Abstract

This is the abstract for my master's thesis. Here is some text for formatting.

At least 18 people have been killed and dozens trapped in the Indian city of Varanasi after a flyover collapsed, crushing vehicles beneath it. The flyover was still being built when portions of its cement structure fell on the road being used under it. Officials from the National Disaster Response Force said 18 bodies had been recovered so far. A rescue operation is continuing for those believed to still be trapped, but their number and condition is unknown. Photographs and video from the scene showed cars and a bus crushed beneath the weight of the concrete, many of which still held people inside. Local media reported that a handful of people had been successfully rescued, as seven cranes attempted to lift the concrete pillar. A large crowd also gathered at the scene. One eyewitness told reporters they were nearby when the collapse happened. "At least four cars, an auto-rickshaw and a minibus were crushed under it," they said.

India's NDTV also reported that many of those trapped are believed to be construction workers who had been building the flyover. The cause of the collapse is not yet known, and an inquiry has been ordered, NDTV added. Major collapses of buildings and other infrastructure are not uncommon in India, where the enforcement of construction standards is weaker than many Western countries. In September, 33 people died when a six-storey Mumbai building toppled and more than 20 people died in 2016 when a flyover collapsed in Kolkata. Other collapses with smaller death tolls are frequent. Varanasi is the home constituency of India's Prime Minister Narendra Modi, who said he was "extremely saddened by the loss of lives due to the collapse". "I pray that the injured recover soon. Spoke to officials and asked them to ensure all possible support to those affected," he tweeted.

Copyright

Disclaimer

Acknowledgements

Table Of Contents

List of Figures

List of Tables

List of Acronyms

Chapter 1

Introduction

Some text here.[1]

Chapter 2

Background

Formal Methods

Functional Programming

Higher Order Logic (HOL) Interactive Theorem Prover

Other Interactive Theorem Provers

Chapter 3

Systems Security Engineering

3.1 NIST Special Publication 800-160

3.2 Verification & Documentation

3.3 Principle of Complete Mediation

3.3.1 Formal Verification Using Computer-Aided Reasoning

Chapter 4

Patrol Base Operations

This is the future works section. But, as I am typing this, it is the current working section for LaTeXThe point here is to get the margins in order. This means that there must be text of sufficient length to visually verify that the text meets LORI's standards. LORI is complying with SU standards for the senior thesis. Therefore, meeting LORI's standards is synonymous with meeting SU's standards. Resistance will only degrade you.

4.1 Motivation

4.2 Ranger Handbook Description

4.3 Describing The Patrol Base Operations

4.4 Hierarchy of Secure State Machines

4.4.1 OMNI-Level

4.4.2 Escape

4.4.3 Top Level

4.4.4 Horizontal Slice

4.4.4.1 ssmPlanPB

4.4.4.2 ssmMoveToORP

4.4.4.3 ssmConductORP

4.4.4.4 ssmMoveToPB

4.4.4.5 ssmConductPB

4.4.5 Vertical Slice

Chapter 5

Secure State Machine Model

5.1 State Machines

5.1.1 Next-state Function

5.1.2 Next-output Function

5.1.3 Transition Commands

5.2 Secure State Machines

5.2.1 State Machine Versus Secure State Machine

5.2.2 Transition Types

5.2.3 Authentication

5.2.4 Authorization

Chapter 6

Patrol Base Operations as Secure State Machines

6.1 ssmPB: An Example from the Hierarchy

6.1.1 Principals

6.1.2 States

6.1.3 Commands

6.1.4 Next-State Function

6.1.5 Next-Output Function

6.1.6 Authentication

6.1.7 Authorization

Chapter 7

Discussion

7.1 Recap

7.2 Mission Accomplished

7.3 Stop-Gaps, Lessons Learned, & Advice

7.4 Other Verifiable Theories

7.4.1 Platoon Theory, Soldier Theory, Squad Theory, etc.

7.4.2 Soldiers in Roles

Chapter 8

Future Work & Implications

This is the future works section. But, as I am typing this, it is the current working section for LaTeX˙The point here is to get the margins in order. This means that there must be text of sufficient length to visually verify that the text meets LORI's standards. LORI is complying with SU standards for the senior thesis. Therefore, meeting LORI's standards is synonymous with meeting SU's standards. Resistance will only degrade you.

8.1 The Devil Is in The Details

Of course, there are top margins and bottom margins. This means that we'll need more text. You know, the best wait to generate text is to just cut-n-paste some random stuff. Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080

Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development

company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080 Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080 Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

8.2 Accountability Systems

Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080 Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment

complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

8.3 Applicability

Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080 Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

Appendices

Appendix A

Access Control Logic Theories: Pretty-Printed Theories

Contents

1 aclfoundation Theory

Built: 25 February 2018
Parent Theories: indexedLists, patternMatches

1.1 Datatypes

Form =
```
    TT
  | FF
  | prop 'aavar
  | notf (('aavar, 'apn, 'il, 'sl) Form)
  | (andf) (('aavar, 'apn, 'il, 'sl) Form)
           (('aavar, 'apn, 'il, 'sl) Form)
  | (orf) (('aavar, 'apn, 'il, 'sl) Form)
          (('aavar, 'apn, 'il, 'sl) Form)
  | (impf) (('aavar, 'apn, 'il, 'sl) Form)
           (('aavar, 'apn, 'il, 'sl) Form)
  | (eqf) (('aavar, 'apn, 'il, 'sl) Form)
          (('aavar, 'apn, 'il, 'sl) Form)
  | (says) ('apn Princ) (('aavar, 'apn, 'il, 'sl) Form)
  | (speaks_for) ('apn Princ) ('apn Princ)
  | (controls) ('apn Princ) (('aavar, 'apn, 'il, 'sl) Form)
  | reps ('apn Princ) ('apn Princ)
         (('aavar, 'apn, 'il, 'sl) Form)
  | (domi) (('apn, 'il) IntLevel) (('apn, 'il) IntLevel)
  | (eqi) (('apn, 'il) IntLevel) (('apn, 'il) IntLevel)
  | (doms) (('apn, 'sl) SecLevel) (('apn, 'sl) SecLevel)
  | (eqs) (('apn, 'sl) SecLevel) (('apn, 'sl) SecLevel)
  | (eqn) num num
  | (lte) num num
  | (lt) num num
```

Kripke =
```
    KS ('aavar -> 'aaworld -> bool)
       ('apn -> 'aaworld -> 'aaworld -> bool) ('apn -> 'il)
       ('apn -> 'sl)
```

Princ =
```
    Name 'apn
  | (meet) ('apn Princ) ('apn Princ)
  | (quoting) ('apn Princ) ('apn Princ) ;
```

IntLevel = `iLab 'il | il 'apn ;`

SecLevel = `sLab 'sl | sl 'apn`

1.2 Definitions

[imapKS_def]

$\vdash \forall Intp\ Jfn\ ilmap\ slmap.$
imapKS (KS $Intp$ Jfn $ilmap$ $slmap$) = $ilmap$

[intpKS_def]

$\vdash \forall Intp\ Jfn\ ilmap\ slmap.$
intpKS (KS $Intp$ Jfn $ilmap$ $slmap$) = $Intp$

[jKS_def]

$\vdash \forall Intp\ Jfn\ ilmap\ slmap.$ jKS (KS $Intp$ Jfn $ilmap$ $slmap$) = Jfn

[O1_def]

$\vdash$ O1 = PO one_weakorder

[one_weakorder_def]

$\vdash \forall x\ y.$ one_weakorder x y $\iff$ T

[po_TY_DEF]

$\vdash \exists rep.$ TYPE_DEFINITION WeakOrder rep

[po_tybij]

$\vdash (\forall a.$ PO (repPO a) = a) $\wedge$
$\forall r.$ WeakOrder r $\iff$ (repPO (PO r) = r)

[prod_PO_def]

$\vdash \forall PO_1\ PO_2.$
prod_PO PO_1 PO_2 = PO (RPROD (repPO PO_1) (repPO PO_2))

[smapKS_def]

$\vdash \forall Intp\ Jfn\ ilmap\ slmap.$
smapKS (KS $Intp$ Jfn $ilmap$ $slmap$) = $slmap$

[Subset_PO_def]

$\vdash$ Subset_PO = PO ($\subseteq$)

1.3 Theorems

[abs_po11]

$\vdash \forall r\ r'.$
WeakOrder r $\Rightarrow$ WeakOrder r' $\Rightarrow$ ((PO r = PO r') $\iff$ (r = r'))

[absPO_fn_onto]

$\vdash \forall a.\ \exists r.$ (a = PO r) $\wedge$ WeakOrder r

[antisym_prod_antisym]

$\vdash \forall r\ s.$
antisymmetric $r \wedge$ antisymmetric $s \Rightarrow$
antisymmetric (RPROD r s)

[EQ_WeakOrder]

$\vdash$ WeakOrder (=)

[KS_bij]

$\vdash \forall M.\ M$ = KS (intpKS M) (jKS M) (imapKS M) (smapKS M)

[one_weakorder_WO]

$\vdash$ WeakOrder one_weakorder

[onto_po]

$\vdash \forall r.$ WeakOrder $r \iff \exists a.\ r$ = repPO a

[po_bij]

$\vdash$ $(\forall a.$ PO (repPO a) = $a) \wedge$
$\forall r.$ WeakOrder $r \iff$ (repPO (PO r) = r)

[PO_repPO]

$\vdash \forall a.$ PO (repPO a) = a

[refl_prod_refl]

$\vdash \forall r\ s.$ reflexive $r \wedge$ reflexive $s \Rightarrow$ reflexive (RPROD r s)

[repPO_iPO_partial_order]

$\vdash$ $(\forall x.$ repPO iPO x $x) \wedge$
$(\forall x\ y.$ repPO iPO x $y \wedge$ repPO iPO y $x \Rightarrow$ (x = y)) $\wedge$
$\forall x\ y\ z.$ repPO iPO x $y \wedge$ repPO iPO y $z \Rightarrow$ repPO iPO x z

[repPO_O1]

$\vdash$ repPO O1 = one_weakorder

[repPO_prod_PO]

$\vdash \forall po_1\ po_2.$
repPO (prod_PO po_1 po_2) = RPROD (repPO po_1) (repPO po_2)

[repPO_Subset_PO]

$\vdash$ repPO Subset_PO = ($\subseteq$)

[RPROD_THM]

$\vdash \forall r\ s\ a\ b.$
RPROD r s a $b \iff r$ (FST a) (FST b) $\wedge$ s (SND a) (SND b)

[SUBSET_WO]

$\vdash$ WeakOrder ($\subseteq$)

[trans_prod_trans]

$\vdash \forall r\ s.$ transitive $r \wedge$ transitive $s \Rightarrow$ transitive (RPROD $r\ s$)

[WeakOrder_Exists]

$\vdash \exists R.$ WeakOrder R

[WO_prod_WO]

$\vdash \forall r\ s.$ WeakOrder $r \wedge$ WeakOrder $s \Rightarrow$ WeakOrder (RPROD $r\ s$)

[WO_repPO]

$\vdash \forall r.$ WeakOrder $r \iff$ (repPO (PO r) = r)

2 aclsemantics Theory

Built: 25 February 2018
Parent Theories: aclfoundation

2.1 Definitions

[Efn_def]

$\vdash$ ($\forall Oi\ Os\ M.$ Efn $Oi\ Os\ M$ TT = $\mathcal{U}$(:'v)) $\wedge$
($\forall Oi\ Os\ M.$ Efn $Oi\ Os\ M$ FF = $\{\}$) $\wedge$
($\forall Oi\ Os\ M\ p.$ Efn $Oi\ Os\ M$ (prop p) = intpKS $M\ p$) $\wedge$
($\forall Oi\ Os\ M\ f.$
Efn $Oi\ Os\ M$ (notf f) = $\mathcal{U}$(:'v) DIFF Efn $Oi\ Os\ M\ f$) $\wedge$
($\forall Oi\ Os\ M\ f_1\ f_2.$
Efn $Oi\ Os\ M$ (f_1 andf f_2) =
Efn $Oi\ Os\ M\ f_1 \cap$ Efn $Oi\ Os\ M\ f_2$) $\wedge$
($\forall Oi\ Os\ M\ f_1\ f_2.$
Efn $Oi\ Os\ M$ (f_1 orf f_2) =
Efn $Oi\ Os\ M\ f_1 \cup$ Efn $Oi\ Os\ M\ f_2$) $\wedge$
($\forall Oi\ Os\ M\ f_1\ f_2.$
Efn $Oi\ Os\ M$ (f_1 impf f_2) =
$\mathcal{U}$(:'v) DIFF Efn $Oi\ Os\ M\ f_1 \cup$ Efn $Oi\ Os\ M\ f_2$) $\wedge$
($\forall Oi\ Os\ M\ f_1\ f_2.$
Efn $Oi\ Os\ M$ (f_1 eqf f_2) =
($\mathcal{U}$(:'v) DIFF Efn $Oi\ Os\ M\ f_1 \cup$ Efn $Oi\ Os\ M\ f_2$) $\cap$
($\mathcal{U}$(:'v) DIFF Efn $Oi\ Os\ M\ f_2 \cup$ Efn $Oi\ Os\ M\ f_1$)) $\wedge$
($\forall Oi\ Os\ M\ P\ f.$
Efn $Oi\ Os\ M$ (P says f) =
$\{w$ | Jext (jKS M) $P\ w \subseteq$ Efn $Oi\ Os\ M\ f\}$) $\wedge$
($\forall Oi\ Os\ M\ P\ Q.$
Efn $Oi\ Os\ M$ (P speaks_for Q) =

if Jext (jKS M) Q RSUBSET Jext (jKS M) P **then** $\mathcal{U}$(:'v)
else { }) ∧
(∀ Oi Os M P f.
Efn Oi Os M (P controls f) =
$\mathcal{U}$(:'v) DIFF {w | Jext (jKS M) P w ⊆ Efn Oi Os M f} ∪
Efn Oi Os M f) ∧
(∀ Oi Os M P Q f.
Efn Oi Os M (reps P Q f) =
$\mathcal{U}$(:'v) DIFF
{w | Jext (jKS M) (P quoting Q) w ⊆ Efn Oi Os M f} ∪
{w | Jext (jKS M) Q w ⊆ Efn Oi Os M f}) ∧
(∀ Oi Os M $intl_1$ $intl_2$.
Efn Oi Os M ($intl_1$ domi $intl_2$) =
if repPO Oi (Lifn M $intl_2$) (Lifn M $intl_1$) **then** $\mathcal{U}$(:'v)
else { }) ∧
(∀ Oi Os M $intl_2$ $intl_1$.
Efn Oi Os M ($intl_2$ eqi $intl_1$) =
(**if** repPO Oi (Lifn M $intl_2$) (Lifn M $intl_1$) **then** $\mathcal{U}$(:'v)
else { }) ∩
if repPO Oi (Lifn M $intl_1$) (Lifn M $intl_2$) **then** $\mathcal{U}$(:'v)
else { }) ∧
(∀ Oi Os M $secl_1$ $secl_2$.
Efn Oi Os M ($secl_1$ doms $secl_2$) =
if repPO Os (Lsfn M $secl_2$) (Lsfn M $secl_1$) **then** $\mathcal{U}$(:'v)
else { }) ∧
(∀ Oi Os M $secl_2$ $secl_1$.
Efn Oi Os M ($secl_2$ eqs $secl_1$) =
(**if** repPO Os (Lsfn M $secl_2$) (Lsfn M $secl_1$) **then** $\mathcal{U}$(:'v)
else { }) ∩
if repPO Os (Lsfn M $secl_1$) (Lsfn M $secl_2$) **then** $\mathcal{U}$(:'v)
else { }) ∧
(∀ Oi Os M $numExp_1$ $numExp_2$.
Efn Oi Os M ($numExp_1$ eqn $numExp_2$) =
if $numExp_1$ = $numExp_2$ **then** $\mathcal{U}$(:'v) **else** { }) ∧
(∀ Oi Os M $numExp_1$ $numExp_2$.
Efn Oi Os M ($numExp_1$ lte $numExp_2$) =
if $numExp_1$ ≤ $numExp_2$ **then** $\mathcal{U}$(:'v) **else** { }) ∧
∀ Oi Os M $numExp_1$ $numExp_2$.
Efn Oi Os M ($numExp_1$ lt $numExp_2$) =
if $numExp_1$ < $numExp_2$ **then** $\mathcal{U}$(:'v) **else** { }

[Jext_def]

⊢ (∀ J s. Jext J (Name s) = J s) ∧
(∀ J P_1 P_2.
Jext J (P_1 meet P_2) = Jext J P_1 RUNION Jext J P_2) ∧
∀ J P_1 P_2. Jext J (P_1 quoting P_2) = Jext J P_2 O Jext J P_1

[Lifn_def]

⊢ (∀ M l. Lifn M (iLab l) = l) ∧
∀ M $name$. Lifn M (il $name$) = imapKS M $name$

[Lsfn_def]

$\vdash$ ($\forall M\ l$. Lsfn M (sLab l) = l) $\wedge$
$\forall M\ name$. Lsfn M (sl $name$) = smapKS M $name$

2.2 Theorems

[andf_def]

$\vdash \forall Oi\ Os\ M\ f_1\ f_2$.
Efn Oi Os M (f_1 andf f_2) = Efn Oi Os M f_1 $\cap$ Efn Oi Os M f_2

[controls_def]

$\vdash \forall Oi\ Os\ M\ P\ f$.
Efn Oi Os M (P controls f) =
$\mathcal{U}$(:'v) DIFF $\{w$ | Jext (jKS M) P $w \subseteq$ Efn Oi Os M $f\}$ $\cup$
Efn Oi Os M f

[controls_says]

$\vdash \forall M\ P\ f$.
Efn Oi Os M (P controls f) = Efn Oi Os M (P says f impf f)

[domi_def]

$\vdash \forall Oi\ Os\ M\ intl_1\ intl_2$.
Efn Oi Os M ($intl_1$ domi $intl_2$) =
if repPO Oi (Lifn M $intl_2$) (Lifn M $intl_1$) **then** $\mathcal{U}$(:'v)
else $\{\}$

[doms_def]

$\vdash \forall Oi\ Os\ M\ secl_1\ secl_2$.
Efn Oi Os M ($secl_1$ doms $secl_2$) =
if repPO Os (Lsfn M $secl_2$) (Lsfn M $secl_1$) **then** $\mathcal{U}$(:'v)
else $\{\}$

[eqf_def]

$\vdash \forall Oi\ Os\ M\ f_1\ f_2$.
Efn Oi Os M (f_1 eqf f_2) =
($\mathcal{U}$(:'v) DIFF Efn Oi Os M f_1 $\cup$ Efn Oi Os M f_2) $\cap$
($\mathcal{U}$(:'v) DIFF Efn Oi Os M f_2 $\cup$ Efn Oi Os M f_1)

[eqf_impf]

$\vdash \forall M\ f_1\ f_2$.
Efn Oi Os M (f_1 eqf f_2) =
Efn Oi Os M ((f_1 impf f_2) andf (f_2 impf f_1))

[eqi_def]

$\vdash \forall\, Oi\ Os\ M\ intl_2\ intl_1 .$
`Efn` $Oi\ Os\ M$ `(`$intl_2$ `eqi` $intl_1$`) =`
`(`**if** `repPO` Oi `(Lifn` $M\ intl_2$`)` `(Lifn` $M\ intl_1$`)` **then** $\mathcal{U}$`(:'v)`
else $\{\}$`)` $\cap$
if `repPO` Oi `(Lifn` $M\ intl_1$`)` `(Lifn` $M\ intl_2$`)` **then** $\mathcal{U}$`(:'v)`
else $\{\}$

[eqi_domi]

$\vdash \forall\, M\ intL_1\ intL_2 .$
`Efn` $Oi\ Os\ M$ `(`$intL_1$ `eqi` $intL_2$`) =`
`Efn` $Oi\ Os\ M$ `(`$intL_2$ `domi` $intL_1$ `andf` $intL_1$ `domi` $intL_2$`)`

[eqn_def]

$\vdash \forall\, Oi\ Os\ M\ numExp_1\ numExp_2 .$
`Efn` $Oi\ Os\ M$ `(`$numExp_1$ `eqn` $numExp_2$`) =`
if $numExp_1$ `=` $numExp_2$ **then** $\mathcal{U}$`(:'v)` **else** $\{\}$

[eqs_def]

$\vdash \forall\, Oi\ Os\ M\ secl_2\ secl_1 .$
`Efn` $Oi\ Os\ M$ `(`$secl_2$ `eqs` $secl_1$`) =`
`(`**if** `repPO` Os `(Lsfn` $M\ secl_2$`)` `(Lsfn` $M\ secl_1$`)` **then** $\mathcal{U}$`(:'v)`
else $\{\}$`)` $\cap$
if `repPO` Os `(Lsfn` $M\ secl_1$`)` `(Lsfn` $M\ secl_2$`)` **then** $\mathcal{U}$`(:'v)`
else $\{\}$

[eqs_doms]

$\vdash \forall\, M\ secL_1\ secL_2 .$
`Efn` $Oi\ Os\ M$ `(`$secL_1$ `eqs` $secL_2$`) =`
`Efn` $Oi\ Os\ M$ `(`$secL_2$ `doms` $secL_1$ `andf` $secL_1$ `doms` $secL_2$`)`

[FF_def]

$\vdash \forall\, Oi\ Os\ M .$ `Efn` $Oi\ Os\ M$ `FF =` $\{\}$

[impf_def]

$\vdash \forall\, Oi\ Os\ M\ f_1\ f_2 .$
`Efn` $Oi\ Os\ M$ `(`f_1 `impf` f_2`) =`
$\mathcal{U}$`(:'v) DIFF Efn` $Oi\ Os\ M\ f_1 \cup$ `Efn` $Oi\ Os\ M\ f_2$

