ACL_SAYS_TAC THEN
PAT_ASSUM ``(M,Oi,Os) sat
           Name (Key (Role CA)) speaks_for Name (PR (Role CA))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Role CA)) says
      Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role CA)) controls
           Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (PR (Role CA)) says
      Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
    (fn th2 => ASSUME_TAC(CONTROLS th1 th2)))) THEN

(* Derive Key Alice says Commander says go *)
PAT_ASSUM ``(M,Oi,Os) sat Name (Key (Staff Alice)) quoting
            Name (PR (Role Commander)) says prop go``
  (fn th => ASSUME_TAC (QUOTING_LR th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Staff Alice)) says
      Name (PR (Role Commander)) says (prop go)``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            Name (PR (Staff Alice)) says Name (PR (Role Commander)) says
            (prop go)``
  (fn th => ASSUME_TAC (QUOTING_RL th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
           reps (Name (PR (Staff Alice))) (Name (PR (Role Commander)))(prop go)`
  (fn th1 =>
    (PAT_ASSUM ``(M,Oi,Os) sat
    Name (PR (Staff Alice)) quoting Name (PR (Role Commander)) says
    (prop go)``
      (fn th2 =>
        (PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role Commander)) controls (prop go)
          (fn th3 => ASSUME_TAC (REPS th1 th2 th3)))))) THEN


PAT_ASSUM ``(M,Oi,Os) sat (prop go)`` (fn th1 =>
  (PAT_ASSUM ``(M,Oi,Os) sat prop go impf prop launch``
    (fn th2 => ASSUME_TAC (ACL_MP th1 th2)))) THEN

ASM_REWRITE_TAC[]
)


val ApRuleActive_thm =
TAC_PROOF(
([], ``(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``),
REPEAT STRIP_TAC THEN
PAT_ASSUM ``(M,Oi,Os) sat
            Name (Key (Role CA)) speaks_for Name (PR (Role CA))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Role CA)) says
      Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob))``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role CA)) controls
            Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (PR (Role CA)) says
      Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob))``
    (fn th2 => ASSUME_TAC(CONTROLS th1 th2)))) THEN

(* Derive Key Bob says Operator says launch *)
PAT_ASSUM ``(M,Oi,Os) sat Name (Key (Staff Bob)) quoting
            Name (PR (Role Operator)) says prop launch``
  (fn th => ASSUME_TAC (QUOTING_LR th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Staff Bob)) says
      Name (PR (Role Operator)) says (prop launch)``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            Name (PR (Staff Bob)) says Name (PR (Role Operator)) says
            (prop launch)``
  (fn th => ASSUME_TAC (QUOTING_RL th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
           reps (Name (PR (Staff Bob))) (Name (PR (Role Operator)))(prop launch)
  (fn th1 =>
    (PAT_ASSUM ``(M,Oi,Os) sat
    Name (PR (Staff Bob)) quoting Name (PR (Role Operator)) says
    (prop launch)``
      (fn th2 =>
        (PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role Operator)) controls (prop launc
          (fn th3 => ASSUME_TAC (REPS th1 th2 th3)))))) THEN


PAT_ASSUM ``(M,Oi,Os) sat (prop launch)`` (fn th1 =>
  (PAT_ASSUM ``(M,Oi,Os) sat prop launch impf prop activate``
    (fn th2 => ASSUME_TAC (ACL_MP th1 th2)))) THEN

ASM_REWRITE_TAC[]
)

val OpRuleAbort_thm =
TAC_PROOF(
([], ``(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``),
REPEAT STRIP_TAC THEN
ACL_SAYS_TAC THEN
PAT_ASSUM ``(M,Oi,Os) sat
            Name (Key (Role CA)) speaks_for Name (PR (Role CA))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Role CA)) says
      Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role CA)) controls
             Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (PR (Role CA)) says
      Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
    (fn th2 => ASSUME_TAC(CONTROLS th1 th2)))) THEN