[lt_def]

$\vdash \forall\, Oi\ Os\ M\ numExp_1\ numExp_2 .$
`Efn` $Oi\ Os\ M$ `(`$numExp_1$ `lt` $numExp_2$`) =`
if $numExp_1 < numExp_2$ **then** $\mathcal{U}$`(:'v)` **else** $\{\}$

[lte_def]

$\vdash \forall\, Oi\ Os\ M\ numExp_1\ numExp_2 .$
`Efn` $Oi\ Os\ M$ `(`$numExp_1$ `lte` $numExp_2$`) =`
if $numExp_1 \leq numExp_2$ **then** $\mathcal{U}$`(:'v)` **else** $\{\}$

[meet_def]

$\vdash \forall J\ P_1\ P_2.$ Jext J (P_1 meet P_2) = Jext J P_1 RUNION Jext J P_2

[name_def]

$\vdash \forall J\ s.$ Jext J (Name s) = J s

[notf_def]

$\vdash \forall Oi\ Os\ M\ f.$ Efn Oi Os M (notf f) = $\mathcal{U}$(:'v) DIFF Efn Oi Os M f

[orf_def]

$\vdash \forall Oi\ Os\ M\ f_1\ f_2.$
Efn Oi Os M (f_1 orf f_2) = Efn Oi Os M f_1 $\cup$ Efn Oi Os M f_2

[prop_def]

$\vdash \forall Oi\ Os\ M\ p.$ Efn Oi Os M (prop p) = intpKS M p

[quoting_def]

$\vdash \forall J\ P_1\ P_2.$ Jext J (P_1 quoting P_2) = Jext J P_2 O Jext J P_1

[reps_def]

$\vdash \forall Oi\ Os\ M\ P\ Q\ f.$
Efn Oi Os M (reps P Q f) =
$\mathcal{U}$(:'v) DIFF
$\{w$ | Jext (jKS M) (P quoting Q) $w \subseteq$ Efn Oi Os M $f\} \cup$
$\{w$ | Jext (jKS M) Q $w \subseteq$ Efn Oi Os M $f\}$

[says_def]

$\vdash \forall Oi\ Os\ M\ P\ f.$
Efn Oi Os M (P says f) =
$\{w$ | Jext (jKS M) P $w \subseteq$ Efn Oi Os M $f\}$

[speaks_for_def]

$\vdash \forall Oi\ Os\ M\ P\ Q.$
Efn Oi Os M (P speaks_for Q) =
if Jext (jKS M) Q RSUBSET Jext (jKS M) P **then** $\mathcal{U}$(:'v)
else $\{\}$

[TT_def]

$\vdash \forall Oi\ Os\ M.$ Efn Oi Os M TT = $\mathcal{U}$(:'v)

3 aclrules Theory

Built: 25 February 2018
Parent Theories: aclsemantics

3.1 Definitions

[sat_def]

$\vdash \forall M\ Oi\ Os\ f.\ (M, Oi, Os)\ \texttt{sat}\ f \iff (\texttt{Efn}\ Oi\ Os\ M\ f = \mathcal{U}(:\texttt{'world}))$

3.2 Theorems

[And_Says]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{meet}\ Q\ \texttt{says}\ f\ \texttt{eqf}\ P\ \texttt{says}\ f\ \texttt{andf}\ Q\ \texttt{says}\ f$

[And_Says_Eq]

$\vdash (M, Oi, Os)\ \texttt{sat}\ P\ \texttt{meet}\ Q\ \texttt{says}\ f \iff$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{says}\ f\ \texttt{andf}\ Q\ \texttt{says}\ f$

[and_says_lemma]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{meet}\ Q\ \texttt{says}\ f\ \texttt{impf}\ P\ \texttt{says}\ f\ \texttt{andf}\ Q\ \texttt{says}\ f$

[Controls_Eq]

$\vdash \forall M\ Oi\ Os\ P\ f.$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{controls}\ f \iff (M, Oi, Os)\ \texttt{sat}\ P\ \texttt{says}\ f\ \texttt{impf}\ f$

[DIFF_UNIV_SUBSET]

$\vdash (\mathcal{U}(:\texttt{'a})\ \texttt{DIFF}\ s \cup t = \mathcal{U}(:\texttt{'a})) \iff s \subseteq t$

[domi_antisymmetric]

$\vdash \forall M\ Oi\ Os\ l_1\ l_2.$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{domi}\ l_2 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ l_2\ \texttt{domi}\ l_1 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{eqi}\ l_2$

[domi_reflexive]

$\vdash \forall M\ Oi\ Os\ l.\ (M, Oi, Os)\ \texttt{sat}\ l\ \texttt{domi}\ l$

[domi_transitive]

$\vdash \forall M\ Oi\ Os\ l_1\ l_2\ l_3.$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{domi}\ l_2 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ l_2\ \texttt{domi}\ l_3 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{domi}\ l_3$

[doms_antisymmetric]

$\vdash \forall M\ Oi\ Os\ l_1\ l_2.$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{doms}\ l_2 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ l_2\ \texttt{doms}\ l_1 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{eqs}\ l_2$

[doms_reflexive]

$\vdash \forall M\ Oi\ Os\ l.\ (M, Oi, Os)\ \texttt{sat}\ l\ \texttt{doms}\ l$

[doms_transitive]

$\vdash \forall M\ Oi\ Os\ l_1\ l_2\ l_3.$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{doms}\ l_2 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ l_2\ \texttt{doms}\ l_3 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{doms}\ l_3$

[eqf_and_impf]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.$
$(M, Oi, Os)\ \texttt{sat}\ f_1\ \texttt{eqf}\ f_2 \iff$
$(M, Oi, Os)\ \texttt{sat}\ (f_1\ \texttt{impf}\ f_2)\ \texttt{andf}\ (f_2\ \texttt{impf}\ f_1)$

[eqf_andf1]

$\vdash \forall M\ Oi\ Os\ f\ f'\ g.$
$(M, Oi, Os)\ \texttt{sat}\ f\ \texttt{eqf}\ f' \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ f\ \texttt{andf}\ g \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ f'\ \texttt{andf}\ g$

[eqf_andf2]

$\vdash \forall M\ Oi\ Os\ f\ f'\ g.$
$(M, Oi, Os)\ \texttt{sat}\ f\ \texttt{eqf}\ f' \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ g\ \texttt{andf}\ f \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ g\ \texttt{andf}\ f'$

[eqf_controls]

$\vdash \forall M\ Oi\ Os\ P\ f\ f'.$
$(M, Oi, Os)\ \texttt{sat}\ f\ \texttt{eqf}\ f' \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{controls}\ f \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{controls}\ f'$

[eqf_eq]

$\vdash\ (\texttt{Efn}\ Oi\ Os\ M\ (f_1\ \texttt{eqf}\ f_2) = \mathcal{U}(\texttt{:'b})) \iff$
$(\texttt{Efn}\ Oi\ Os\ M\ f_1 = \texttt{Efn}\ Oi\ Os\ M\ f_2)$

[eqf_eqf1]

$\vdash \forall M\ Oi\ Os\ f\ f'\ g.$
$(M, Oi, Os)\ \texttt{sat}\ f\ \texttt{eqf}\ f' \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ f\ \texttt{eqf}\ g \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ f'\ \texttt{eqf}\ g$

[eqf_eqf2]

$\vdash \forall M\ Oi\ Os\ f\ f'\ g.$
$(M, Oi, Os)\ \texttt{sat}\ f\ \texttt{eqf}\ f' \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ g\ \texttt{eqf}\ f \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ g\ \texttt{eqf}\ f'$

[eqf_impf1]

$\vdash \forall M\ Oi\ Os\ f\ f'\ g.$
(M, Oi, Os) sat f eqf $f' \Rightarrow$
(M, Oi, Os) sat f impf $g \Rightarrow$
(M, Oi, Os) sat f' impf g

[eqf_impf2]

$\vdash \forall M\ Oi\ Os\ f\ f'\ g.$
(M, Oi, Os) sat f eqf $f' \Rightarrow$
(M, Oi, Os) sat g impf $f \Rightarrow$
(M, Oi, Os) sat g impf f'

[eqf_notf]

$\vdash \forall M\ Oi\ Os\ f\ f'.$
(M, Oi, Os) sat f eqf $f' \Rightarrow$
(M, Oi, Os) sat notf $f \Rightarrow$
(M, Oi, Os) sat notf f'

[eqf_orf1]

$\vdash \forall M\ Oi\ Os\ f\ f'\ g.$
(M, Oi, Os) sat f eqf $f' \Rightarrow$
(M, Oi, Os) sat f orf $g \Rightarrow$
(M, Oi, Os) sat f' orf g

[eqf_orf2]

$\vdash \forall M\ Oi\ Os\ f\ f'\ g.$
(M, Oi, Os) sat f eqf $f' \Rightarrow$
(M, Oi, Os) sat g orf $f \Rightarrow$
(M, Oi, Os) sat g orf f'

[eqf_reps]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f\ f'.$
(M, Oi, Os) sat f eqf $f' \Rightarrow$
(M, Oi, Os) sat reps $P\ Q\ f \Rightarrow$
(M, Oi, Os) sat reps $P\ Q\ f'$

[eqf_sat]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.$
(M, Oi, Os) sat f_1 eqf $f_2 \Rightarrow$
$((M, Oi, Os)$ sat $f_1 \iff (M, Oi, Os)$ sat $f_2)$

[eqf_says]

$\vdash \forall M\ Oi\ Os\ P\ f\ f'.$
(M, Oi, Os) sat f eqf $f' \Rightarrow$
(M, Oi, Os) sat P says $f \Rightarrow$
(M, Oi, Os) sat P says f'

[eqi_Eq]

$\vdash \forall M\ Oi\ Os\ l_1\ l_2.$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{eqi}\ l_2 \iff$
$(M, Oi, Os)\ \texttt{sat}\ l_2\ \texttt{domi}\ l_1\ \texttt{andf}\ l_1\ \texttt{domi}\ l_2$

[eqs_Eq]

$\vdash \forall M\ Oi\ Os\ l_1\ l_2.$
$(M, Oi, Os)\ \texttt{sat}\ l_1\ \texttt{eqs}\ l_2 \iff$
$(M, Oi, Os)\ \texttt{sat}\ l_2\ \texttt{doms}\ l_1\ \texttt{andf}\ l_1\ \texttt{doms}\ l_2$

[Idemp_Speaks_For]

$\vdash \forall M\ Oi\ Os\ P.\ (M, Oi, Os)\ \texttt{sat}\ P\ \texttt{speaks\_for}\ P$

[Image_cmp]

$\vdash \forall R_1\ R_2\ R_3\ u.\ (R_1\ \texttt{O}\ R_2)\ u \subseteq R_3 \iff R_2\ u \subseteq \{y\ |\ R_1\ y \subseteq R_3\}$

[Image_SUBSET]

$\vdash \forall R_1\ R_2.\ R_2\ \texttt{RSUBSET}\ R_1 \Rightarrow \forall w.\ R_2\ w \subseteq R_1\ w$

[Image_UNION]

$\vdash \forall R_1\ R_2\ w.\ (R_1\ \texttt{RUNION}\ R_2)\ w = R_1\ w \cup R_2\ w$

[INTER_EQ_UNIV]

$\vdash (s \cap t = \mathcal{U}(:\texttt{'a})) \iff (s = \mathcal{U}(:\texttt{'a})) \wedge (t = \mathcal{U}(:\texttt{'a}))$

[Modus_Ponens]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.$
$(M, Oi, Os)\ \texttt{sat}\ f_1 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ f_1\ \texttt{impf}\ f_2 \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ f_2$

[Mono_speaks_for]

$\vdash \forall M\ Oi\ Os\ P\ P'\ Q\ Q'.$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{speaks\_for}\ P' \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ Q\ \texttt{speaks\_for}\ Q' \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{quoting}\ Q\ \texttt{speaks\_for}\ P'\ \texttt{quoting}\ Q'$

[MP_Says]

$\vdash \forall M\ Oi\ Os\ P\ f_1\ f_2.$
$(M, Oi, Os)\ \texttt{sat}$
$P\ \texttt{says}\ (f_1\ \texttt{impf}\ f_2)\ \texttt{impf}\ P\ \texttt{says}\ f_1\ \texttt{impf}\ P\ \texttt{says}\ f_2$

[Quoting]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{quoting}\ Q\ \texttt{says}\ f\ \texttt{eqf}\ P\ \texttt{says}\ Q\ \texttt{says}\ f$

[Quoting_Eq]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
(M, Oi, Os) sat P quoting Q says $f \iff$
(M, Oi, Os) sat P says Q says f

[reps_def_lemma]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
Efn $Oi\ Os\ M$ (reps $P\ Q\ f$) =
Efn $Oi\ Os\ M$ (P quoting Q says f impf Q says f)

[Reps_Eq]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
(M, Oi, Os) sat reps $P\ Q\ f \iff$
(M, Oi, Os) sat P quoting Q says f impf Q says f

[sat_allworld]

$\vdash \forall M\ f.\ (M, Oi, Os)$ sat $f \iff \forall w.\ w \in$ Efn $Oi\ Os\ M\ f$

[sat_andf_eq_and_sat]

$\vdash (M, Oi, Os)$ sat f_1 andf $f_2 \iff$
(M, Oi, Os) sat $f_1 \wedge (M, Oi, Os)$ sat f_2

[sat_TT]

$\vdash (M, Oi, Os)$ sat TT

[Says]

$\vdash \forall M\ Oi\ Os\ P\ f.\ (M, Oi, Os)$ sat $f \Rightarrow (M, Oi, Os)$ sat P says f

[says_and_lemma]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
(M, Oi, Os) sat P says f andf Q says f impf P meet Q says f

[Speaks_For]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
(M, Oi, Os) sat P speaks_for Q impf P says f impf Q says f

[speaks_for_SUBSET]

$\vdash \forall R_3\ R_2\ R_1.$
R_2 RSUBSET $R_1 \Rightarrow \forall w.\ \{w \mid R_1\ w \subseteq R_3\} \subseteq \{w \mid R_2\ w \subseteq R_3\}$

[SUBSET_Image_SUBSET]

$\vdash \forall R_1\ R_2\ R_3.$
$(\forall w_1.\ R_2\ w_1 \subseteq R_1\ w_1) \Rightarrow$
$\forall w.\ \{w \mid R_1\ w \subseteq R_3\} \subseteq \{w \mid R_2\ w \subseteq R_3\}$

[Trans_Speaks_For]
$\vdash \forall M\ Oi\ Os\ P\ Q\ R.$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{speaks\_for}\ Q \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ Q\ \texttt{speaks\_for}\ R \Rightarrow$
$(M, Oi, Os)\ \texttt{sat}\ P\ \texttt{speaks\_for}\ R$

[UNIV_DIFF_SUBSET]
$\vdash \forall R_1\ R_2.\ R_1 \subseteq R_2 \Rightarrow (\mathcal{U}(:'a)\ \texttt{DIFF}\ R_1 \cup R_2 = \mathcal{U}(:'a))$

[world_and]
$\vdash \forall M\ Oi\ Os\ f_1\ f_2\ w.$
$w \in \texttt{Efn}\ Oi\ Os\ M\ (f_1\ \texttt{andf}\ f_2) \iff$
$w \in \texttt{Efn}\ Oi\ Os\ M\ f_1 \wedge w \in \texttt{Efn}\ Oi\ Os\ M\ f_2$

[world_eq]
$\vdash \forall M\ Oi\ Os\ f_1\ f_2\ w.$
$w \in \texttt{Efn}\ Oi\ Os\ M\ (f_1\ \texttt{eqf}\ f_2) \iff$
$(w \in \texttt{Efn}\ Oi\ Os\ M\ f_1 \iff w \in \texttt{Efn}\ Oi\ Os\ M\ f_2)$

[world_eqn]
$\vdash \forall M\ Oi\ Os\ n_1\ n_2\ w.\ w \in \texttt{Efn}\ Oi\ Os\ m\ (n_1\ \texttt{eqn}\ n_2) \iff (n_1 = n_2)$

[world_F]
$\vdash \forall M\ Oi\ Os\ w.\ w \notin \texttt{Efn}\ Oi\ Os\ M\ \texttt{FF}$

[world_imp]
$\vdash \forall M\ Oi\ Os\ f_1\ f_2\ w.$
$w \in \texttt{Efn}\ Oi\ Os\ M\ (f_1\ \texttt{impf}\ f_2) \iff$
$w \in \texttt{Efn}\ Oi\ Os\ M\ f_1 \Rightarrow w \in \texttt{Efn}\ Oi\ Os\ M\ f_2$

[world_lt]
$\vdash \forall M\ Oi\ Os\ n_1\ n_2\ w.\ w \in \texttt{Efn}\ Oi\ Os\ m\ (n_1\ \texttt{lt}\ n_2) \iff n_1 < n_2$

[world_lte]
$\vdash \forall M\ Oi\ Os\ n_1\ n_2\ w.\ w \in \texttt{Efn}\ Oi\ Os\ m\ (n_1\ \texttt{lte}\ n_2) \iff n_1 \leq n_2$

[world_not]
$\vdash \forall M\ Oi\ Os\ f\ w.\ w \in \texttt{Efn}\ Oi\ Os\ M\ (\texttt{notf}\ f) \iff w \notin \texttt{Efn}\ Oi\ Os\ M\ f$

[world_or]
$\vdash \forall M\ f_1\ f_2\ w.$
$w \in \texttt{Efn}\ Oi\ Os\ M\ (f_1\ \texttt{orf}\ f_2) \iff$
$w \in \texttt{Efn}\ Oi\ Os\ M\ f_1 \vee w \in \texttt{Efn}\ Oi\ Os\ M\ f_2$

[world_says]
$\vdash \forall M\ Oi\ Os\ P\ f\ w.$
$w \in \texttt{Efn}\ Oi\ Os\ M\ (P\ \texttt{says}\ f) \iff$
$\forall v.\ v \in \texttt{Jext}\ (\texttt{jKS}\ M)\ P\ w \Rightarrow v \in \texttt{Efn}\ Oi\ Os\ M\ f$

[world_T]
$\vdash \forall M\ Oi\ Os\ w.\ w \in \texttt{Efn}\ Oi\ Os\ M\ \texttt{TT}$

4 aclDrules Theory

Built: 25 February 2018
Parent Theories: aclrules

4.1 Theorems

[Conjunction]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.$
(M, Oi, Os) sat $f_1 \Rightarrow$
(M, Oi, Os) sat $f_2 \Rightarrow$
(M, Oi, Os) sat f_1 andf f_2

[Controls]

$\vdash \forall M\ Oi\ Os\ P\ f.$
(M, Oi, Os) sat P says $f \Rightarrow$
(M, Oi, Os) sat P controls $f \Rightarrow$
(M, Oi, Os) sat f

[Derived_Controls]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
(M, Oi, Os) sat P speaks_for $Q \Rightarrow$
(M, Oi, Os) sat Q controls $f \Rightarrow$
(M, Oi, Os) sat P controls f

[Derived_Speaks_For]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
(M, Oi, Os) sat P speaks_for $Q \Rightarrow$
(M, Oi, Os) sat P says $f \Rightarrow$
(M, Oi, Os) sat Q says f

[Disjunction1]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.\ (M, Oi, Os)$ sat $f_1 \Rightarrow (M, Oi, Os)$ sat f_1 orf f_2

[Disjunction2]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.\ (M, Oi, Os)$ sat $f_2 \Rightarrow (M, Oi, Os)$ sat f_1 orf f_2

[Disjunctive_Syllogism]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.$
(M, Oi, Os) sat f_1 orf $f_2 \Rightarrow$
(M, Oi, Os) sat notf $f_1 \Rightarrow$
(M, Oi, Os) sat f_2

[Double_Negation]

$\vdash \forall M\ Oi\ Os\ f.\ (M, Oi, Os)$ sat notf (notf f) $\Rightarrow (M, Oi, Os)$ sat f

[eqn_eqn]

$\vdash$ (M, Oi, Os) sat c_1 eqn n_1 $\Rightarrow$
(M, Oi, Os) sat c_2 eqn n_2 $\Rightarrow$
(M, Oi, Os) sat n_1 eqn n_2 $\Rightarrow$
(M, Oi, Os) sat c_1 eqn c_2

[eqn_lt]

$\vdash$ (M, Oi, Os) sat c_1 eqn n_1 $\Rightarrow$
(M, Oi, Os) sat c_2 eqn n_2 $\Rightarrow$
(M, Oi, Os) sat n_1 lt n_2 $\Rightarrow$
(M, Oi, Os) sat c_1 lt c_2

[eqn_lte]

$\vdash$ (M, Oi, Os) sat c_1 eqn n_1 $\Rightarrow$
(M, Oi, Os) sat c_2 eqn n_2 $\Rightarrow$
(M, Oi, Os) sat n_1 lte n_2 $\Rightarrow$
(M, Oi, Os) sat c_1 lte c_2

[Hypothetical_Syllogism]

$\vdash$ $\forall M\ Oi\ Os\ f_1\ f_2\ f_3$.
(M, Oi, Os) sat f_1 impf f_2 $\Rightarrow$
(M, Oi, Os) sat f_2 impf f_3 $\Rightarrow$
(M, Oi, Os) sat f_1 impf f_3

[il_domi]

$\vdash$ $\forall M\ Oi\ Os\ P\ Q\ l_1\ l_2$.
(M, Oi, Os) sat il P eqi l_1 $\Rightarrow$
(M, Oi, Os) sat il Q eqi l_2 $\Rightarrow$
(M, Oi, Os) sat l_2 domi l_1 $\Rightarrow$
(M, Oi, Os) sat il Q domi il P

[INTER_EQ_UNIV]

$\vdash$ $\forall s_1\ s_2$. $(s_1 \cap s_2 = \mathcal{U}(:'a)) \iff (s_1 = \mathcal{U}(:'a)) \wedge (s_2 = \mathcal{U}(:'a))$

[Modus_Tollens]

$\vdash$ $\forall M\ Oi\ Os\ f_1\ f_2$.
(M, Oi, Os) sat f_1 impf f_2 $\Rightarrow$
(M, Oi, Os) sat notf f_2 $\Rightarrow$
(M, Oi, Os) sat notf f_1

[Rep_Controls_Eq]

$\vdash$ $\forall M\ Oi\ Os\ A\ B\ f$.
(M, Oi, Os) sat reps $A\ B\ f$ $\iff$
(M, Oi, Os) sat A controls B says f

[Rep_Says]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
(M, Oi, Os) sat reps $P\ Q\ f \Rightarrow$
(M, Oi, Os) sat P quoting Q says $f \Rightarrow$
(M, Oi, Os) sat Q says f

[Reps]

$\vdash \forall M\ Oi\ Os\ P\ Q\ f.$
(M, Oi, Os) sat reps $P\ Q\ f \Rightarrow$
(M, Oi, Os) sat P quoting Q says $f \Rightarrow$
(M, Oi, Os) sat Q controls $f \Rightarrow$
(M, Oi, Os) sat f

[Says_Simplification1]

$\vdash \forall M\ Oi\ Os\ P\ f_1\ f_2.$
(M, Oi, Os) sat P says $(f_1$ andf $f_2) \Rightarrow (M, Oi, Os)$ sat P says f_1

[Says_Simplification2]

$\vdash \forall M\ Oi\ Os\ P\ f_1\ f_2.$
(M, Oi, Os) sat P says $(f_1$ andf $f_2) \Rightarrow (M, Oi, Os)$ sat P says f_2

[Simplification1]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.\ (M, Oi, Os)$ sat f_1 andf $f_2 \Rightarrow (M, Oi, Os)$ sat f_1

[Simplification2]

$\vdash \forall M\ Oi\ Os\ f_1\ f_2.\ (M, Oi, Os)$ sat f_1 andf $f_2 \Rightarrow (M, Oi, Os)$ sat f_2

[sl_doms]

$\vdash \forall M\ Oi\ Os\ P\ Q\ l_1\ l_2.$
(M, Oi, Os) sat sl P eqs $l_1 \Rightarrow$
(M, Oi, Os) sat sl Q eqs $l_2 \Rightarrow$
(M, Oi, Os) sat l_2 doms $l_1 \Rightarrow$
(M, Oi, Os) sat sl Q doms sl P

Index

Appendix B

Secure State Machine & Patrol Base Operations: Pretty-Printed Theories

Contents

1 OMNIType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

1.1 Datatypes

command = ESCc escCommand | SLc 'slCommand

escCommand = returnToBase | changeMission | resupply
| reactToContact

escOutput = ReturnToBase | ChangeMission | Resupply
| ReactToContact

escState = RTB | CM | RESUPPLY | RTC

output = ESCo escOutput | SLo 'slOutput

principal = SR 'stateRole

state = ESCs escState | SLs 'slState

1.2 Theorems

[command_distinct_clauses]

$\vdash \forall a'\ a.$ ESCc $a \neq$ SLc a'

[command_one_one]

$\vdash$ $(\forall a\ a'.$ (ESCc a = ESCc $a') \iff (a = a')) \wedge$
$\forall a\ a'.$ (SLc a = SLc $a') \iff (a = a')$

[escCommand_distinct_clauses]

$\vdash$ returnToBase $\neq$ changeMission $\wedge$ returnToBase $\neq$ resupply $\wedge$
returnToBase $\neq$ reactToContact $\wedge$ changeMission $\neq$ resupply $\wedge$
changeMission $\neq$ reactToContact $\wedge$ resupply $\neq$ reactToContact

[escOutput_distinct_clauses]

$\vdash$ ReturnToBase $\neq$ ChangeMission $\wedge$ ReturnToBase $\neq$ Resupply $\wedge$
ReturnToBase $\neq$ ReactToContact $\wedge$ ChangeMission $\neq$ Resupply $\wedge$
ChangeMission $\neq$ ReactToContact $\wedge$ Resupply $\neq$ ReactToContact

[escState_distinct_clauses]

$\vdash$ RTB $\neq$ CM $\wedge$ RTB $\neq$ RESUPPLY $\wedge$ RTB $\neq$ RTC $\wedge$ CM $\neq$ RESUPPLY $\wedge$
CM $\neq$ RTC $\wedge$ RESUPPLY $\neq$ RTC

[output_distinct_clauses]

$\vdash \forall a'\ a.\ \texttt{ESCo}\ a \neq \texttt{SLo}\ a'$

[output_one_one]

$\vdash (\forall a\ a'.\ (\texttt{ESCo}\ a = \texttt{ESCo}\ a') \iff (a = a')) \wedge$
$\forall a\ a'.\ (\texttt{SLo}\ a = \texttt{SLo}\ a') \iff (a = a')$

[principal_one_one]

$\vdash \forall a\ a'.\ (\texttt{SR}\ a = \texttt{SR}\ a') \iff (a = a')$

[state_distinct_clauses]

$\vdash \forall a'\ a.\ \texttt{ESCs}\ a \neq \texttt{SLs}\ a'$

[state_one_one]

$\vdash (\forall a\ a'.\ (\texttt{ESCs}\ a = \texttt{ESCs}\ a') \iff (a = a')) \wedge$
$\forall a\ a'.\ (\texttt{SLs}\ a = \texttt{SLs}\ a') \iff (a = a')$

2 ssm11 Theory

Built: 13 May 2018
Parent Theories: satList

2.1 Datatypes

```
configuration =
    CFG (('command order, 'principal, 'd, 'e) Form -> bool)
        ('state -> ('command order, 'principal, 'd, 'e) Form)
        (('command order, 'principal, 'd, 'e) Form list)
        (('command order, 'principal, 'd, 'e) Form list) 'state
        ('output list)

order = SOME 'command | NONE

trType = discard 'command | trap 'command | exec 'command
```

2.2 Definitions

[TR_def]

$\vdash \texttt{TR} =$
$(\lambda a_0\ a_1\ a_2\ a_3.$
$\forall TR'.$
$(\forall a_0\ a_1\ a_2\ a_3.$
$(\exists authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s$
$securityContext\ stateInterp\ cmd\ ins\ outs.$
$(a_0 = (M, Oi, Os)) \wedge (a_1 = \texttt{exec}\ cmd) \wedge$
$(a_2 =$

```
          CFG authenticationTest stateInterp
            securityContext (P says prop (SOME cmd)::ins) s
            outs) ∧
         (a3 =
          CFG authenticationTest stateInterp
            securityContext ins (NS s (exec cmd))
            (Out s (exec cmd)::outs)) ∧
         authenticationTest (P says prop (SOME cmd)) ∧
         CFGInterpret (M,Oi,Os)
           (CFG authenticationTest stateInterp
              securityContext (P says prop (SOME cmd)::ins)
              s outs)) ∨
      (∃ authenticationTest P NS M Oi Os Out s
         securityContext stateInterp cmd ins outs.
         (a0 = (M,Oi,Os)) ∧ (a1 = trap cmd) ∧
         (a2 =
          CFG authenticationTest stateInterp
            securityContext (P says prop (SOME cmd)::ins) s
            outs) ∧
         (a3 =
          CFG authenticationTest stateInterp
            securityContext ins (NS s (trap cmd))
            (Out s (trap cmd)::outs)) ∧
         authenticationTest (P says prop (SOME cmd)) ∧
         CFGInterpret (M,Oi,Os)
           (CFG authenticationTest stateInterp
              securityContext (P says prop (SOME cmd)::ins)
              s outs)) ∨
      (∃ authenticationTest NS M Oi Os Out s securityContext
         stateInterp cmd x ins outs.
         (a0 = (M,Oi,Os)) ∧ (a1 = discard cmd) ∧
         (a2 =
          CFG authenticationTest stateInterp
            securityContext (x::ins) s outs) ∧
         (a3 =
          CFG authenticationTest stateInterp
            securityContext ins (NS s (discard cmd))
            (Out s (discard cmd)::outs)) ∧
         ¬authenticationTest x) ⇒
      TR' a0 a1 a2 a3) ⇒
   TR' a0 a1 a2 a3)
```

2.3 Theorems

[CFGInterpret_def]

```
⊢ CFGInterpret (M,Oi,Os)
    (CFG authenticationTest stateInterp securityContext
       (input::ins) state outputStream) ⟺
```

(M, Oi, Os) `satList` *securityContext* $\wedge$ (M, Oi, Os) `sat` *input* $\wedge$
(M, Oi, Os) `sat` *stateInterp state*

[CFGInterpret_ind]

$\vdash \forall P.$
$(\forall M\ Oi\ Os\ authenticationTest\ stateInterp\ securityContext$
input ins state outputStream.
$P\ (M, Oi, Os)$
$(\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext$
$(input :: ins)\ state\ outputStream)) \wedge$
$(\forall v_{15}\ v_{10}\ v_{11}\ v_{12}\ v_{13}\ v_{14}.$
$P\ v_{15}\ (\texttt{CFG}\ v_{10}\ v_{11}\ v_{12}\ []\ v_{13}\ v_{14})) \Rightarrow$
$\forall v\ v_1\ v_2\ v_3.\ P\ (v, v_1, v_2)\ v_3$

[configuration_one_one]

$\vdash \forall a_0\ a_1\ a_2\ a_3\ a_4\ a_5\ a_0'\ a_1'\ a_2'\ a_3'\ a_4'\ a_5'.$
$(\texttt{CFG}\ a_0\ a_1\ a_2\ a_3\ a_4\ a_5 = \texttt{CFG}\ a_0'\ a_1'\ a_2'\ a_3'\ a_4'\ a_5') \iff$
$(a_0 = a_0') \wedge (a_1 = a_1') \wedge (a_2 = a_2') \wedge (a_3 = a_3') \wedge$
$(a_4 = a_4') \wedge (a_5 = a_5')$

[order_distinct_clauses]

$\vdash \forall a.$ `SOME` $a \neq$ `NONE`

[order_one_one]

$\vdash \forall a\ a'.$ `(SOME` a `= SOME` a'`)` $\iff$ $(a = a')$

[TR_cases]

$\vdash \forall a_0\ a_1\ a_2\ a_3.$
`TR` $a_0\ a_1\ a_2\ a_3 \iff$
$(\exists authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s\ securityContext$
stateInterp cmd ins outs.
$(a_0 = (M, Oi, Os)) \wedge (a_1 = \texttt{exec}\ cmd) \wedge$
$(a_2 =$
$\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext$
$(P\ \texttt{says prop (SOME}\ cmd) :: ins)\ s\ outs) \wedge$
$(a_3 =$
$\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\ ins$
$(NS\ s\ (\texttt{exec}\ cmd))\ (Out\ s\ (\texttt{exec}\ cmd) :: outs)) \wedge$
$authenticationTest\ (P\ \texttt{says prop (SOME}\ cmd))\ \wedge$
$\texttt{CFGInterpret}\ (M, Oi, Os)$
$(\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext$
$(P\ \texttt{says prop (SOME}\ cmd) :: ins)\ s\ outs)) \vee$
$(\exists authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s\ securityContext$
stateInterp cmd ins outs.
$(a_0 = (M, Oi, Os)) \wedge (a_1 = \texttt{trap}\ cmd) \wedge$
$(a_2 =$
$\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext$
$(P\ \texttt{says prop (SOME}\ cmd) :: ins)\ s\ outs) \wedge$

$(a_3 =$
$\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}\ \mathit{ins}$
$(\mathit{NS}\ s\ (\texttt{trap}\ \mathit{cmd}))\ (\mathit{Out}\ s\ (\texttt{trap}\ \mathit{cmd})::\mathit{outs})) \wedge$
$\mathit{authenticationTest}\ (P\ \texttt{says prop (SOME}\ \mathit{cmd})) \wedge$
$\texttt{CFGInterpret}\ (M, Oi, Os)$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}$
$(P\ \texttt{says prop (SOME}\ \mathit{cmd})::\mathit{ins})\ s\ \mathit{outs})) \vee$
$\exists\, \mathit{authenticationTest}\ \mathit{NS}\ M\ \mathit{Oi}\ \mathit{Os}\ \mathit{Out}\ s\ \mathit{securityContext}$
$\mathit{stateInterp}\ \mathit{cmd}\ x\ \mathit{ins}\ \mathit{outs}.$
$(a_0 = (M, Oi, Os)) \wedge (a_1 = \texttt{discard}\ \mathit{cmd}) \wedge$
$(a_2 =$
$\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}$
$(x::\mathit{ins})\ s\ \mathit{outs}) \wedge$
$(a_3 =$
$\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}\ \mathit{ins}$
$(\mathit{NS}\ s\ (\texttt{discard}\ \mathit{cmd}))\ (\mathit{Out}\ s\ (\texttt{discard}\ \mathit{cmd})::\mathit{outs})) \wedge$
$\neg \mathit{authenticationTest}\ x$

[TR_discard_cmd_rule]

$\vdash \texttt{TR}\ (M, Oi, Os)\ (\texttt{discard}\ \mathit{cmd})$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}$
$(x::\mathit{ins})\ s\ \mathit{outs})$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}\ \mathit{ins}$
$(\mathit{NS}\ s\ (\texttt{discard}\ \mathit{cmd}))\ (\mathit{Out}\ s\ (\texttt{discard}\ \mathit{cmd})::\mathit{outs})) \iff$
$\neg \mathit{authenticationTest}\ x$

[TR_EQ_rules_thm]

$\vdash (\texttt{TR}\ (M, Oi, Os)\ (\texttt{exec}\ \mathit{cmd})$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}$
$(P\ \texttt{says prop (SOME}\ \mathit{cmd})::\mathit{ins})\ s\ \mathit{outs})$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}\ \mathit{ins}$
$(\mathit{NS}\ s\ (\texttt{exec}\ \mathit{cmd}))\ (\mathit{Out}\ s\ (\texttt{exec}\ \mathit{cmd})::\mathit{outs})) \iff$
$\mathit{authenticationTest}\ (P\ \texttt{says prop (SOME}\ \mathit{cmd})) \wedge$
$\texttt{CFGInterpret}\ (M, Oi, Os)$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}$
$(P\ \texttt{says prop (SOME}\ \mathit{cmd})::\mathit{ins})\ s\ \mathit{outs})) \wedge$
$(\texttt{TR}\ (M, Oi, Os)\ (\texttt{trap}\ \mathit{cmd})$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}$
$(P\ \texttt{says prop (SOME}\ \mathit{cmd})::\mathit{ins})\ s\ \mathit{outs})$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}\ \mathit{ins}$
$(\mathit{NS}\ s\ (\texttt{trap}\ \mathit{cmd}))\ (\mathit{Out}\ s\ (\texttt{trap}\ \mathit{cmd})::\mathit{outs})) \iff$
$\mathit{authenticationTest}\ (P\ \texttt{says prop (SOME}\ \mathit{cmd})) \wedge$
$\texttt{CFGInterpret}\ (M, Oi, Os)$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}$
$(P\ \texttt{says prop (SOME}\ \mathit{cmd})::\mathit{ins})\ s\ \mathit{outs})) \wedge$
$(\texttt{TR}\ (M, Oi, Os)\ (\texttt{discard}\ \mathit{cmd})$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}$
$(x::\mathit{ins})\ s\ \mathit{outs})$
$(\texttt{CFG}\ \mathit{authenticationTest}\ \mathit{stateInterp}\ \mathit{securityContext}\ \mathit{ins}$

$$(NS\ s\ (\texttt{discard}\ cmd))\ (Out\ s\ (\texttt{discard}\ cmd)::outs)) \iff \neg authenticationTest\ x)$$

[TR_exec_cmd_rule]

$$\begin{array}{l}\vdash \forall\, authenticationTest\ securityContext\ stateInterp\ P\ cmd\ ins\ s\ outs.\\ (\forall\, M\ Oi\ Os.\\ \quad \texttt{CFGInterpret}\ (M,Oi,Os)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd)::ins)\ s\ outs) \Rightarrow\\ \quad (M,Oi,Os)\ \texttt{sat prop (SOME}\ cmd)) \Rightarrow\\ \forall\, NS\ Out\ M\ Oi\ Os.\\ \quad \texttt{TR}\ (M,Oi,Os)\ (\texttt{exec}\ cmd)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd)::ins)\ s\ outs)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\ ins\\ \quad\quad\quad (NS\ s\ (\texttt{exec}\ cmd))\ (Out\ s\ (\texttt{exec}\ cmd)::outs)) \iff\\ \quad authenticationTest\ (P\ \texttt{says prop (SOME}\ cmd)) \wedge\\ \quad \texttt{CFGInterpret}\ (M,Oi,Os)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd)::ins)\ s\ outs) \wedge\\ \quad (M,Oi,Os)\ \texttt{sat prop (SOME}\ cmd)\end{array}$$

[TR_ind]

$$\begin{array}{l}\vdash \forall\, TR'.\\ (\forall\, authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s\ securityContext\\ \quad stateInterp\ cmd\ ins\ outs.\\ \quad authenticationTest\ (P\ \texttt{says prop (SOME}\ cmd)) \wedge\\ \quad \texttt{CFGInterpret}\ (M,Oi,Os)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd)::ins)\ s\ outs) \Rightarrow\\ \quad TR'\ (M,Oi,Os)\ (\texttt{exec}\ cmd)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd)::ins)\ s\ outs)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad ins\ (NS\ s\ (\texttt{exec}\ cmd))\ (Out\ s\ (\texttt{exec}\ cmd)::outs))) \wedge\\ (\forall\, authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s\ securityContext\\ \quad stateInterp\ cmd\ ins\ outs.\\ \quad authenticationTest\ (P\ \texttt{says prop (SOME}\ cmd)) \wedge\\ \quad \texttt{CFGInterpret}\ (M,Oi,Os)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd)::ins)\ s\ outs) \Rightarrow\\ \quad TR'\ (M,Oi,Os)\ (\texttt{trap}\ cmd)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd)::ins)\ s\ outs)\\ \quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\\ \quad\quad\quad ins\ (NS\ s\ (\texttt{trap}\ cmd))\ (Out\ s\ (\texttt{trap}\ cmd)::outs))) \wedge\\ (\forall\, authenticationTest\ NS\ M\ Oi\ Os\ Out\ s\ securityContext\\ \quad stateInterp\ cmd\ x\ ins\ outs.\end{array}$$

$\neg authenticationTest\ x \Rightarrow$
$TR'\ (M, Oi, Os)$ (discard cmd)
(CFG $authenticationTest\ stateInterp\ securityContext$
(x::ins) $s\ outs$)
(CFG $authenticationTest\ stateInterp\ securityContext$
ins ($NS\ s$ (discard cmd))
($Out\ s$ (discard cmd)::$outs$))) $\Rightarrow$
$\forall a_0\ a_1\ a_2\ a_3.$ TR $a_0\ a_1\ a_2\ a_3 \Rightarrow TR'\ a_0\ a_1\ a_2\ a_3$

[TR_rules]

$\vdash$ ($\forall$ $authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s\ securityContext$
$stateInterp\ cmd\ ins\ outs.$
$authenticationTest$ (P says prop (SOME cmd)) $\wedge$
CFGInterpret (M, Oi, Os)
(CFG $authenticationTest\ stateInterp\ securityContext$
(P says prop (SOME cmd)::ins) $s\ outs$) $\Rightarrow$
TR (M, Oi, Os) (exec cmd)
(CFG $authenticationTest\ stateInterp\ securityContext$
(P says prop (SOME cmd)::ins) $s\ outs$)
(CFG $authenticationTest\ stateInterp\ securityContext\ ins$
($NS\ s$ (exec cmd)) ($Out\ s$ (exec cmd)::$outs$))) $\wedge$
($\forall$ $authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s\ securityContext$
$stateInterp\ cmd\ ins\ outs.$
$authenticationTest$ (P says prop (SOME cmd)) $\wedge$
CFGInterpret (M, Oi, Os)
(CFG $authenticationTest\ stateInterp\ securityContext$
(P says prop (SOME cmd)::ins) $s\ outs$) $\Rightarrow$
TR (M, Oi, Os) (trap cmd)
(CFG $authenticationTest\ stateInterp\ securityContext$
(P says prop (SOME cmd)::ins) $s\ outs$)
(CFG $authenticationTest\ stateInterp\ securityContext\ ins$
($NS\ s$ (trap cmd)) ($Out\ s$ (trap cmd)::$outs$))) $\wedge$
$\forall$ $authenticationTest\ NS\ M\ Oi\ Os\ Out\ s\ securityContext$
$stateInterp\ cmd\ x\ ins\ outs.$
$\neg authenticationTest\ x \Rightarrow$
TR (M, Oi, Os) (discard cmd)
(CFG $authenticationTest\ stateInterp\ securityContext$
(x::ins) $s\ outs$)
(CFG $authenticationTest\ stateInterp\ securityContext\ ins$
($NS\ s$ (discard cmd)) ($Out\ s$ (discard cmd)::$outs$))

[TR_strongind]