(* Derive Key Alice says Commander says nogo *)
PAT_ASSUM ``(M,Oi,Os) sat Name (Key (Staff Alice)) quoting
             Name (PR (Role Commander)) says prop nogo``
  (fn th => ASSUME_TAC (QUOTING_LR th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
             Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Staff Alice)) says
      Name (PR (Role Commander)) says (prop nogo)``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat
             Name (PR (Staff Alice)) says Name (PR (Role Commander)) says
             (prop nogo)``
  (fn th => ASSUME_TAC (QUOTING_RL th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            reps (Name (PR (Staff Alice))) (Name (PR (Role Commander)))(prop nogo)``
  (fn th1 =>
    (PAT_ASSUM ``(M,Oi,Os) sat
    Name (PR (Staff Alice)) quoting Name (PR (Role Commander)) says
    (prop nogo)``
      (fn th2 =>
        (PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role Commander)) controls (prop nogo)``
          (fn th3 => ASSUME_TAC (REPS th1 th2 th3)))))) THEN


PAT_ASSUM ``(M,Oi,Os) sat (prop nogo)`` (fn th1 =>
  (PAT_ASSUM ``(M,Oi,Os) sat prop nogo impf prop abort``
    (fn th2 => ASSUME_TAC (ACL_MP th1 th2)))) THEN

ASM_REWRITE_TAC[]
)
```

```
val ApRuleStandDown_thm =
TAC_PROOF(
 ([], ``(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``),
REPEAT STRIP_TAC THEN
PAT_ASSUM ``(M,Oi,Os) sat
            Name (Key (Role CA)) speaks_for Name (PR (Role CA))``
   (fn th1 =>
     (PAT_ASSUM
     ``(M,Oi,Os) sat Name (Key (Role CA)) says
       Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob))``
     (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role CA)) controls
            Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob))``
   (fn th1 =>
     (PAT_ASSUM
     ``(M,Oi,Os) sat Name (PR (Role CA)) says
       Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob))``
     (fn th2 => ASSUME_TAC(CONTROLS th1 th2)))) THEN

(* Derive Key Bob says Operator says abort *)
PAT_ASSUM ``(M,Oi,Os) sat Name (Key (Staff Bob)) quoting
            Name (PR (Role Operator)) says prop abort``
   (fn th => ASSUME_TAC (QUOTING_LR th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            Name (Key (Staff Bob)) speaks_for Name (PR (Staff Bob))``
   (fn th1 =>
     (PAT_ASSUM
     ``(M,Oi,Os) sat Name (Key (Staff Bob)) says
       Name (PR (Role Operator)) says (prop abort)``
     (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            Name (PR (Staff Bob)) says Name (PR (Role Operator)) says
            (prop abort)``
   (fn th => ASSUME_TAC (QUOTING_RL th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
           reps (Name (PR (Staff Bob))) (Name (PR (Role Operator)))(prop abort)``
   (fn th1 =>
     (PAT_ASSUM ``(M,Oi,Os) sat
     Name (PR (Staff Bob)) quoting Name (PR (Role Operator)) says
     (prop abort)``
       (fn th2 =>
         (PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role Operator)) controls (prop abort)``
           (fn th3 => ASSUME_TAC (REPS th1 th2 th3)))))) THEN


PAT_ASSUM ``(M,Oi,Os) sat (prop abort)`` (fn th1 =>
   (PAT_ASSUM ``(M,Oi,Os) sat prop abort impf prop stand_down``
     (fn th2 => ASSUME_TAC (ACL_MP th1 th2)))) THEN

ASM_REWRITE_TAC[]
)

val _ = save_thm(”OpRuleLaunch_thm”, OpRuleLaunch_thm)
val _ = save_thm(”ApRuleActive_thm”, ApRuleActive_thm)
val _ = save_thm(”OpRuleAbort_thm”, OpRuleAbort_thm)
val _ = save_thm(”ApRuleStandDown_thm”, ApRuleStandDown_thm)

val _ = print_theory ”-”;

val _ = export_theory();

end (* structure *)
```