$\vdash \forall TR'.$
($\forall$ $authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s\ securityContext$
$stateInterp\ cmd\ ins\ outs.$
$authenticationTest$ (P says prop (SOME cmd)) $\wedge$
CFGInterpret (M, Oi, Os)
(CFG $authenticationTest\ stateInterp\ securityContext$
(P says prop (SOME cmd)::ins) $s\ outs$) $\Rightarrow$

$$
\begin{array}{l}
\quad TR'\ (M, Oi, Os)\ (\texttt{exec}\ cmd) \\
\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd\texttt{)::}ins)\ s\ outs) \\
\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad ins\ (NS\ s\ (\texttt{exec}\ cmd))\ (Out\ s\ (\texttt{exec}\ cmd)\texttt{::}outs))) \wedge \\
(\forall\, authenticationTest\ P\ NS\ M\ Oi\ Os\ Out\ s\ securityContext \\
\quad stateInterp\ cmd\ ins\ outs. \\
\quad authenticationTest\ (P\ \texttt{says prop (SOME}\ cmd\texttt{)}) \wedge \\
\quad \texttt{CFGInterpret}\ (M, Oi, Os) \\
\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd\texttt{)::}ins)\ s\ outs) \Rightarrow \\
\quad TR'\ (M, Oi, Os)\ (\texttt{trap}\ cmd) \\
\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd\texttt{)::}ins)\ s\ outs) \\
\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad ins\ (NS\ s\ (\texttt{trap}\ cmd))\ (Out\ s\ (\texttt{trap}\ cmd)\texttt{::}outs))) \wedge \\
(\forall\, authenticationTest\ NS\ M\ Oi\ Os\ Out\ s\ securityContext \\
\quad stateInterp\ cmd\ x\ ins\ outs. \\
\quad \neg authenticationTest\ x \Rightarrow \\
\quad TR'\ (M, Oi, Os)\ (\texttt{discard}\ cmd) \\
\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad (x\texttt{::}ins)\ s\ outs) \\
\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad ins\ (NS\ s\ (\texttt{discard}\ cmd)) \\
\quad\quad\quad (Out\ s\ (\texttt{discard}\ cmd)\texttt{::}outs))) \Rightarrow \\
\forall\, a_0\ a_1\ a_2\ a_3.\ \texttt{TR}\ a_0\ a_1\ a_2\ a_3 \Rightarrow TR'\ a_0\ a_1\ a_2\ a_3
\end{array}
$$

[TR_trap_cmd_rule]

$$
\begin{array}{l}
\vdash \forall\, authenticationTest\ stateInterp\ securityContext\ P\ cmd\ ins\ s \\
\quad outs. \\
\quad (\forall\, M\ Oi\ Os. \\
\quad\quad \texttt{CFGInterpret}\ (M, Oi, Os) \\
\quad\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd\texttt{)::}ins)\ s\ outs) \Rightarrow \\
\quad\quad (M, Oi, Os)\ \texttt{sat prop NONE}) \Rightarrow \\
\quad \forall\, NS\ Out\ M\ Oi\ Os. \\
\quad\quad \texttt{TR}\ (M, Oi, Os)\ (\texttt{trap}\ cmd) \\
\quad\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd\texttt{)::}ins)\ s\ outs) \\
\quad\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext\ ins \\
\quad\quad\quad\quad (NS\ s\ (\texttt{trap}\ cmd))\ (Out\ s\ (\texttt{trap}\ cmd)\texttt{::}outs)) \iff \\
\quad\quad authenticationTest\ (P\ \texttt{says prop (SOME}\ cmd\texttt{)}) \wedge \\
\quad\quad \texttt{CFGInterpret}\ (M, Oi, Os) \\
\quad\quad\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext \\
\quad\quad\quad\quad (P\ \texttt{says prop (SOME}\ cmd\texttt{)::}ins)\ s\ outs) \wedge \\
\quad\quad (M, Oi, Os)\ \texttt{sat prop NONE}
\end{array}
$$

[TRrule0]

$$
\begin{array}{l}
\vdash \texttt{TR}\ (M, Oi, Os)\ (\texttt{exec}\ cmd) \\
\quad (\texttt{CFG}\ authenticationTest\ stateInterp\ securityContext
\end{array}
$$

```
      (P says prop (SOME cmd)::ins) s outs)
    (CFG authenticationTest stateInterp securityContext ins
      (NS s (exec cmd)) (Out s (exec cmd)::outs)) ⟺
  authenticationTest (P says prop (SOME cmd)) ∧
  CFGInterpret (M,Oi,Os)
    (CFG authenticationTest stateInterp securityContext
      (P says prop (SOME cmd)::ins) s outs)
```

[TRrule1]

```
⊢ TR (M,Oi,Os) (trap cmd)
    (CFG authenticationTest stateInterp securityContext
      (P says prop (SOME cmd)::ins) s outs)
    (CFG authenticationTest stateInterp securityContext ins
      (NS s (trap cmd)) (Out s (trap cmd)::outs)) ⟺
  authenticationTest (P says prop (SOME cmd)) ∧
  CFGInterpret (M,Oi,Os)
    (CFG authenticationTest stateInterp securityContext
      (P says prop (SOME cmd)::ins) s outs)
```

[trType_distinct_clauses]

```
⊢ (∀a' a. discard a ≠ trap a') ∧ (∀a' a. discard a ≠ exec a') ∧
  ∀a' a. trap a ≠ exec a'
```

[trType_one_one]

```
⊢ (∀a a'. (discard a = discard a') ⟺ (a = a')) ∧
  (∀a a'. (trap a = trap a') ⟺ (a = a')) ∧
  ∀a a'. (exec a = exec a') ⟺ (a = a')
```

3 ssm Theory

Built: 13 May 2018
Parent Theories: satList

3.1 Datatypes

```
configuration =
    CFG (('command option, 'principal, 'd, 'e) Form -> bool)
        ('state ->
         ('command option, 'principal, 'd, 'e) Form list ->
         ('command option, 'principal, 'd, 'e) Form list)
        (('command option, 'principal, 'd, 'e) Form list ->
         ('command option, 'principal, 'd, 'e) Form list)
        (('command option, 'principal, 'd, 'e) Form list list)
        'state ('output list)

trType = discard 'cmdlist | trap 'cmdlist | exec 'cmdlist
```

3.2 Definitions

[authenticationTest_def]

$$\vdash \forall\, elementTest\ x.\ \texttt{authenticationTest}\ elementTest\ x \iff \texttt{FOLDR}\ (\lambda p\ q.\ p \wedge q)\ \texttt{T}\ (\texttt{MAP}\ elementTest\ x)$$

[commandList_def]

$$\vdash \forall x.\ \texttt{commandList}\ x = \texttt{MAP extractCommand}\ x$$

[inputList_def]

$$\vdash \forall xs.\ \texttt{inputList}\ xs = \texttt{MAP extractInput}\ xs$$

[propCommandList_def]

$$\vdash \forall x.\ \texttt{propCommandList}\ x = \texttt{MAP extractPropCommand}\ x$$

[TR_def]

$$\begin{array}{l}
\vdash \texttt{TR} = \\
\quad (\lambda\, a_0\ a_1\ a_2\ a_3.\ \\
\qquad \forall\, TR'.\ \\
\qquad\quad (\forall\, a_0\ a_1\ a_2\ a_3.\ \\
\qquad\qquad (\exists\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins\ outs.\ \\
\qquad\qquad\quad (a_0 = (M, Oi, Os)) \wedge (a_1 = \texttt{exec (inputList}\ x\texttt{)}) \wedge \\
\qquad\qquad\quad (a_2 = \\
\qquad\qquad\quad\ \texttt{CFG}\ elementTest\ stateInterp\ context\ (x :: ins)\ s\ outs) \wedge \\
\qquad\qquad\quad (a_3 = \\
\qquad\qquad\quad\ \texttt{CFG}\ elementTest\ stateInterp\ context\ ins\ (NS\ s\ \texttt{(exec (inputList}\ x\texttt{))})\ (Out\ s\ \texttt{(exec (inputList}\ x\texttt{))} :: outs)) \wedge \\
\qquad\qquad\quad \texttt{authenticationTest}\ elementTest\ x \wedge \\
\qquad\qquad\quad \texttt{CFGInterpret}\ (M, Oi, Os)\ (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x :: ins)\ s\ outs)) \vee \\
\qquad\qquad (\exists\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins\ outs.\ \\
\qquad\qquad\quad (a_0 = (M, Oi, Os)) \wedge (a_1 = \texttt{trap (inputList}\ x\texttt{)}) \wedge \\
\qquad\qquad\quad (a_2 = \\
\qquad\qquad\quad\ \texttt{CFG}\ elementTest\ stateInterp\ context\ (x :: ins)\ s\ outs) \wedge \\
\qquad\qquad\quad (a_3 = \\
\qquad\qquad\quad\ \texttt{CFG}\ elementTest\ stateInterp\ context\ ins\ (NS\ s\ \texttt{(trap (inputList}\ x\texttt{))})\ (Out\ s\ \texttt{(trap (inputList}\ x\texttt{))} :: outs)) \wedge \\
\qquad\qquad\quad \texttt{authenticationTest}\ elementTest\ x \wedge \\
\qquad\qquad\quad \texttt{CFGInterpret}\ (M, Oi, Os)\ (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x :: ins)\ s
\end{array}$$

```
            outs)) ∨
        (∃ elementTest NS M Oi Os Out s context stateInterp x
           ins outs.
           (a0 = (M,Oi,Os)) ∧ (a1 = discard (inputList x)) ∧
           (a2 =
            CFG elementTest stateInterp context (x::ins) s
              outs) ∧
           (a3 =
            CFG elementTest stateInterp context ins
              (NS s (discard (inputList x)))
              (Out s (discard (inputList x))::outs)) ∧
           ¬authenticationTest elementTest x) ⇒
        TR′ a0 a1 a2 a3) ⇒
     TR′ a0 a1 a2 a3)
```

3.3 Theorems

[CFGInterpret_def]

```
⊢ CFGInterpret (M,Oi,Os)
     (CFG elementTest stateInterp context (x::ins) state
        outStream) ⟺
   (M,Oi,Os) satList context x ∧ (M,Oi,Os) satList x ∧
   (M,Oi,Os) satList stateInterp state x
```

[CFGInterpret_ind]

```
⊢ ∀ P.
     (∀ M Oi Os elementTest stateInterp context x ins state
         outStream.
        P (M,Oi,Os)
          (CFG elementTest stateInterp context (x::ins) state
             outStream)) ∧
     (∀ v15 v10 v11 v12 v13 v14.
        P v15 (CFG v10 v11 v12 [] v13 v14)) ⇒
     ∀ v v1 v2 v3. P (v,v1,v2) v3
```

[configuration_one_one]

```
⊢ ∀ a0 a1 a2 a3 a4 a5 a0′ a1′ a2′ a3′ a4′ a5′.
     (CFG a0 a1 a2 a3 a4 a5 = CFG a0′ a1′ a2′ a3′ a4′ a5′) ⟺
     (a0 = a0′) ∧ (a1 = a1′) ∧ (a2 = a2′) ∧ (a3 = a3′) ∧
     (a4 = a4′) ∧ (a5 = a5′)
```

[extractCommand_def]

```
⊢ extractCommand (P says prop (SOME cmd)) = cmd
```

[extractCommand_ind]

```
⊢ ∀ P′.
     (∀ P cmd. P′ (P says prop (SOME cmd))) ∧ P′ TT ∧ P′ FF ∧
     (∀ v1. P′ (prop v1)) ∧ (∀ v3. P′ (notf v3)) ∧
```

$(\forall\, v_6\ v_7.\ P'\ (v_6\ \texttt{andf}\ v_7)) \wedge (\forall\, v_{10}\ v_{11}.\ P'\ (v_{10}\ \texttt{orf}\ v_{11})) \wedge$
$(\forall\, v_{14}\ v_{15}.\ P'\ (v_{14}\ \texttt{impf}\ v_{15})) \wedge$
$(\forall\, v_{18}\ v_{19}.\ P'\ (v_{18}\ \texttt{eqf}\ v_{19})) \wedge (\forall\, v129.\ P'\ (v129\ \texttt{says TT})) \wedge$
$(\forall\, v130.\ P'\ (v130\ \texttt{says FF})) \wedge$
$(\forall\, v132.\ P'\ (v132\ \texttt{says prop NONE})) \wedge$
$(\forall\, v133\ v_{66}.\ P'\ (v133\ \texttt{says notf}\ v_{66})) \wedge$
$(\forall\, v134\ v_{69}\ v_{70}.\ P'\ (v134\ \texttt{says}\ (v_{69}\ \texttt{andf}\ v_{70}))) \wedge$
$(\forall\, v135\ v_{73}\ v_{74}.\ P'\ (v135\ \texttt{says}\ (v_{73}\ \texttt{orf}\ v_{74}))) \wedge$
$(\forall\, v136\ v_{77}\ v_{78}.\ P'\ (v136\ \texttt{says}\ (v_{77}\ \texttt{impf}\ v_{78}))) \wedge$
$(\forall\, v137\ v_{81}\ v_{82}.\ P'\ (v137\ \texttt{says}\ (v_{81}\ \texttt{eqf}\ v_{82}))) \wedge$
$(\forall\, v138\ v_{85}\ v_{86}.\ P'\ (v138\ \texttt{says}\ v_{85}\ \texttt{says}\ v_{86})) \wedge$
$(\forall\, v139\ v_{89}\ v_{90}.\ P'\ (v139\ \texttt{says}\ v_{89}\ \texttt{speaks\_for}\ v_{90})) \wedge$
$(\forall\, v140\ v_{93}\ v_{94}.\ P'\ (v140\ \texttt{says}\ v_{93}\ \texttt{controls}\ v_{94})) \wedge$
$(\forall\, v141\ v_{98}\ v_{99}\ v100.\ P'\ (v141\ \texttt{says reps}\ v_{98}\ v_{99}\ v100)) \wedge$
$(\forall\, v142\ v103\ v104.\ P'\ (v142\ \texttt{says}\ v103\ \texttt{domi}\ v104)) \wedge$
$(\forall\, v143\ v107\ v108.\ P'\ (v143\ \texttt{says}\ v107\ \texttt{eqi}\ v108)) \wedge$
$(\forall\, v144\ v111\ v112.\ P'\ (v144\ \texttt{says}\ v111\ \texttt{doms}\ v112)) \wedge$
$(\forall\, v145\ v115\ v116.\ P'\ (v145\ \texttt{says}\ v115\ \texttt{eqs}\ v116)) \wedge$
$(\forall\, v146\ v119\ v120.\ P'\ (v146\ \texttt{says}\ v119\ \texttt{eqn}\ v120)) \wedge$
$(\forall\, v147\ v123\ v124.\ P'\ (v147\ \texttt{says}\ v123\ \texttt{lte}\ v124)) \wedge$
$(\forall\, v148\ v127\ v128.\ P'\ (v148\ \texttt{says}\ v127\ \texttt{lt}\ v128)) \wedge$
$(\forall\, v_{24}\ v_{25}.\ P'\ (v_{24}\ \texttt{speaks\_for}\ v_{25})) \wedge$
$(\forall\, v_{28}\ v_{29}.\ P'\ (v_{28}\ \texttt{controls}\ v_{29})) \wedge$
$(\forall\, v_{33}\ v_{34}\ v_{35}.\ P'\ (\texttt{reps}\ v_{33}\ v_{34}\ v_{35})) \wedge$
$(\forall\, v_{38}\ v_{39}.\ P'\ (v_{38}\ \texttt{domi}\ v_{39})) \wedge$
$(\forall\, v_{42}\ v_{43}.\ P'\ (v_{42}\ \texttt{eqi}\ v_{43})) \wedge$
$(\forall\, v_{46}\ v_{47}.\ P'\ (v_{46}\ \texttt{doms}\ v_{47})) \wedge$
$(\forall\, v_{50}\ v_{51}.\ P'\ (v_{50}\ \texttt{eqs}\ v_{51})) \wedge$
$(\forall\, v_{54}\ v_{55}.\ P'\ (v_{54}\ \texttt{eqn}\ v_{55})) \wedge$
$(\forall\, v_{58}\ v_{59}.\ P'\ (v_{58}\ \texttt{lte}\ v_{59})) \wedge$
$(\forall\, v_{62}\ v_{63}.\ P'\ (v_{62}\ \texttt{lt}\ v_{63})) \Rightarrow$
$\forall\, v.\ P'\ v$

[extractInput_def]

$\vdash$ extractInput (P says prop x) = x

[extractInput_ind]

$\vdash \forall\, P'.$
$(\forall\, P\ x.\ P'\ (P\ \texttt{says prop}\ x)) \wedge P'\ \texttt{TT} \wedge P'\ \texttt{FF} \wedge$
$(\forall\, v_1.\ P'\ (\texttt{prop}\ v_1)) \wedge (\forall\, v_3.\ P'\ (\texttt{notf}\ v_3)) \wedge$
$(\forall\, v_6\ v_7.\ P'\ (v_6\ \texttt{andf}\ v_7)) \wedge (\forall\, v_{10}\ v_{11}.\ P'\ (v_{10}\ \texttt{orf}\ v_{11})) \wedge$
$(\forall\, v_{14}\ v_{15}.\ P'\ (v_{14}\ \texttt{impf}\ v_{15})) \wedge$
$(\forall\, v_{18}\ v_{19}.\ P'\ (v_{18}\ \texttt{eqf}\ v_{19})) \wedge (\forall\, v129.\ P'\ (v129\ \texttt{says TT})) \wedge$
$(\forall\, v130.\ P'\ (v130\ \texttt{says FF})) \wedge$
$(\forall\, v131\ v_{66}.\ P'\ (v131\ \texttt{says notf}\ v_{66})) \wedge$
$(\forall\, v132\ v_{69}\ v_{70}.\ P'\ (v132\ \texttt{says}\ (v_{69}\ \texttt{andf}\ v_{70}))) \wedge$
$(\forall\, v133\ v_{73}\ v_{74}.\ P'\ (v133\ \texttt{says}\ (v_{73}\ \texttt{orf}\ v_{74}))) \wedge$
$(\forall\, v134\ v_{77}\ v_{78}.\ P'\ (v134\ \texttt{says}\ (v_{77}\ \texttt{impf}\ v_{78}))) \wedge$
$(\forall\, v135\ v_{81}\ v_{82}.\ P'\ (v135\ \texttt{says}\ (v_{81}\ \texttt{eqf}\ v_{82}))) \wedge$

$(\forall\, v136\ v_{85}\ v_{86}.\ P'\ (v136\ \texttt{says}\ v_{85}\ \texttt{says}\ v_{86})) \wedge$
$(\forall\, v137\ v_{89}\ v_{90}.\ P'\ (v137\ \texttt{says}\ v_{89}\ \texttt{speaks\_for}\ v_{90})) \wedge$
$(\forall\, v138\ v_{93}\ v_{94}.\ P'\ (v138\ \texttt{says}\ v_{93}\ \texttt{controls}\ v_{94})) \wedge$
$(\forall\, v139\ v_{98}\ v_{99}\ v100.\ P'\ (v139\ \texttt{says reps}\ v_{98}\ v_{99}\ v100)) \wedge$
$(\forall\, v140\ v103\ v104.\ P'\ (v140\ \texttt{says}\ v103\ \texttt{domi}\ v104)) \wedge$
$(\forall\, v141\ v107\ v108.\ P'\ (v141\ \texttt{says}\ v107\ \texttt{eqi}\ v108)) \wedge$
$(\forall\, v142\ v111\ v112.\ P'\ (v142\ \texttt{says}\ v111\ \texttt{doms}\ v112)) \wedge$
$(\forall\, v143\ v115\ v116.\ P'\ (v143\ \texttt{says}\ v115\ \texttt{eqs}\ v116)) \wedge$
$(\forall\, v144\ v119\ v120.\ P'\ (v144\ \texttt{says}\ v119\ \texttt{eqn}\ v120)) \wedge$
$(\forall\, v145\ v123\ v124.\ P'\ (v145\ \texttt{says}\ v123\ \texttt{lte}\ v124)) \wedge$
$(\forall\, v146\ v127\ v128.\ P'\ (v146\ \texttt{says}\ v127\ \texttt{lt}\ v128)) \wedge$
$(\forall\, v_{24}\ v_{25}.\ P'\ (v_{24}\ \texttt{speaks\_for}\ v_{25})) \wedge$
$(\forall\, v_{28}\ v_{29}.\ P'\ (v_{28}\ \texttt{controls}\ v_{29})) \wedge$
$(\forall\, v_{33}\ v_{34}\ v_{35}.\ P'\ (\texttt{reps}\ v_{33}\ v_{34}\ v_{35})) \wedge$
$(\forall\, v_{38}\ v_{39}.\ P'\ (v_{38}\ \texttt{domi}\ v_{39})) \wedge$
$(\forall\, v_{42}\ v_{43}.\ P'\ (v_{42}\ \texttt{eqi}\ v_{43})) \wedge$
$(\forall\, v_{46}\ v_{47}.\ P'\ (v_{46}\ \texttt{doms}\ v_{47})) \wedge$
$(\forall\, v_{50}\ v_{51}.\ P'\ (v_{50}\ \texttt{eqs}\ v_{51})) \wedge$
$(\forall\, v_{54}\ v_{55}.\ P'\ (v_{54}\ \texttt{eqn}\ v_{55})) \wedge$
$(\forall\, v_{58}\ v_{59}.\ P'\ (v_{58}\ \texttt{lte}\ v_{59})) \wedge$
$(\forall\, v_{62}\ v_{63}.\ P'\ (v_{62}\ \texttt{lt}\ v_{63})) \Rightarrow$
$\forall\, v.\ P'\ v$

[extractPropCommand_def]

$\vdash$ extractPropCommand (P says prop (SOME cmd)) = prop (SOME cmd)

[extractPropCommand_ind]

$\vdash \forall\, P'.$
$(\forall\, P\ cmd.\ P'\ (P\ \texttt{says prop (SOME}\ cmd))) \wedge P'\ \texttt{TT} \wedge P'\ \texttt{FF} \wedge$
$(\forall\, v_1.\ P'\ (\texttt{prop}\ v_1)) \wedge (\forall\, v_3.\ P'\ (\texttt{notf}\ v_3)) \wedge$
$(\forall\, v_6\ v_7.\ P'\ (v_6\ \texttt{andf}\ v_7)) \wedge (\forall\, v_{10}\ v_{11}.\ P'\ (v_{10}\ \texttt{orf}\ v_{11})) \wedge$
$(\forall\, v_{14}\ v_{15}.\ P'\ (v_{14}\ \texttt{impf}\ v_{15})) \wedge$
$(\forall\, v_{18}\ v_{19}.\ P'\ (v_{18}\ \texttt{eqf}\ v_{19})) \wedge (\forall\, v129.\ P'\ (v129\ \texttt{says TT})) \wedge$
$(\forall\, v130.\ P'\ (v130\ \texttt{says FF})) \wedge$
$(\forall\, v132.\ P'\ (v132\ \texttt{says prop NONE})) \wedge$
$(\forall\, v133\ v_{66}.\ P'\ (v133\ \texttt{says notf}\ v_{66})) \wedge$
$(\forall\, v134\ v_{69}\ v_{70}.\ P'\ (v134\ \texttt{says}\ (v_{69}\ \texttt{andf}\ v_{70}))) \wedge$
$(\forall\, v135\ v_{73}\ v_{74}.\ P'\ (v135\ \texttt{says}\ (v_{73}\ \texttt{orf}\ v_{74}))) \wedge$
$(\forall\, v136\ v_{77}\ v_{78}.\ P'\ (v136\ \texttt{says}\ (v_{77}\ \texttt{impf}\ v_{78}))) \wedge$
$(\forall\, v137\ v_{81}\ v_{82}.\ P'\ (v137\ \texttt{says}\ (v_{81}\ \texttt{eqf}\ v_{82}))) \wedge$
$(\forall\, v138\ v_{85}\ v_{86}.\ P'\ (v138\ \texttt{says}\ v_{85}\ \texttt{says}\ v_{86})) \wedge$
$(\forall\, v139\ v_{89}\ v_{90}.\ P'\ (v139\ \texttt{says}\ v_{89}\ \texttt{speaks\_for}\ v_{90})) \wedge$
$(\forall\, v140\ v_{93}\ v_{94}.\ P'\ (v140\ \texttt{says}\ v_{93}\ \texttt{controls}\ v_{94})) \wedge$
$(\forall\, v141\ v_{98}\ v_{99}\ v100.\ P'\ (v141\ \texttt{says reps}\ v_{98}\ v_{99}\ v100)) \wedge$
$(\forall\, v142\ v103\ v104.\ P'\ (v142\ \texttt{says}\ v103\ \texttt{domi}\ v104)) \wedge$
$(\forall\, v143\ v107\ v108.\ P'\ (v143\ \texttt{says}\ v107\ \texttt{eqi}\ v108)) \wedge$
$(\forall\, v144\ v111\ v112.\ P'\ (v144\ \texttt{says}\ v111\ \texttt{doms}\ v112)) \wedge$
$(\forall\, v145\ v115\ v116.\ P'\ (v145\ \texttt{says}\ v115\ \texttt{eqs}\ v116)) \wedge$
$(\forall\, v146\ v119\ v120.\ P'\ (v146\ \texttt{says}\ v119\ \texttt{eqn}\ v120)) \wedge$

$(\forall\, v147\ v123\ v124.\ P'\ (v147\ \texttt{says}\ v123\ \texttt{lte}\ v124)) \wedge$
$(\forall\, v148\ v127\ v128.\ P'\ (v148\ \texttt{says}\ v127\ \texttt{lt}\ v128)) \wedge$
$(\forall\, v_{24}\ v_{25}.\ P'\ (v_{24}\ \texttt{speaks\_for}\ v_{25})) \wedge$
$(\forall\, v_{28}\ v_{29}.\ P'\ (v_{28}\ \texttt{controls}\ v_{29})) \wedge$
$(\forall\, v_{33}\ v_{34}\ v_{35}.\ P'\ (\texttt{reps}\ v_{33}\ v_{34}\ v_{35})) \wedge$
$(\forall\, v_{38}\ v_{39}.\ P'\ (v_{38}\ \texttt{domi}\ v_{39})) \wedge$
$(\forall\, v_{42}\ v_{43}.\ P'\ (v_{42}\ \texttt{eqi}\ v_{43})) \wedge$
$(\forall\, v_{46}\ v_{47}.\ P'\ (v_{46}\ \texttt{doms}\ v_{47})) \wedge$
$(\forall\, v_{50}\ v_{51}.\ P'\ (v_{50}\ \texttt{eqs}\ v_{51})) \wedge$
$(\forall\, v_{54}\ v_{55}.\ P'\ (v_{54}\ \texttt{eqn}\ v_{55})) \wedge$
$(\forall\, v_{58}\ v_{59}.\ P'\ (v_{58}\ \texttt{lte}\ v_{59})) \wedge$
$(\forall\, v_{62}\ v_{63}.\ P'\ (v_{62}\ \texttt{lt}\ v_{63})) \Rightarrow$
$\forall\, v.\ P'\ v$

[TR_cases]

$\vdash \forall\, a_0\ a_1\ a_2\ a_3.$
$\texttt{TR}\ a_0\ a_1\ a_2\ a_3 \iff$
$(\exists\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins\ outs.$
$(a_0\ \texttt{=}\ (M, Oi, Os)) \wedge (a_1\ \texttt{=}\ \texttt{exec}\ (\texttt{inputList}\ x)) \wedge$
$(a_2\ \texttt{=}$
$\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs) \wedge$
$(a_3\ \texttt{=}$
$\texttt{CFG}\ elementTest\ stateInterp\ context\ ins$
$(NS\ s\ (\texttt{exec}\ (\texttt{inputList}\ x)))$
$(Out\ s\ (\texttt{exec}\ (\texttt{inputList}\ x))::outs)) \wedge$
$\texttt{authenticationTest}\ elementTest\ x \wedge$
$\texttt{CFGInterpret}\ (M, Oi, Os)$
$(\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs)) \vee$
$(\exists\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins\ outs.$
$(a_0\ \texttt{=}\ (M, Oi, Os)) \wedge (a_1\ \texttt{=}\ \texttt{trap}\ (\texttt{inputList}\ x)) \wedge$
$(a_2\ \texttt{=}$
$\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs) \wedge$
$(a_3\ \texttt{=}$
$\texttt{CFG}\ elementTest\ stateInterp\ context\ ins$
$(NS\ s\ (\texttt{trap}\ (\texttt{inputList}\ x)))$
$(Out\ s\ (\texttt{trap}\ (\texttt{inputList}\ x))::outs)) \wedge$
$\texttt{authenticationTest}\ elementTest\ x \wedge$
$\texttt{CFGInterpret}\ (M, Oi, Os)$
$(\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs)) \vee$
$\exists\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins\ outs.$
$(a_0\ \texttt{=}\ (M, Oi, Os)) \wedge (a_1\ \texttt{=}\ \texttt{discard}\ (\texttt{inputList}\ x)) \wedge$
$(a_2\ \texttt{=}$
$\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs) \wedge$
$(a_3\ \texttt{=}$

```
        CFG elementTest stateInterp context ins
          (NS s (discard (inputList x)))
          (Out s (discard (inputList x))::outs)) ∧
       ¬authenticationTest elementTest x
```

[TR_discard_cmd_rule]

```
⊢ TR (M,Oi,Os) (discard (inputList x))
    (CFG elementTest stateInterp context (x::ins) s outs)
    (CFG elementTest stateInterp context ins
       (NS s (discard (inputList x)))
       (Out s (discard (inputList x))::outs)) ⟺
  ¬authenticationTest elementTest x
```

[TR_EQ_rules_thm]

```
⊢ (TR (M,Oi,Os) (exec (inputList x))
     (CFG elementTest stateInterp context (x::ins) s outs)
     (CFG elementTest stateInterp context ins
        (NS s (exec (inputList x)))
        (Out s (exec (inputList x))::outs)) ⟺
   authenticationTest elementTest x ∧
   CFGInterpret (M,Oi,Os)
     (CFG elementTest stateInterp context (x::ins) s outs)) ∧
  (TR (M,Oi,Os) (trap (inputList x))
     (CFG elementTest stateInterp context (x::ins) s outs)
     (CFG elementTest stateInterp context ins
        (NS s (trap (inputList x)))
        (Out s (trap (inputList x))::outs)) ⟺
   authenticationTest elementTest x ∧
   CFGInterpret (M,Oi,Os)
     (CFG elementTest stateInterp context (x::ins) s outs)) ∧
  (TR (M,Oi,Os) (discard (inputList x))
     (CFG elementTest stateInterp context (x::ins) s outs)
     (CFG elementTest stateInterp context ins
        (NS s (discard (inputList x)))
        (Out s (discard (inputList x))::outs)) ⟺
   ¬authenticationTest elementTest x)
```

[TR_exec_cmd_rule]

```
⊢ ∀elementTest context stateInterp x ins s outs.
    (∀M Oi Os.
       CFGInterpret (M,Oi,Os)
         (CFG elementTest stateInterp context (x::ins) s
            outs) ⇒
       (M,Oi,Os) satList propCommandList x) ⇒
    ∀NS Out M Oi Os.
      TR (M,Oi,Os) (exec (inputList x))
        (CFG elementTest stateInterp context (x::ins) s outs)
        (CFG elementTest stateInterp context ins
```

$$
\begin{array}{l}
\quad\quad\quad\quad (NS\ s\ (\texttt{exec}\ (\texttt{inputList}\ x))) \\
\quad\quad\quad\quad (Out\ s\ (\texttt{exec}\ (\texttt{inputList}\ x))::outs)) \iff \\
\quad\quad \texttt{authenticationTest}\ elementTest\ x\ \wedge \\
\quad\quad \texttt{CFGInterpret}\ (M, Oi, Os) \\
\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs)\ \wedge \\
\quad\quad (M, Oi, Os)\ \texttt{satList propCommandList}\ x
\end{array}
$$

[TR_ind]

$$
\begin{array}{l}
\vdash\ \forall\, TR'. \\
\quad\quad (\forall\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins \\
\quad\quad\quad outs. \\
\quad\quad\quad \texttt{authenticationTest}\ elementTest\ x\ \wedge \\
\quad\quad\quad \texttt{CFGInterpret}\ (M, Oi, Os) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s \\
\quad\quad\quad\quad\quad outs)\ \Rightarrow \\
\quad\quad\quad TR'\ (M, Oi, Os)\ (\texttt{exec}\ (\texttt{inputList}\ x)) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ ins \\
\quad\quad\quad\quad\quad (NS\ s\ (\texttt{exec}\ (\texttt{inputList}\ x))) \\
\quad\quad\quad\quad\quad (Out\ s\ (\texttt{exec}\ (\texttt{inputList}\ x))::outs)))\ \wedge \\
\quad\quad (\forall\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins \\
\quad\quad\quad outs. \\
\quad\quad\quad \texttt{authenticationTest}\ elementTest\ x\ \wedge \\
\quad\quad\quad \texttt{CFGInterpret}\ (M, Oi, Os) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s \\
\quad\quad\quad\quad\quad outs)\ \Rightarrow \\
\quad\quad\quad TR'\ (M, Oi, Os)\ (\texttt{trap}\ (\texttt{inputList}\ x)) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ ins \\
\quad\quad\quad\quad\quad (NS\ s\ (\texttt{trap}\ (\texttt{inputList}\ x))) \\
\quad\quad\quad\quad\quad (Out\ s\ (\texttt{trap}\ (\texttt{inputList}\ x))::outs)))\ \wedge \\
\quad\quad (\forall\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins \\
\quad\quad\quad outs. \\
\quad\quad\quad \neg\texttt{authenticationTest}\ elementTest\ x\ \Rightarrow \\
\quad\quad\quad TR'\ (M, Oi, Os)\ (\texttt{discard}\ (\texttt{inputList}\ x)) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ ins \\
\quad\quad\quad\quad\quad (NS\ s\ (\texttt{discard}\ (\texttt{inputList}\ x))) \\
\quad\quad\quad\quad\quad (Out\ s\ (\texttt{discard}\ (\texttt{inputList}\ x))::outs)))\ \Rightarrow \\
\quad\quad \forall\, a_0\ a_1\ a_2\ a_3.\ \texttt{TR}\ a_0\ a_1\ a_2\ a_3\ \Rightarrow\ TR'\ a_0\ a_1\ a_2\ a_3
\end{array}
$$

[TR_rules]

$$
\begin{array}{l}
\vdash\ (\forall\, elementTest\ NS\ M\ Oi\ Os\ Out\ s\ context\ stateInterp\ x\ ins \\
\quad\quad\quad outs. \\
\quad\quad\quad \texttt{authenticationTest}\ elementTest\ x\ \wedge \\
\quad\quad\quad \texttt{CFGInterpret}\ (M, Oi, Os) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs)\ \Rightarrow \\
\quad\quad\quad \texttt{TR}\ (M, Oi, Os)\ (\texttt{exec}\ (\texttt{inputList}\ x)) \\
\quad\quad\quad\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ s\ outs)
\end{array}
$$

```
      (CFG elementTest stateInterp context ins
         (NS s (exec (inputList x)))
         (Out s (exec (inputList x))::outs))) ∧
  (∀ elementTest NS M Oi Os Out s context stateInterp x ins
     outs.
    authenticationTest elementTest x ∧
    CFGInterpret (M,Oi,Os)
      (CFG elementTest stateInterp context (x::ins) s outs) ⇒
    TR (M,Oi,Os) (trap (inputList x))
      (CFG elementTest stateInterp context (x::ins) s outs)
      (CFG elementTest stateInterp context ins
         (NS s (trap (inputList x)))
         (Out s (trap (inputList x))::outs))) ∧
  ∀ elementTest NS M Oi Os Out s context stateInterp x ins outs.
    ¬authenticationTest elementTest x ⇒
    TR (M,Oi,Os) (discard (inputList x))
      (CFG elementTest stateInterp context (x::ins) s outs)
      (CFG elementTest stateInterp context ins
         (NS s (discard (inputList x)))
         (Out s (discard (inputList x))::outs))
```

[TR_strongind]

```
⊢ ∀ TR'.
    (∀ elementTest NS M Oi Os Out s context stateInterp x ins
       outs.
      authenticationTest elementTest x ∧
      CFGInterpret (M,Oi,Os)
        (CFG elementTest stateInterp context (x::ins) s
           outs) ⇒
      TR' (M,Oi,Os) (exec (inputList x))
        (CFG elementTest stateInterp context (x::ins) s outs)
        (CFG elementTest stateInterp context ins
           (NS s (exec (inputList x)))
           (Out s (exec (inputList x))::outs))) ∧
    (∀ elementTest NS M Oi Os Out s context stateInterp x ins
       outs.
      authenticationTest elementTest x ∧
      CFGInterpret (M,Oi,Os)
        (CFG elementTest stateInterp context (x::ins) s
           outs) ⇒
      TR' (M,Oi,Os) (trap (inputList x))
        (CFG elementTest stateInterp context (x::ins) s outs)
        (CFG elementTest stateInterp context ins
           (NS s (trap (inputList x)))
           (Out s (trap (inputList x))::outs))) ∧
    (∀ elementTest NS M Oi Os Out s context stateInterp x ins
       outs.
      ¬authenticationTest elementTest x ⇒
      TR' (M,Oi,Os) (discard (inputList x))
```

(CFG *elementTest stateInterp context* $(x::ins)$ s *outs*)
(CFG *elementTest stateInterp context ins*
(*NS* s (discard (inputList x)))
(*Out* s (discard (inputList x))::*outs*))) $\Rightarrow$
$\forall a_0\ a_1\ a_2\ a_3$. TR $a_0\ a_1\ a_2\ a_3 \Rightarrow TR'\ a_0\ a_1\ a_2\ a_3$

[TR_trap_cmd_rule]

$\vdash$ $\forall$ *elementTest context stateInterp x ins s outs*.
($\forall$ *M Oi Os*.
CFGInterpret (M,Oi,Os)
(CFG *elementTest stateInterp context* $(x::ins)$ s
outs) $\Rightarrow$
(M,Oi,Os) sat prop NONE) $\Rightarrow$
$\forall$ *NS Out M Oi Os*.
TR (M,Oi,Os) (trap (inputList x))
(CFG *elementTest stateInterp context* $(x::ins)$ s *outs*)
(CFG *elementTest stateInterp context ins*
(*NS* s (trap (inputList x)))
(*Out* s (trap (inputList x))::*outs*)) $\iff$
authenticationTest *elementTest x* $\wedge$
CFGInterpret (M,Oi,Os)
(CFG *elementTest stateInterp context* $(x::ins)$ s *outs*) $\wedge$
(M,Oi,Os) sat prop NONE

[TRrule0]

$\vdash$ TR (M,Oi,Os) (exec (inputList x))
(CFG *elementTest stateInterp context* $(x::ins)$ s *outs*)
(CFG *elementTest stateInterp context ins*
(*NS* s (exec (inputList x)))
(*Out* s (exec (inputList x))::*outs*)) $\iff$
authenticationTest *elementTest x* $\wedge$
CFGInterpret (M,Oi,Os)
(CFG *elementTest stateInterp context* $(x::ins)$ s *outs*)

[TRrule1]

$\vdash$ TR (M,Oi,Os) (trap (inputList x))
(CFG *elementTest stateInterp context* $(x::ins)$ s *outs*)
(CFG *elementTest stateInterp context ins*
(*NS* s (trap (inputList x)))
(*Out* s (trap (inputList x))::*outs*)) $\iff$
authenticationTest *elementTest x* $\wedge$
CFGInterpret (M,Oi,Os)
(CFG *elementTest stateInterp context* $(x::ins)$ s *outs*)

[trType_distinct_clauses]

$\vdash$ ($\forall a'\ a$. discard $a \neq$ trap a') $\wedge$ ($\forall a'\ a$. discard $a \neq$ exec a') $\wedge$
$\forall a'\ a$. trap $a \neq$ exec a'

[trType_one_one]
⊢ (∀ a a'. (discard a = discard a') ⟺ (a = a')) ∧
(∀ a a'. (trap a = trap a') ⟺ (a = a')) ∧
∀ a a'. (exec a = exec a') ⟺ (a = a')

4 satList Theory

Built: 13 May 2018
Parent Theories: aclDrules

4.1 Definitions

[satList_def]
⊢ ∀ M Oi Os $formList$.
(M,Oi,Os) satList $formList$ ⟺
FOLDR (λ x y. x ∧ y) T (MAP (λ f. (M,Oi,Os) sat f) $formList$)

4.2 Theorems

[satList_conj]
⊢ ∀ l_1 l_2 M Oi Os.
(M,Oi,Os) satList l_1 ∧ (M,Oi,Os) satList l_2 ⟺
(M,Oi,Os) satList (l_1 ++ l_2)

[satList_CONS]
⊢ ∀ h t M Oi Os.
(M,Oi,Os) satList (h::t) ⟺
(M,Oi,Os) sat h ∧ (M,Oi,Os) satList t

[satList_nil]
⊢ (M,Oi,Os) satList []

5 ssmPB Theory

Built: 13 May 2018
Parent Theories: PBType, ssm11, OMNIType

5.1 Definitions

[secContext_def]
⊢ ∀ cmd.
secContext cmd =
[Name PlatoonLeader controls prop (SOME (SLc cmd))]

[ssmPBStateInterp_def]
⊢ ∀ $state$. ssmPBStateInterp $state$ = TT

5.2 Theorems

[authenticationTest_cmd_reject_lemma]

$\vdash \forall\, cmd.\ \neg$authenticationTest (prop (SOME cmd))

[authenticationTest_def]

$\vdash$ (authenticationTest (Name PlatoonLeader says prop cmd) $\iff$
T) $\wedge$ (authenticationTest TT $\iff$ F) $\wedge$
(authenticationTest FF $\iff$ F) $\wedge$
(authenticationTest (prop v) $\iff$ F) $\wedge$
(authenticationTest (notf v_1) $\iff$ F) $\wedge$
(authenticationTest (v_2 andf v_3) $\iff$ F) $\wedge$
(authenticationTest (v_4 orf v_5) $\iff$ F) $\wedge$
(authenticationTest (v_6 impf v_7) $\iff$ F) $\wedge$
(authenticationTest (v_8 eqf v_9) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says TT) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says FF) $\iff$ F) $\wedge$
(authenticationTest (*v133* meet *v134* says prop v_{66}) $\iff$ F) $\wedge$
(authenticationTest (*v135* quoting *v136* says prop v_{66}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says notf v_{67}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says (v_{68} andf v_{69})) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says (v_{70} orf v_{71})) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says (v_{72} impf v_{73})) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says (v_{74} eqf v_{75})) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{76} says v_{77}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{78} speaks_for v_{79}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{80} controls v_{81}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says reps v_{82} v_{83} v_{84}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{85} domi v_{86}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{87} eqi v_{88}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{89} doms v_{90}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{91} eqs v_{92}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{93} eqn v_{94}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{95} lte v_{96}) $\iff$ F) $\wedge$
(authenticationTest (v_{10} says v_{97} lt v_{98}) $\iff$ F) $\wedge$
(authenticationTest (v_{12} speaks_for v_{13}) $\iff$ F) $\wedge$
(authenticationTest (v_{14} controls v_{15}) $\iff$ F) $\wedge$
(authenticationTest (reps v_{16} v_{17} v_{18}) $\iff$ F) $\wedge$
(authenticationTest (v_{19} domi v_{20}) $\iff$ F) $\wedge$
(authenticationTest (v_{21} eqi v_{22}) $\iff$ F) $\wedge$
(authenticationTest (v_{23} doms v_{24}) $\iff$ F) $\wedge$
(authenticationTest (v_{25} eqs v_{26}) $\iff$ F) $\wedge$
(authenticationTest (v_{27} eqn v_{28}) $\iff$ F) $\wedge$
(authenticationTest (v_{29} lte v_{30}) $\iff$ F) $\wedge$
(authenticationTest (v_{31} lt v_{32}) $\iff$ F)

[authenticationTest_ind]

$\vdash \forall\, P.$
(∀ cmd. P (Name PlatoonLeader says prop cmd)) $\wedge$ P TT $\wedge$

P FF $\wedge$ $(\forall v.\ P$ (prop v)) $\wedge$ $(\forall v_1.\ P$ (notf v_1)) $\wedge$
$(\forall v_2\ v_3.\ P$ (v_2 andf v_3)) $\wedge$ $(\forall v_4\ v_5.\ P$ (v_4 orf v_5)) $\wedge$
$(\forall v_6\ v_7.\ P$ (v_6 impf v_7)) $\wedge$ $(\forall v_8\ v_9.\ P$ (v_8 eqf v_9)) $\wedge$
$(\forall v_{10}.\ P$ (v_{10} says TT)) $\wedge$ $(\forall v_{10}.\ P$ (v_{10} says FF)) $\wedge$
$(\forall v133\ v134\ v_{66}.\ P$ ($v133$ meet $v134$ says prop v_{66})) $\wedge$
$(\forall v135\ v136\ v_{66}.\ P$ ($v135$ quoting $v136$ says prop v_{66})) $\wedge$
$(\forall v_{10}\ v_{67}.\ P$ (v_{10} says notf v_{67})) $\wedge$
$(\forall v_{10}\ v_{68}\ v_{69}.\ P$ (v_{10} says (v_{68} andf v_{69}))) $\wedge$
$(\forall v_{10}\ v_{70}\ v_{71}.\ P$ (v_{10} says (v_{70} orf v_{71}))) $\wedge$
$(\forall v_{10}\ v_{72}\ v_{73}.\ P$ (v_{10} says (v_{72} impf v_{73}))) $\wedge$
$(\forall v_{10}\ v_{74}\ v_{75}.\ P$ (v_{10} says (v_{74} eqf v_{75}))) $\wedge$
$(\forall v_{10}\ v_{76}\ v_{77}.\ P$ (v_{10} says v_{76} says v_{77})) $\wedge$
$(\forall v_{10}\ v_{78}\ v_{79}.\ P$ (v_{10} says v_{78} speaks_for v_{79})) $\wedge$
$(\forall v_{10}\ v_{80}\ v_{81}.\ P$ (v_{10} says v_{80} controls v_{81})) $\wedge$
$(\forall v_{10}\ v_{82}\ v_{83}\ v_{84}.\ P$ (v_{10} says reps v_{82} v_{83} v_{84})) $\wedge$
$(\forall v_{10}\ v_{85}\ v_{86}.\ P$ (v_{10} says v_{85} domi v_{86})) $\wedge$
$(\forall v_{10}\ v_{87}\ v_{88}.\ P$ (v_{10} says v_{87} eqi v_{88})) $\wedge$
$(\forall v_{10}\ v_{89}\ v_{90}.\ P$ (v_{10} says v_{89} doms v_{90})) $\wedge$
$(\forall v_{10}\ v_{91}\ v_{92}.\ P$ (v_{10} says v_{91} eqs v_{92})) $\wedge$
$(\forall v_{10}\ v_{93}\ v_{94}.\ P$ (v_{10} says v_{93} eqn v_{94})) $\wedge$
$(\forall v_{10}\ v_{95}\ v_{96}.\ P$ (v_{10} says v_{95} lte v_{96})) $\wedge$
$(\forall v_{10}\ v_{97}\ v_{98}.\ P$ (v_{10} says v_{97} lt v_{98})) $\wedge$
$(\forall v_{12}\ v_{13}.\ P$ (v_{12} speaks_for v_{13})) $\wedge$
$(\forall v_{14}\ v_{15}.\ P$ (v_{14} controls v_{15})) $\wedge$
$(\forall v_{16}\ v_{17}\ v_{18}.\ P$ (reps v_{16} v_{17} v_{18})) $\wedge$
$(\forall v_{19}\ v_{20}.\ P$ (v_{19} domi v_{20})) $\wedge$
$(\forall v_{21}\ v_{22}.\ P$ (v_{21} eqi v_{22})) $\wedge$
$(\forall v_{23}\ v_{24}.\ P$ (v_{23} doms v_{24})) $\wedge$
$(\forall v_{25}\ v_{26}.\ P$ (v_{25} eqs v_{26})) $\wedge$ $(\forall v_{27}\ v_{28}.\ P$ (v_{27} eqn v_{28})) $\wedge$
$(\forall v_{29}\ v_{30}.\ P$ (v_{29} lte v_{30})) $\wedge$ $(\forall v_{31}\ v_{32}.\ P$ (v_{31} lt v_{32})) $\Rightarrow$
$\forall v.\ P\ v$

[PBNS_def]

```
⊢ (PBNS PLAN_PB (exec (SLc crossLD)) = MOVE_TO_ORP) ∧
  (PBNS PLAN_PB (exec (SLc incomplete)) = PLAN_PB) ∧
  (PBNS MOVE_TO_ORP (exec (SLc conductORP)) = CONDUCT_ORP) ∧
  (PBNS MOVE_TO_ORP (exec (SLc incomplete)) = MOVE_TO_ORP) ∧
  (PBNS CONDUCT_ORP (exec (SLc moveToPB)) = MOVE_TO_PB) ∧
  (PBNS CONDUCT_ORP (exec (SLc incomplete)) = CONDUCT_ORP) ∧
  (PBNS MOVE_TO_PB (exec (SLc conductPB)) = CONDUCT_PB) ∧
  (PBNS MOVE_TO_PB (exec (SLc incomplete)) = MOVE_TO_PB) ∧
  (PBNS CONDUCT_PB (exec (SLc completePB)) = COMPLETE_PB) ∧
  (PBNS CONDUCT_PB (exec (SLc incomplete)) = CONDUCT_PB) ∧
  (PBNS s (trap (SLc cmd)) = s) ∧
  (PBNS s (discard (SLc cmd)) = s)
```

[PBNS_ind]

```
⊢ ∀P.
    P PLAN_PB (exec (SLc crossLD)) ∧
```

P PLAN_PB (exec (SLc incomplete)) ∧
P MOVE_TO_ORP (exec (SLc conductORP)) ∧
P MOVE_TO_ORP (exec (SLc incomplete)) ∧
P CONDUCT_ORP (exec (SLc moveToPB)) ∧
P CONDUCT_ORP (exec (SLc incomplete)) ∧
P MOVE_TO_PB (exec (SLc conductPB)) ∧
P MOVE_TO_PB (exec (SLc incomplete)) ∧
P CONDUCT_PB (exec (SLc completePB)) ∧
P CONDUCT_PB (exec (SLc incomplete)) ∧
(∀ s cmd. P s (trap (SLc cmd))) ∧
(∀ s cmd. P s (discard (SLc cmd))) ∧
(∀ s v_6. P s (discard (ESCc v_6))) ∧
(∀ s v_9. P s (trap (ESCc v_9))) ∧
(∀ v_{12}. P PLAN_PB (exec (ESCc v_{12}))) ∧
P PLAN_PB (exec (SLc conductORP)) ∧
P PLAN_PB (exec (SLc moveToPB)) ∧
P PLAN_PB (exec (SLc conductPB)) ∧
P PLAN_PB (exec (SLc completePB)) ∧
(∀ v_{15}. P MOVE_TO_ORP (exec (ESCc v_{15}))) ∧
P MOVE_TO_ORP (exec (SLc crossLD)) ∧
P MOVE_TO_ORP (exec (SLc moveToPB)) ∧
P MOVE_TO_ORP (exec (SLc conductPB)) ∧
P MOVE_TO_ORP (exec (SLc completePB)) ∧
(∀ v_{18}. P CONDUCT_ORP (exec (ESCc v_{18}))) ∧
P CONDUCT_ORP (exec (SLc crossLD)) ∧
P CONDUCT_ORP (exec (SLc conductORP)) ∧
P CONDUCT_ORP (exec (SLc conductPB)) ∧
P CONDUCT_ORP (exec (SLc completePB)) ∧
(∀ v_{21}. P MOVE_TO_PB (exec (ESCc v_{21}))) ∧
P MOVE_TO_PB (exec (SLc crossLD)) ∧
P MOVE_TO_PB (exec (SLc conductORP)) ∧
P MOVE_TO_PB (exec (SLc moveToPB)) ∧
P MOVE_TO_PB (exec (SLc completePB)) ∧
(∀ v_{24}. P CONDUCT_PB (exec (ESCc v_{24}))) ∧
P CONDUCT_PB (exec (SLc crossLD)) ∧
P CONDUCT_PB (exec (SLc conductORP)) ∧
P CONDUCT_PB (exec (SLc moveToPB)) ∧
P CONDUCT_PB (exec (SLc conductPB)) ∧
(∀ v_{26}. P COMPLETE_PB (exec v_{26})) ⇒
∀ v v_1. P v v_1

[PBOut_def]

⊢ (PBOut PLAN_PB (exec (SLc crossLD)) = MoveToORP) ∧
(PBOut PLAN_PB (exec (SLc incomplete)) = PlanPB) ∧
(PBOut MOVE_TO_ORP (exec (SLc conductORP)) = ConductORP) ∧
(PBOut MOVE_TO_ORP (exec (SLc incomplete)) = MoveToORP) ∧
(PBOut CONDUCT_ORP (exec (SLc moveToPB)) = MoveToPB) ∧
(PBOut CONDUCT_ORP (exec (SLc incomplete)) = ConductORP) ∧
(PBOut MOVE_TO_PB (exec (SLc conductPB)) = ConductPB) ∧

(PBOut MOVE_TO_PB (exec (SLc incomplete)) = MoveToPB) ∧
(PBOut CONDUCT_PB (exec (SLc completePB)) = CompletePB) ∧
(PBOut CONDUCT_PB (exec (SLc incomplete)) = ConductPB) ∧
(PBOut s (trap (SLc cmd)) = unAuthorized) ∧
(PBOut s (discard (SLc cmd)) = unAuthenticated)

[PBOut_ind]

⊢ ∀ P.
P PLAN_PB (exec (SLc crossLD)) ∧
P PLAN_PB (exec (SLc incomplete)) ∧
P MOVE_TO_ORP (exec (SLc conductORP)) ∧
P MOVE_TO_ORP (exec (SLc incomplete)) ∧
P CONDUCT_ORP (exec (SLc moveToPB)) ∧
P CONDUCT_ORP (exec (SLc incomplete)) ∧
P MOVE_TO_PB (exec (SLc conductPB)) ∧
P MOVE_TO_PB (exec (SLc incomplete)) ∧
P CONDUCT_PB (exec (SLc completePB)) ∧
P CONDUCT_PB (exec (SLc incomplete)) ∧
(∀ s cmd. P s (trap (SLc cmd))) ∧
(∀ s cmd. P s (discard (SLc cmd))) ∧
(∀ s v_6. P s (discard (ESCc v_6))) ∧
(∀ s v_9. P s (trap (ESCc v_9))) ∧
(∀ v_{12}. P PLAN_PB (exec (ESCc v_{12}))) ∧
P PLAN_PB (exec (SLc conductORP)) ∧
P PLAN_PB (exec (SLc moveToPB)) ∧
P PLAN_PB (exec (SLc conductPB)) ∧
P PLAN_PB (exec (SLc completePB)) ∧
(∀ v_{15}. P MOVE_TO_ORP (exec (ESCc v_{15}))) ∧
P MOVE_TO_ORP (exec (SLc crossLD)) ∧
P MOVE_TO_ORP (exec (SLc moveToPB)) ∧
P MOVE_TO_ORP (exec (SLc conductPB)) ∧
P MOVE_TO_ORP (exec (SLc completePB)) ∧
(∀ v_{18}. P CONDUCT_ORP (exec (ESCc v_{18}))) ∧
P CONDUCT_ORP (exec (SLc crossLD)) ∧
P CONDUCT_ORP (exec (SLc conductORP)) ∧
P CONDUCT_ORP (exec (SLc conductPB)) ∧
P CONDUCT_ORP (exec (SLc completePB)) ∧
(∀ v_{21}. P MOVE_TO_PB (exec (ESCc v_{21}))) ∧
P MOVE_TO_PB (exec (SLc crossLD)) ∧
P MOVE_TO_PB (exec (SLc conductORP)) ∧
P MOVE_TO_PB (exec (SLc moveToPB)) ∧
P MOVE_TO_PB (exec (SLc completePB)) ∧
(∀ v_{24}. P CONDUCT_PB (exec (ESCc v_{24}))) ∧
P CONDUCT_PB (exec (SLc crossLD)) ∧
P CONDUCT_PB (exec (SLc conductORP)) ∧
P CONDUCT_PB (exec (SLc moveToPB)) ∧
P CONDUCT_PB (exec (SLc conductPB)) ∧
(∀ v_{26}. P COMPLETE_PB (exec v_{26})) ⇒
∀ v v_1. P v v_1

[PlatoonLeader_exec_slCommand_justified_thm]

```
⊢ ∀ NS Out M Oi Os.
    TR (M, Oi, Os) (exec (SLc slCommand))
      (CFG authenticationTest ssmPBStateInterp
         (secContext slCommand)
         (Name PlatoonLeader says prop (SOME (SLc slCommand))::
              ins) s outs)
      (CFG authenticationTest ssmPBStateInterp
         (secContext slCommand) ins
         (NS s (exec (SLc slCommand)))
         (Out s (exec (SLc slCommand))::outs)) ⟺
    authenticationTest
      (Name PlatoonLeader says prop (SOME (SLc slCommand))) ∧
    CFGInterpret (M, Oi, Os)
      (CFG authenticationTest ssmPBStateInterp
         (secContext slCommand)
         (Name PlatoonLeader says prop (SOME (SLc slCommand))::
              ins) s outs) ∧
    (M, Oi, Os) sat prop (SOME (SLc slCommand))
```

[PlatoonLeader_slCommand_lemma]

```
⊢ CFGInterpret (M, Oi, Os)
    (CFG authenticationTest ssmPBStateInterp
       (secContext slCommand)
       (Name PlatoonLeader says prop (SOME (SLc slCommand))::
            ins) s outs) ⇒
  (M, Oi, Os) sat prop (SOME (SLc slCommand))
```

6 PBTypeIntegrated Theory

Built: 13 May 2018
Parent Theories: OMNIType

6.1 Datatypes

```
omniCommand = ssmPlanPBComplete | ssmMoveToORPComplete
            | ssmConductORPComplete | ssmMoveToPBComplete
            | ssmConductPBComplete | invalidOmniCommand

plCommand = crossLD | conductORP | moveToPB | conductPB
          | completePB | incomplete

slCommand = PL PBTypeIntegrated$plCommand | OMNI omniCommand

slOutput = PlanPB | MoveToORP | ConductORP | MoveToPB
         | ConductPB | CompletePB | unAuthenticated
         | unAuthorized
```

slState = PLAN_PB | MOVE_TO_ORP | CONDUCT_ORP | MOVE_TO_PB
| CONDUCT_PB | COMPLETE_PB

stateRole = PlatoonLeader | Omni

6.2 Theorems

[omniCommand_distinct_clauses]

⊢ ssmPlanPBComplete ≠ ssmMoveToORPComplete ∧
ssmPlanPBComplete ≠ ssmConductORPComplete ∧
ssmPlanPBComplete ≠ ssmMoveToPBComplete ∧
ssmPlanPBComplete ≠ ssmConductPBComplete ∧
ssmPlanPBComplete ≠ invalidOmniCommand ∧
ssmMoveToORPComplete ≠ ssmConductORPComplete ∧
ssmMoveToORPComplete ≠ ssmMoveToPBComplete ∧
ssmMoveToORPComplete ≠ ssmConductPBComplete ∧
ssmMoveToORPComplete ≠ invalidOmniCommand ∧
ssmConductORPComplete ≠ ssmMoveToPBComplete ∧
ssmConductORPComplete ≠ ssmConductPBComplete ∧
ssmConductORPComplete ≠ invalidOmniCommand ∧
ssmMoveToPBComplete ≠ ssmConductPBComplete ∧
ssmMoveToPBComplete ≠ invalidOmniCommand ∧
ssmConductPBComplete ≠ invalidOmniCommand

[plCommand_distinct_clauses]

⊢ crossLD ≠ conductORP ∧ crossLD ≠ moveToPB ∧
crossLD ≠ conductPB ∧ crossLD ≠ completePB ∧
crossLD ≠ incomplete ∧ conductORP ≠ moveToPB ∧
conductORP ≠ conductPB ∧ conductORP ≠ completePB ∧
conductORP ≠ incomplete ∧ moveToPB ≠ conductPB ∧
moveToPB ≠ completePB ∧ moveToPB ≠ incomplete ∧
conductPB ≠ completePB ∧ conductPB ≠ incomplete ∧
completePB ≠ incomplete

[slCommand_distinct_clauses]

⊢ ∀ a' a. PL a ≠ OMNI a'

[slCommand_one_one]

⊢ (∀ a a'. (PL a = PL a') ⟺ (a = a')) ∧
∀ a a'. (OMNI a = OMNI a') ⟺ (a = a')

[slOutput_distinct_clauses]

⊢ PlanPB ≠ MoveToORP ∧ PlanPB ≠ ConductORP ∧
PlanPB ≠ MoveToPB ∧ PlanPB ≠ ConductPB ∧
PlanPB ≠ CompletePB ∧ PlanPB ≠ unAuthenticated ∧
PlanPB ≠ unAuthorized ∧ MoveToORP ≠ ConductORP ∧
MoveToORP ≠ MoveToPB ∧ MoveToORP ≠ ConductPB ∧
MoveToORP ≠ CompletePB ∧ MoveToORP ≠ unAuthenticated ∧

```
   MoveToORP ≠ unAuthorized ∧ ConductORP ≠ MoveToPB ∧
   ConductORP ≠ ConductPB ∧ ConductORP ≠ CompletePB ∧
   ConductORP ≠ unAuthenticated ∧ ConductORP ≠ unAuthorized ∧
   MoveToPB ≠ ConductPB ∧ MoveToPB ≠ CompletePB ∧
   MoveToPB ≠ unAuthenticated ∧ MoveToPB ≠ unAuthorized ∧
   ConductPB ≠ CompletePB ∧ ConductPB ≠ unAuthenticated ∧
   ConductPB ≠ unAuthorized ∧ CompletePB ≠ unAuthenticated ∧
   CompletePB ≠ unAuthorized ∧ unAuthenticated ≠ unAuthorized
```

[slState_distinct_clauses]

```
⊢ PLAN_PB ≠ MOVE_TO_ORP ∧ PLAN_PB ≠ CONDUCT_ORP ∧
  PLAN_PB ≠ MOVE_TO_PB ∧ PLAN_PB ≠ CONDUCT_PB ∧
  PLAN_PB ≠ COMPLETE_PB ∧ MOVE_TO_ORP ≠ CONDUCT_ORP ∧
  MOVE_TO_ORP ≠ MOVE_TO_PB ∧ MOVE_TO_ORP ≠ CONDUCT_PB ∧
  MOVE_TO_ORP ≠ COMPLETE_PB ∧ CONDUCT_ORP ≠ MOVE_TO_PB ∧
  CONDUCT_ORP ≠ CONDUCT_PB ∧ CONDUCT_ORP ≠ COMPLETE_PB ∧
  MOVE_TO_PB ≠ CONDUCT_PB ∧ MOVE_TO_PB ≠ COMPLETE_PB ∧
  CONDUCT_PB ≠ COMPLETE_PB
```

[stateRole_distinct_clauses]

```
⊢ PlatoonLeader ≠ Omni
```

7 PBIntegratedDef Theory

Built: 13 May 2018

Parent Theories: PBTypeIntegrated, aclfoundation

7.1 Definitions

[secAuthorization_def]

```
⊢ ∀xs. secAuthorization xs = secHelper (getOmniCommand xs)
```

[secHelper_def]

```
⊢ ∀cmd.
    secHelper cmd =
    [Name Omni controls prop (SOME (SLc (OMNI cmd)))]
```

7.2 Theorems

[getOmniCommand_def]

```
⊢ (getOmniCommand [] = invalidOmniCommand) ∧
  (∀xs cmd.
     getOmniCommand
       (Name Omni controls prop (SOME (SLc (OMNI cmd)))::xs) =
     cmd) ∧
  (∀xs. getOmniCommand (TT::xs) = getOmniCommand xs) ∧
```

```
(∀ xs. getOmniCommand (FF::xs) = getOmniCommand xs) ∧
(∀ xs v2. getOmniCommand (prop v2::xs) = getOmniCommand xs) ∧
(∀ xs v3. getOmniCommand (notf v3::xs) = getOmniCommand xs) ∧
(∀ xs v5 v4.
   getOmniCommand (v4 andf v5::xs) = getOmniCommand xs) ∧
(∀ xs v7 v6.
   getOmniCommand (v6 orf v7::xs) = getOmniCommand xs) ∧
(∀ xs v9 v8.
   getOmniCommand (v8 impf v9::xs) = getOmniCommand xs) ∧
(∀ xs v11 v10.
   getOmniCommand (v10 eqf v11::xs) = getOmniCommand xs) ∧
(∀ xs v13 v12.
   getOmniCommand (v12 says v13::xs) = getOmniCommand xs) ∧
(∀ xs v15 v14.
   getOmniCommand (v14 speaks_for v15::xs) =
   getOmniCommand xs) ∧
(∀ xs v16.
   getOmniCommand (v16 controls TT::xs) =
   getOmniCommand xs) ∧
(∀ xs v16.
   getOmniCommand (v16 controls FF::xs) =
   getOmniCommand xs) ∧
(∀ xs v134.
   getOmniCommand (Name v134 controls prop NONE::xs) =
   getOmniCommand xs) ∧
(∀ xs v144.
   getOmniCommand
     (Name PlatoonLeader controls prop (SOME v144)::xs) =
   getOmniCommand xs) ∧
(∀ xs v146.
   getOmniCommand
     (Name Omni controls prop (SOME (ESCc v146))::xs) =
   getOmniCommand xs) ∧
(∀ xs v150.
   getOmniCommand
     (Name Omni controls prop (SOME (SLc (PL v150)))::xs) =
   getOmniCommand xs) ∧
(∀ xs v68 v136 v135.
   getOmniCommand (v135 meet v136 controls prop v68::xs) =
   getOmniCommand xs) ∧
(∀ xs v68 v138 v137.
   getOmniCommand (v137 quoting v138 controls prop v68::xs) =
   getOmniCommand xs) ∧
(∀ xs v69 v16.
   getOmniCommand (v16 controls notf v69::xs) =
   getOmniCommand xs) ∧
(∀ xs v71 v70 v16.
   getOmniCommand (v16 controls (v70 andf v71)::xs) =
   getOmniCommand xs) ∧
```

```
(∀ xs v73 v72 v16 .
   getOmniCommand (v16 controls (v72 orf v73)::xs) =
   getOmniCommand xs) ∧
(∀ xs v75 v74 v16 .
   getOmniCommand (v16 controls (v74 impf v75)::xs) =
   getOmniCommand xs) ∧
(∀ xs v77 v76 v16 .
   getOmniCommand (v16 controls (v76 eqf v77)::xs) =
   getOmniCommand xs) ∧
(∀ xs v79 v78 v16 .
   getOmniCommand (v16 controls v78 says v79::xs) =
   getOmniCommand xs) ∧
(∀ xs v81 v80 v16 .
   getOmniCommand (v16 controls v80 speaks_for v81::xs) =
   getOmniCommand xs) ∧
(∀ xs v83 v82 v16 .
   getOmniCommand (v16 controls v82 controls v83::xs) =
   getOmniCommand xs) ∧
(∀ xs v86 v85 v84 v16 .
   getOmniCommand (v16 controls reps v84 v85 v86::xs) =
   getOmniCommand xs) ∧
(∀ xs v88 v87 v16 .
   getOmniCommand (v16 controls v87 domi v88::xs) =
   getOmniCommand xs) ∧
(∀ xs v90 v89 v16 .
   getOmniCommand (v16 controls v89 eqi v90::xs) =
   getOmniCommand xs) ∧
(∀ xs v92 v91 v16 .
   getOmniCommand (v16 controls v91 doms v92::xs) =
   getOmniCommand xs) ∧
(∀ xs v94 v93 v16 .
   getOmniCommand (v16 controls v93 eqs v94::xs) =
   getOmniCommand xs) ∧
(∀ xs v96 v95 v16 .
   getOmniCommand (v16 controls v95 eqn v96::xs) =
   getOmniCommand xs) ∧
(∀ xs v98 v97 v16 .
   getOmniCommand (v16 controls v97 lte v98::xs) =
   getOmniCommand xs) ∧
(∀ xs v99 v16 v100 .
   getOmniCommand (v16 controls v99 lt v100::xs) =
   getOmniCommand xs) ∧
(∀ xs v20 v19 v18 .
   getOmniCommand (reps v18 v19 v20::xs) =
   getOmniCommand xs) ∧
(∀ xs v22 v21 .
   getOmniCommand (v21 domi v22::xs) = getOmniCommand xs) ∧
(∀ xs v24 v23 .
   getOmniCommand (v23 eqi v24::xs) = getOmniCommand xs) ∧
```

```
   (∀ xs v26 v25.
      getOmniCommand (v25 doms v26::xs) = getOmniCommand xs) ∧
   (∀ xs v28 v27.
      getOmniCommand (v27 eqs v28::xs) = getOmniCommand xs) ∧
   (∀ xs v30 v29.
      getOmniCommand (v29 eqn v30::xs) = getOmniCommand xs) ∧
   (∀ xs v32 v31.
      getOmniCommand (v31 lte v32::xs) = getOmniCommand xs) ∧
  ∀ xs v34 v33.
    getOmniCommand (v33 lt v34::xs) = getOmniCommand xs
```

[getOmniCommand_ind]

```
⊢ ∀ P.
    P [] ∧
    (∀ cmd xs.
       P
         (Name Omni controls prop (SOME (SLc (OMNI cmd)))::
              xs)) ∧ (∀ xs. P xs ⇒ P (TT::xs)) ∧
    (∀ xs. P xs ⇒ P (FF::xs)) ∧
    (∀ v2 xs. P xs ⇒ P (prop v2::xs)) ∧
    (∀ v3 xs. P xs ⇒ P (notf v3::xs)) ∧
    (∀ v4 v5 xs. P xs ⇒ P (v4 andf v5::xs)) ∧
    (∀ v6 v7 xs. P xs ⇒ P (v6 orf v7::xs)) ∧
    (∀ v8 v9 xs. P xs ⇒ P (v8 impf v9::xs)) ∧
    (∀ v10 v11 xs. P xs ⇒ P (v10 eqf v11::xs)) ∧
    (∀ v12 v13 xs. P xs ⇒ P (v12 says v13::xs)) ∧
    (∀ v14 v15 xs. P xs ⇒ P (v14 speaks_for v15::xs)) ∧
    (∀ v16 xs. P xs ⇒ P (v16 controls TT::xs)) ∧
    (∀ v16 xs. P xs ⇒ P (v16 controls FF::xs)) ∧
    (∀ v134 xs. P xs ⇒ P (Name v134 controls prop NONE::xs)) ∧
    (∀ v144 xs.
       P xs ⇒
       P (Name PlatoonLeader controls prop (SOME v144)::xs)) ∧
    (∀ v146 xs.
       P xs ⇒
       P (Name Omni controls prop (SOME (ESCc v146))::xs)) ∧
    (∀ v150 xs.
       P xs ⇒
       P
         (Name Omni controls prop (SOME (SLc (PL v150)))::
              xs)) ∧
    (∀ v135 v136 v68 xs.
       P xs ⇒ P (v135 meet v136 controls prop v68::xs)) ∧
    (∀ v137 v138 v68 xs.
       P xs ⇒ P (v137 quoting v138 controls prop v68::xs)) ∧
    (∀ v16 v69 xs. P xs ⇒ P (v16 controls notf v69::xs)) ∧
    (∀ v16 v70 v71 xs.
       P xs ⇒ P (v16 controls (v70 andf v71)::xs)) ∧
    (∀ v16 v72 v73 xs.
```

$$
\begin{array}{l}
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ (v_{72}\ \texttt{orf}\ v_{73})::xs)) \wedge \\
(\forall v_{16}\ v_{74}\ v_{75}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ (v_{74}\ \texttt{impf}\ v_{75})::xs)) \wedge \\
(\forall v_{16}\ v_{76}\ v_{77}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ (v_{76}\ \texttt{eqf}\ v_{77})::xs)) \wedge \\
(\forall v_{16}\ v_{78}\ v_{79}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{78}\ \texttt{says}\ v_{79}::xs)) \wedge \\
(\forall v_{16}\ v_{80}\ v_{81}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{80}\ \texttt{speaks\_for}\ v_{81}::xs)) \wedge \\
(\forall v_{16}\ v_{82}\ v_{83}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{82}\ \texttt{controls}\ v_{83}::xs)) \wedge \\
(\forall v_{16}\ v_{84}\ v_{85}\ v_{86}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls reps}\ v_{84}\ v_{85}\ v_{86}::xs)) \wedge \\
(\forall v_{16}\ v_{87}\ v_{88}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{87}\ \texttt{domi}\ v_{88}::xs)) \wedge \\
(\forall v_{16}\ v_{89}\ v_{90}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{89}\ \texttt{eqi}\ v_{90}::xs)) \wedge \\
(\forall v_{16}\ v_{91}\ v_{92}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{91}\ \texttt{doms}\ v_{92}::xs)) \wedge \\
(\forall v_{16}\ v_{93}\ v_{94}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{93}\ \texttt{eqs}\ v_{94}::xs)) \wedge \\
(\forall v_{16}\ v_{95}\ v_{96}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{95}\ \texttt{eqn}\ v_{96}::xs)) \wedge \\
(\forall v_{16}\ v_{97}\ v_{98}\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{97}\ \texttt{lte}\ v_{98}::xs)) \wedge \\
(\forall v_{16}\ v_{99}\ v100\ xs. \\
\quad P\ xs \Rightarrow P\ (v_{16}\ \texttt{controls}\ v_{99}\ \texttt{lt}\ v100::xs)) \wedge \\
(\forall v_{18}\ v_{19}\ v_{20}\ xs.\ P\ xs \Rightarrow P\ (\texttt{reps}\ v_{18}\ v_{19}\ v_{20}::xs)) \wedge \\
(\forall v_{21}\ v_{22}\ xs.\ P\ xs \Rightarrow P\ (v_{21}\ \texttt{domi}\ v_{22}::xs)) \wedge \\
(\forall v_{23}\ v_{24}\ xs.\ P\ xs \Rightarrow P\ (v_{23}\ \texttt{eqi}\ v_{24}::xs)) \wedge \\
(\forall v_{25}\ v_{26}\ xs.\ P\ xs \Rightarrow P\ (v_{25}\ \texttt{doms}\ v_{26}::xs)) \wedge \\
(\forall v_{27}\ v_{28}\ xs.\ P\ xs \Rightarrow P\ (v_{27}\ \texttt{eqs}\ v_{28}::xs)) \wedge \\
(\forall v_{29}\ v_{30}\ xs.\ P\ xs \Rightarrow P\ (v_{29}\ \texttt{eqn}\ v_{30}::xs)) \wedge \\
(\forall v_{31}\ v_{32}\ xs.\ P\ xs \Rightarrow P\ (v_{31}\ \texttt{lte}\ v_{32}::xs)) \wedge \\
(\forall v_{33}\ v_{34}\ xs.\ P\ xs \Rightarrow P\ (v_{33}\ \texttt{lt}\ v_{34}::xs)) \Rightarrow \\
\forall v.\ P\ v
\end{array}
$$

[secContext_def]

```
⊢ (secContext PLAN_PB (x::xs) =
    [prop (SOME (SLc (OMNI ssmPlanPBComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL crossLD)))]) ∧
   (secContext MOVE_TO_ORP (x::xs) =
    [prop (SOME (SLc (OMNI ssmMoveToORPComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL conductORP)))]) ∧
   (secContext CONDUCT_ORP (x::xs) =
    [prop (SOME (SLc (OMNI ssmConductORPComplete))) impf
     Name PlatoonLeader controls
```

```
     prop (SOME (SLc (PL moveToPB)))]) ∧
   (secContext MOVE_TO_PB (x::xs) =
    [prop (SOME (SLc (OMNI ssmMoveToPBComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL conductPB)))]) ∧
   (secContext CONDUCT_PB (x::xs) =
    [prop (SOME (SLc (OMNI ssmConductPBComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL completePB)))])
```

[secContext_ind]

```
⊢ ∀P.
    (∀x xs. P PLAN_PB (x::xs)) ∧
    (∀x xs. P MOVE_TO_ORP (x::xs)) ∧
    (∀x xs. P CONDUCT_ORP (x::xs)) ∧
    (∀x xs. P MOVE_TO_PB (x::xs)) ∧
    (∀x xs. P CONDUCT_PB (x::xs)) ∧ (∀v4. P v4 []) ∧
    (∀v5 v6. P COMPLETE_PB (v5::v6)) ⇒
    ∀v v1. P v v1
```

8 ssmConductORP Theory

Built: 13 May 2018

Parent Theories: ConductORPType, ssm11, OMNIType

8.1 Definitions

[secContextConductORP_def]

```
⊢ ∀plcmd psgcmd incomplete.
    secContextConductORP plcmd psgcmd incomplete =
    [Name PlatoonLeader controls prop (SOME (SLc (PL plcmd)));
     Name PlatoonSergeant controls
     prop (SOME (SLc (PSG psgcmd)));
     Name PlatoonLeader says
     prop (SOME (SLc (PSG psgcmd))) impf prop NONE;
     Name PlatoonSergeant says
     prop (SOME (SLc (PL plcmd))) impf prop NONE]
```

[ssmConductORPStateInterp_def]

```
⊢ ∀slState. ssmConductORPStateInterp slState = TT
```

8.2 Theorems

[authTestConductORP_cmd_reject_lemma]

```
⊢ ∀cmd. ¬authTestConductORP (prop (SOME cmd))
```

[authTestConductORP_def]

$\vdash$ (authTestConductORP (Name PlatoonLeader says prop cmd) $\iff$ T) $\wedge$
(authTestConductORP (Name PlatoonSergeant says prop cmd) $\iff$ T) $\wedge$ (authTestConductORP TT $\iff$ F) $\wedge$
(authTestConductORP FF $\iff$ F) $\wedge$
(authTestConductORP (prop v) $\iff$ F) $\wedge$
(authTestConductORP (notf v_1) $\iff$ F) $\wedge$
(authTestConductORP (v_2 andf v_3) $\iff$ F) $\wedge$
(authTestConductORP (v_4 orf v_5) $\iff$ F) $\wedge$
(authTestConductORP (v_6 impf v_7) $\iff$ F) $\wedge$
(authTestConductORP (v_8 eqf v_9) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says TT) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says FF) $\iff$ F) $\wedge$
(authTestConductORP (*v133* meet *v134* says prop v_{66}) $\iff$ F) $\wedge$
(authTestConductORP (*v135* quoting *v136* says prop v_{66}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says notf v_{67}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says (v_{68} andf v_{69})) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says (v_{70} orf v_{71})) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says (v_{72} impf v_{73})) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says (v_{74} eqf v_{75})) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{76} says v_{77}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{78} speaks_for v_{79}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{80} controls v_{81}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says reps v_{82} v_{83} v_{84}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{85} domi v_{86}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{87} eqi v_{88}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{89} doms v_{90}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{91} eqs v_{92}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{93} eqn v_{94}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{95} lte v_{96}) $\iff$ F) $\wedge$
(authTestConductORP (v_{10} says v_{97} lt v_{98}) $\iff$ F) $\wedge$
(authTestConductORP (v_{12} speaks_for v_{13}) $\iff$ F) $\wedge$
(authTestConductORP (v_{14} controls v_{15}) $\iff$ F) $\wedge$
(authTestConductORP (reps v_{16} v_{17} v_{18}) $\iff$ F) $\wedge$
(authTestConductORP (v_{19} domi v_{20}) $\iff$ F) $\wedge$
(authTestConductORP (v_{21} eqi v_{22}) $\iff$ F) $\wedge$
(authTestConductORP (v_{23} doms v_{24}) $\iff$ F) $\wedge$
(authTestConductORP (v_{25} eqs v_{26}) $\iff$ F) $\wedge$
(authTestConductORP (v_{27} eqn v_{28}) $\iff$ F) $\wedge$
(authTestConductORP (v_{29} lte v_{30}) $\iff$ F) $\wedge$
(authTestConductORP (v_{31} lt v_{32}) $\iff$ F)

[authTestConductORP_ind]

$\vdash \forall P.$
$(\forall cmd.\ P$ (Name PlatoonLeader says prop cmd)) $\wedge$
$(\forall cmd.\ P$ (Name PlatoonSergeant says prop cmd)) $\wedge$ P TT $\wedge$
P FF $\wedge$ $(\forall v.\ P$ (prop v)) $\wedge$ $(\forall v_1.\ P$ (notf v_1)) $\wedge$
$(\forall v_2\ v_3.\ P$ (v_2 andf v_3)) $\wedge$ $(\forall v_4\ v_5.\ P$ (v_4 orf v_5)) $\wedge$

$(\forall\, v_6\ v_7.\ P\ (v_6$ impf $v_7)) \wedge (\forall\, v_8\ v_9.\ P\ (v_8$ eqf $v_9)) \wedge$
$(\forall\, v_{10}.\ P\ (v_{10}$ says TT$)) \wedge (\forall\, v_{10}.\ P\ (v_{10}$ says FF$)) \wedge$
$(\forall\, v133\ v134\ v_{66}.\ P\ (v133$ meet $v134$ says prop $v_{66})) \wedge$
$(\forall\, v135\ v136\ v_{66}.\ P\ (v135$ quoting $v136$ says prop $v_{66})) \wedge$
$(\forall\, v_{10}\ v_{67}.\ P\ (v_{10}$ says notf $v_{67})) \wedge$
$(\forall\, v_{10}\ v_{68}\ v_{69}.\ P\ (v_{10}$ says $(v_{68}$ andf $v_{69}))) \wedge$
$(\forall\, v_{10}\ v_{70}\ v_{71}.\ P\ (v_{10}$ says $(v_{70}$ orf $v_{71}))) \wedge$
$(\forall\, v_{10}\ v_{72}\ v_{73}.\ P\ (v_{10}$ says $(v_{72}$ impf $v_{73}))) \wedge$
$(\forall\, v_{10}\ v_{74}\ v_{75}.\ P\ (v_{10}$ says $(v_{74}$ eqf $v_{75}))) \wedge$
$(\forall\, v_{10}\ v_{76}\ v_{77}.\ P\ (v_{10}$ says v_{76} says $v_{77})) \wedge$
$(\forall\, v_{10}\ v_{78}\ v_{79}.\ P\ (v_{10}$ says v_{78} speaks_for $v_{79})) \wedge$
$(\forall\, v_{10}\ v_{80}\ v_{81}.\ P\ (v_{10}$ says v_{80} controls $v_{81})) \wedge$
$(\forall\, v_{10}\ v_{82}\ v_{83}\ v_{84}.\ P\ (v_{10}$ says reps $v_{82}\ v_{83}\ v_{84})) \wedge$
$(\forall\, v_{10}\ v_{85}\ v_{86}.\ P\ (v_{10}$ says v_{85} domi $v_{86})) \wedge$
$(\forall\, v_{10}\ v_{87}\ v_{88}.\ P\ (v_{10}$ says v_{87} eqi $v_{88})) \wedge$
$(\forall\, v_{10}\ v_{89}\ v_{90}.\ P\ (v_{10}$ says v_{89} doms $v_{90})) \wedge$
$(\forall\, v_{10}\ v_{91}\ v_{92}.\ P\ (v_{10}$ says v_{91} eqs $v_{92})) \wedge$
$(\forall\, v_{10}\ v_{93}\ v_{94}.\ P\ (v_{10}$ says v_{93} eqn $v_{94})) \wedge$
$(\forall\, v_{10}\ v_{95}\ v_{96}.\ P\ (v_{10}$ says v_{95} lte $v_{96})) \wedge$
$(\forall\, v_{10}\ v_{97}\ v_{98}.\ P\ (v_{10}$ says v_{97} lt $v_{98})) \wedge$
$(\forall\, v_{12}\ v_{13}.\ P\ (v_{12}$ speaks_for $v_{13})) \wedge$
$(\forall\, v_{14}\ v_{15}.\ P\ (v_{14}$ controls $v_{15})) \wedge$
$(\forall\, v_{16}\ v_{17}\ v_{18}.\ P\ ($reps $v_{16}\ v_{17}\ v_{18})) \wedge$
$(\forall\, v_{19}\ v_{20}.\ P\ (v_{19}$ domi $v_{20})) \wedge$
$(\forall\, v_{21}\ v_{22}.\ P\ (v_{21}$ eqi $v_{22})) \wedge$
$(\forall\, v_{23}\ v_{24}.\ P\ (v_{23}$ doms $v_{24})) \wedge$
$(\forall\, v_{25}\ v_{26}.\ P\ (v_{25}$ eqs $v_{26})) \wedge (\forall\, v_{27}\ v_{28}.\ P\ (v_{27}$ eqn $v_{28})) \wedge$
$(\forall\, v_{29}\ v_{30}.\ P\ (v_{29}$ lte $v_{30})) \wedge (\forall\, v_{31}\ v_{32}.\ P\ (v_{31}$ lt $v_{32})) \Rightarrow$
$\forall\, v.\ P\ v$

[conductORPNS_def]

```
⊢ (conductORPNS CONDUCT_ORP (exec (PL secure)) = SECURE) ∧
  (conductORPNS CONDUCT_ORP (exec (PL plIncomplete)) =
   CONDUCT_ORP) ∧
  (conductORPNS SECURE (exec (PSG actionsIn)) = ACTIONS_IN) ∧
  (conductORPNS SECURE (exec (PSG psgIncomplete)) = SECURE) ∧
  (conductORPNS ACTIONS_IN (exec (PL withdraw)) = WITHDRAW) ∧
  (conductORPNS ACTIONS_IN (exec (PL plIncomplete)) =
   ACTIONS_IN) ∧
  (conductORPNS WITHDRAW (exec (PL complete)) = COMPLETE) ∧
  (conductORPNS WITHDRAW (exec (PL plIncomplete)) = WITHDRAW) ∧
  (conductORPNS s (trap (PL cmd')) = s) ∧
  (conductORPNS s (trap (PSG cmd)) = s) ∧
  (conductORPNS s (discard (PL cmd')) = s) ∧
  (conductORPNS s (discard (PSG cmd)) = s)
```

[conductORPNS_ind]

```
⊢ ∀P.
    P CONDUCT_ORP (exec (PL secure)) ∧
```

```
     P CONDUCT_ORP (exec (PL plIncomplete)) ∧
     P SECURE (exec (PSG actionsIn)) ∧
     P SECURE (exec (PSG psgIncomplete)) ∧
     P ACTIONS_IN (exec (PL withdraw)) ∧
     P ACTIONS_IN (exec (PL plIncomplete)) ∧
     P WITHDRAW (exec (PL complete)) ∧
     P WITHDRAW (exec (PL plIncomplete)) ∧
     (∀ s cmd. P s (trap (PL cmd))) ∧
     (∀ s cmd. P s (trap (PSG cmd))) ∧
     (∀ s cmd. P s (discard (PL cmd))) ∧
     (∀ s cmd. P s (discard (PSG cmd))) ∧
     P CONDUCT_ORP (exec (PL withdraw)) ∧
     P CONDUCT_ORP (exec (PL complete)) ∧
     (∀ v11. P CONDUCT_ORP (exec (PSG v11))) ∧
     (∀ v13. P SECURE (exec (PL v13))) ∧
     P ACTIONS_IN (exec (PL secure)) ∧
     P ACTIONS_IN (exec (PL complete)) ∧
     (∀ v17. P ACTIONS_IN (exec (PSG v17))) ∧
     P WITHDRAW (exec (PL secure)) ∧
     P WITHDRAW (exec (PL withdraw)) ∧
     (∀ v20. P WITHDRAW (exec (PSG v20))) ∧
     (∀ v21. P COMPLETE (exec v21)) ⇒
     ∀ v v1. P v v1
```

[conductORPOut_def]

```
⊢ (conductORPOut CONDUCT_ORP (exec (PL secure)) = Secure) ∧
  (conductORPOut CONDUCT_ORP (exec (PL plIncomplete)) =
   ConductORP) ∧
  (conductORPOut SECURE (exec (PSG actionsIn)) = ActionsIn) ∧
  (conductORPOut SECURE (exec (PSG psgIncomplete)) = Secure) ∧
  (conductORPOut ACTIONS_IN (exec (PL withdraw)) = Withdraw) ∧
  (conductORPOut ACTIONS_IN (exec (PL plIncomplete)) =
   ActionsIn) ∧
  (conductORPOut WITHDRAW (exec (PL complete)) = Complete) ∧
  (conductORPOut WITHDRAW (exec (PL plIncomplete)) =
   Withdraw) ∧
  (conductORPOut s (trap (PL cmd')) = unAuthorized) ∧
  (conductORPOut s (trap (PSG cmd)) = unAuthorized) ∧
  (conductORPOut s (discard (PL cmd')) = unAuthenticated) ∧
  (conductORPOut s (discard (PSG cmd)) = unAuthenticated)
```

[conductORPOut_ind]

```
⊢ ∀ P.
     P CONDUCT_ORP (exec (PL secure)) ∧
     P CONDUCT_ORP (exec (PL plIncomplete)) ∧
     P SECURE (exec (PSG actionsIn)) ∧
     P SECURE (exec (PSG psgIncomplete)) ∧
     P ACTIONS_IN (exec (PL withdraw)) ∧
     P ACTIONS_IN (exec (PL plIncomplete)) ∧
```

P WITHDRAW (exec (PL complete)) $\wedge$
P WITHDRAW (exec (PL plIncomplete)) $\wedge$
$(\forall s\ cmd.\ P\ s$ (trap (PL cmd))) $\wedge$
$(\forall s\ cmd.\ P\ s$ (trap (PSG cmd))) $\wedge$
$(\forall s\ cmd.\ P\ s$ (discard (PL cmd))) $\wedge$
$(\forall s\ cmd.\ P\ s$ (discard (PSG cmd))) $\wedge$
P CONDUCT_ORP (exec (PL withdraw)) $\wedge$
P CONDUCT_ORP (exec (PL complete)) $\wedge$
$(\forall v_{11}.\ P$ CONDUCT_ORP (exec (PSG v_{11}))) $\wedge$
$(\forall v_{13}.\ P$ SECURE (exec (PL v_{13}))) $\wedge$
P ACTIONS_IN (exec (PL secure)) $\wedge$
P ACTIONS_IN (exec (PL complete)) $\wedge$
$(\forall v_{17}.\ P$ ACTIONS_IN (exec (PSG v_{17}))) $\wedge$
P WITHDRAW (exec (PL secure)) $\wedge$
P WITHDRAW (exec (PL withdraw)) $\wedge$
$(\forall v_{20}.\ P$ WITHDRAW (exec (PSG v_{20}))) $\wedge$
$(\forall v_{21}.\ P$ COMPLETE (exec v_{21})) $\Rightarrow$
$\forall v\ v_1.\ P\ v\ v_1$

[PlatoonLeader_exec_plCommand_justified_thm]

$\vdash \forall NS\ Out\ M\ Oi\ Os.$
TR (M, Oi, Os) (exec (SLc (PL $plCommand$)))
(CFG authTestConductORP ssmConductORPStateInterp
(secContextConductORP $plCommand$ $psgCommand$ $incomplete$)
(Name PlatoonLeader says
prop (SOME (SLc (PL $plCommand$)))::ins) s $outs$)
(CFG authTestConductORP ssmConductORPStateInterp
(secContextConductORP $plCommand$ $psgCommand$ $incomplete$)
ins (NS s (exec (SLc (PL $plCommand$))))
(Out s (exec (SLc (PL $plCommand$)))::$outs$)) $\Longleftrightarrow$
authTestConductORP
(Name PlatoonLeader says
prop (SOME (SLc (PL $plCommand$)))) $\wedge$
CFGInterpret (M, Oi, Os)
(CFG authTestConductORP ssmConductORPStateInterp
(secContextConductORP $plCommand$ $psgCommand$ $incomplete$)
(Name PlatoonLeader says
prop (SOME (SLc (PL $plCommand$)))::ins) s $outs$) $\wedge$
(M, Oi, Os) sat prop (SOME (SLc (PL $plCommand$)))

[PlatoonLeader_plCommand_lemma]

$\vdash$ CFGInterpret (M, Oi, Os)
(CFG authTestConductORP ssmConductORPStateInterp
(secContextConductORP $plCommand$ $psgCommand$ $incomplete$)
(Name PlatoonLeader says
prop (SOME (SLc (PL $plCommand$)))::ins) s $outs$) $\Rightarrow$
(M, Oi, Os) sat prop (SOME (SLc (PL $plCommand$)))

[PlatoonSergeant_exec_psgCommand_justified_thm]
```
⊢ ∀NS Out M Oi Os.
    TR (M,Oi,Os) (exec (SLc (PSG psgCommand)))
      (CFG authTestConductORP ssmConductORPStateInterp
         (secContextConductORP plCommand psgCommand incomplete)
         (Name PlatoonSergeant says
          prop (SOME (SLc (PSG psgCommand)))::ins) s outs)
      (CFG authTestConductORP ssmConductORPStateInterp
         (secContextConductORP plCommand psgCommand incomplete)
         ins (NS s (exec (SLc (PSG psgCommand))))
         (Out s (exec (SLc (PSG psgCommand)))::outs)) ⟺
    authTestConductORP
      (Name PlatoonSergeant says
       prop (SOME (SLc (PSG psgCommand)))) ∧
    CFGInterpret (M,Oi,Os)
      (CFG authTestConductORP ssmConductORPStateInterp
         (secContextConductORP plCommand psgCommand incomplete)
         (Name PlatoonSergeant says
          prop (SOME (SLc (PSG psgCommand)))::ins) s outs) ∧
    (M,Oi,Os) sat prop (SOME (SLc (PSG psgCommand)))
```

[PlatoonSergeant_psgCommand_lemma]
```
⊢ CFGInterpret (M,Oi,Os)
    (CFG authTestConductORP ssmConductORPStateInterp
       (secContextConductORP plCommand psgCommand incomplete)
       (Name PlatoonSergeant says
        prop (SOME (SLc (PSG psgCommand)))::ins) s outs) ⇒
  (M,Oi,Os) sat prop (SOME (SLc (PSG psgCommand)))
```

9 ConductORPType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

9.1 Datatypes

```
plCommand = secure | withdraw | complete | plIncomplete

psgCommand = actionsIn | psgIncomplete

slCommand =
    PL ConductORPType$plCommand
  | PSG ConductORPType$psgCommand

slOutput = ConductORP | Secure | ActionsIn | Withdraw | Complete
         | unAuthenticated | unAuthorized

slState = CONDUCT_ORP | SECURE | ACTIONS_IN | WITHDRAW
        | COMPLETE

stateRole = PlatoonLeader | PlatoonSergeant
```

9.2 Theorems

[plCommand_distinct_clauses]

$\vdash$ secure $\neq$ withdraw $\wedge$ secure $\neq$ complete $\wedge$
secure $\neq$ plIncomplete $\wedge$ withdraw $\neq$ complete $\wedge$
withdraw $\neq$ plIncomplete $\wedge$ complete $\neq$ plIncomplete

[psgCommand_distinct_clauses]

$\vdash$ actionsIn $\neq$ psgIncomplete

[slCommand_distinct_clauses]

$\vdash \forall a'\ a.$ PL $a \neq$ PSG a'

[slCommand_one_one]

$\vdash (\forall a\ a'.$ (PL a = PL a') $\iff$ $(a = a')) \wedge$
$\forall a\ a'.$ (PSG a = PSG a') $\iff$ $(a = a')$

[slOutput_distinct_clauses]

$\vdash$ ConductORP $\neq$ Secure $\wedge$ ConductORP $\neq$ ActionsIn $\wedge$
ConductORP $\neq$ Withdraw $\wedge$ ConductORP $\neq$ Complete $\wedge$
ConductORP $\neq$ unAuthenticated $\wedge$ ConductORP $\neq$ unAuthorized $\wedge$
Secure $\neq$ ActionsIn $\wedge$ Secure $\neq$ Withdraw $\wedge$ Secure $\neq$ Complete $\wedge$
Secure $\neq$ unAuthenticated $\wedge$ Secure $\neq$ unAuthorized $\wedge$
ActionsIn $\neq$ Withdraw $\wedge$ ActionsIn $\neq$ Complete $\wedge$
ActionsIn $\neq$ unAuthenticated $\wedge$ ActionsIn $\neq$ unAuthorized $\wedge$
Withdraw $\neq$ Complete $\wedge$ Withdraw $\neq$ unAuthenticated $\wedge$
Withdraw $\neq$ unAuthorized $\wedge$ Complete $\neq$ unAuthenticated $\wedge$
Complete $\neq$ unAuthorized $\wedge$ unAuthenticated $\neq$ unAuthorized

[slRole_distinct_clauses]

$\vdash$ PlatoonLeader $\neq$ PlatoonSergeant

[slState_distinct_clauses]

$\vdash$ CONDUCT_ORP $\neq$ SECURE $\wedge$ CONDUCT_ORP $\neq$ ACTIONS_IN $\wedge$
CONDUCT_ORP $\neq$ WITHDRAW $\wedge$ CONDUCT_ORP $\neq$ COMPLETE $\wedge$
SECURE $\neq$ ACTIONS_IN $\wedge$ SECURE $\neq$ WITHDRAW $\wedge$ SECURE $\neq$ COMPLETE $\wedge$
ACTIONS_IN $\neq$ WITHDRAW $\wedge$ ACTIONS_IN $\neq$ COMPLETE $\wedge$
WITHDRAW $\neq$ COMPLETE

10 ssmConductPB Theory

Built: 13 May 2018
Parent Theories: ConductPBType, ssm11, OMNIType

10.1 Definitions

[secContextConductPB_def]

⊢ ∀ *plcmd psgcmd incomplete*.
secContextConductPB *plcmd psgcmd incomplete* =
[Name PlatoonLeader controls prop (SOME (SLc (PL *plcmd*)));
Name PlatoonSergeant controls
prop (SOME (SLc (PSG *psgcmd*)));
Name PlatoonLeader says
prop (SOME (SLc (PSG *psgcmd*))) impf prop NONE;
Name PlatoonSergeant says
prop (SOME (SLc (PL *plcmd*))) impf prop NONE]

[ssmConductPBStateInterp_def]

⊢ ∀ *slState*. ssmConductPBStateInterp *slState* = TT

10.2 Theorems

[authTestConductPB_cmd_reject_lemma]

⊢ ∀ *cmd*. ¬authTestConductPB (prop (SOME *cmd*))

[authTestConductPB_def]

⊢ (authTestConductPB (Name PlatoonLeader says prop *cmd*) ⟺ T) ∧
(authTestConductPB (Name PlatoonSergeant says prop *cmd*) ⟺
T) ∧ (authTestConductPB TT ⟺ F) ∧
(authTestConductPB FF ⟺ F) ∧
(authTestConductPB (prop v) ⟺ F) ∧
(authTestConductPB (notf v_1) ⟺ F) ∧
(authTestConductPB (v_2 andf v_3) ⟺ F) ∧
(authTestConductPB (v_4 orf v_5) ⟺ F) ∧
(authTestConductPB (v_6 impf v_7) ⟺ F) ∧
(authTestConductPB (v_8 eqf v_9) ⟺ F) ∧
(authTestConductPB (v_{10} says TT) ⟺ F) ∧
(authTestConductPB (v_{10} says FF) ⟺ F) ∧
(authTestConductPB (*v133* meet *v134* says prop v_{66}) ⟺ F) ∧
(authTestConductPB (*v135* quoting *v136* says prop v_{66}) ⟺ F) ∧
(authTestConductPB (v_{10} says notf v_{67}) ⟺ F) ∧
(authTestConductPB (v_{10} says (v_{68} andf v_{69})) ⟺ F) ∧
(authTestConductPB (v_{10} says (v_{70} orf v_{71})) ⟺ F) ∧
(authTestConductPB (v_{10} says (v_{72} impf v_{73})) ⟺ F) ∧
(authTestConductPB (v_{10} says (v_{74} eqf v_{75})) ⟺ F) ∧
(authTestConductPB (v_{10} says v_{76} says v_{77}) ⟺ F) ∧
(authTestConductPB (v_{10} says v_{78} speaks_for v_{79}) ⟺ F) ∧
(authTestConductPB (v_{10} says v_{80} controls v_{81}) ⟺ F) ∧
(authTestConductPB (v_{10} says reps v_{82} v_{83} v_{84}) ⟺ F) ∧
(authTestConductPB (v_{10} says v_{85} domi v_{86}) ⟺ F) ∧
(authTestConductPB (v_{10} says v_{87} eqi v_{88}) ⟺ F) ∧
(authTestConductPB (v_{10} says v_{89} doms v_{90}) ⟺ F) ∧

(authTestConductPB (v_{10} says v_{91} eqs v_{92}) $\iff$ F) $\wedge$
(authTestConductPB (v_{10} says v_{93} eqn v_{94}) $\iff$ F) $\wedge$
(authTestConductPB (v_{10} says v_{95} lte v_{96}) $\iff$ F) $\wedge$
(authTestConductPB (v_{10} says v_{97} lt v_{98}) $\iff$ F) $\wedge$
(authTestConductPB (v_{12} speaks_for v_{13}) $\iff$ F) $\wedge$
(authTestConductPB (v_{14} controls v_{15}) $\iff$ F) $\wedge$
(authTestConductPB (reps v_{16} v_{17} v_{18}) $\iff$ F) $\wedge$
(authTestConductPB (v_{19} domi v_{20}) $\iff$ F) $\wedge$
(authTestConductPB (v_{21} eqi v_{22}) $\iff$ F) $\wedge$
(authTestConductPB (v_{23} doms v_{24}) $\iff$ F) $\wedge$
(authTestConductPB (v_{25} eqs v_{26}) $\iff$ F) $\wedge$
(authTestConductPB (v_{27} eqn v_{28}) $\iff$ F) $\wedge$
(authTestConductPB (v_{29} lte v_{30}) $\iff$ F) $\wedge$
(authTestConductPB (v_{31} lt v_{32}) $\iff$ F)

[authTestConductPB_ind]

$\vdash \forall P.$
(∀ *cmd*. P (Name PlatoonLeader says prop *cmd*)) $\wedge$
(∀ *cmd*. P (Name PlatoonSergeant says prop *cmd*)) $\wedge$ P TT $\wedge$
P FF $\wedge$ (∀ v. P (prop v)) $\wedge$ (∀ v_1. P (notf v_1)) $\wedge$
(∀ v_2 v_3. P (v_2 andf v_3)) $\wedge$ (∀ v_4 v_5. P (v_4 orf v_5)) $\wedge$
(∀ v_6 v_7. P (v_6 impf v_7)) $\wedge$ (∀ v_8 v_9. P (v_8 eqf v_9)) $\wedge$
(∀ v_{10}. P (v_{10} says TT)) $\wedge$ (∀ v_{10}. P (v_{10} says FF)) $\wedge$
(∀ *v133* *v134* v_{66}. P (*v133* meet *v134* says prop v_{66})) $\wedge$
(∀ *v135* *v136* v_{66}. P (*v135* quoting *v136* says prop v_{66})) $\wedge$
(∀ v_{10} v_{67}. P (v_{10} says notf v_{67})) $\wedge$
(∀ v_{10} v_{68} v_{69}. P (v_{10} says (v_{68} andf v_{69}))) $\wedge$
(∀ v_{10} v_{70} v_{71}. P (v_{10} says (v_{70} orf v_{71}))) $\wedge$
(∀ v_{10} v_{72} v_{73}. P (v_{10} says (v_{72} impf v_{73}))) $\wedge$
(∀ v_{10} v_{74} v_{75}. P (v_{10} says (v_{74} eqf v_{75}))) $\wedge$
(∀ v_{10} v_{76} v_{77}. P (v_{10} says v_{76} says v_{77})) $\wedge$
(∀ v_{10} v_{78} v_{79}. P (v_{10} says v_{78} speaks_for v_{79})) $\wedge$
(∀ v_{10} v_{80} v_{81}. P (v_{10} says v_{80} controls v_{81})) $\wedge$
(∀ v_{10} v_{82} v_{83} v_{84}. P (v_{10} says reps v_{82} v_{83} v_{84})) $\wedge$
(∀ v_{10} v_{85} v_{86}. P (v_{10} says v_{85} domi v_{86})) $\wedge$
(∀ v_{10} v_{87} v_{88}. P (v_{10} says v_{87} eqi v_{88})) $\wedge$
(∀ v_{10} v_{89} v_{90}. P (v_{10} says v_{89} doms v_{90})) $\wedge$
(∀ v_{10} v_{91} v_{92}. P (v_{10} says v_{91} eqs v_{92})) $\wedge$
(∀ v_{10} v_{93} v_{94}. P (v_{10} says v_{93} eqn v_{94})) $\wedge$
(∀ v_{10} v_{95} v_{96}. P (v_{10} says v_{95} lte v_{96})) $\wedge$
(∀ v_{10} v_{97} v_{98}. P (v_{10} says v_{97} lt v_{98})) $\wedge$
(∀ v_{12} v_{13}. P (v_{12} speaks_for v_{13})) $\wedge$
(∀ v_{14} v_{15}. P (v_{14} controls v_{15})) $\wedge$
(∀ v_{16} v_{17} v_{18}. P (reps v_{16} v_{17} v_{18})) $\wedge$
(∀ v_{19} v_{20}. P (v_{19} domi v_{20})) $\wedge$
(∀ v_{21} v_{22}. P (v_{21} eqi v_{22})) $\wedge$
(∀ v_{23} v_{24}. P (v_{23} doms v_{24})) $\wedge$
(∀ v_{25} v_{26}. P (v_{25} eqs v_{26})) $\wedge$ (∀ v_{27} v_{28}. P (v_{27} eqn v_{28})) $\wedge$
(∀ v_{29} v_{30}. P (v_{29} lte v_{30})) $\wedge$ (∀ v_{31} v_{32}. P (v_{31} lt v_{32})) $\Rightarrow$

$\forall v.\ P\ v$

[conductPBNS_def]

$\vdash$ (conductPBNS CONDUCT_PB (exec (PL securePB)) = SECURE_PB) $\wedge$
(conductPBNS CONDUCT_PB (exec (PL plIncompletePB)) =
CONDUCT_PB) $\wedge$
(conductPBNS SECURE_PB (exec (PSG actionsInPB)) =
ACTIONS_IN_PB) $\wedge$
(conductPBNS SECURE_PB (exec (PSG psgIncompletePB)) =
SECURE_PB) $\wedge$
(conductPBNS ACTIONS_IN_PB (exec (PL withdrawPB)) =
WITHDRAW_PB) $\wedge$
(conductPBNS ACTIONS_IN_PB (exec (PL plIncompletePB)) =
ACTIONS_IN_PB) $\wedge$
(conductPBNS WITHDRAW_PB (exec (PL completePB)) =
COMPLETE_PB) $\wedge$
(conductPBNS WITHDRAW_PB (exec (PL plIncompletePB)) =
WITHDRAW_PB) $\wedge$ (conductPBNS s (trap (PL cmd')) = s) $\wedge$
(conductPBNS s (trap (PSG cmd)) = s) $\wedge$
(conductPBNS s (discard (PL cmd')) = s) $\wedge$
(conductPBNS s (discard (PSG cmd)) = s)

[conductPBNS_ind]

$\vdash \forall P.$
P CONDUCT_PB (exec (PL securePB)) $\wedge$
P CONDUCT_PB (exec (PL plIncompletePB)) $\wedge$
P SECURE_PB (exec (PSG actionsInPB)) $\wedge$
P SECURE_PB (exec (PSG psgIncompletePB)) $\wedge$
P ACTIONS_IN_PB (exec (PL withdrawPB)) $\wedge$
P ACTIONS_IN_PB (exec (PL plIncompletePB)) $\wedge$
P WITHDRAW_PB (exec (PL completePB)) $\wedge$
P WITHDRAW_PB (exec (PL plIncompletePB)) $\wedge$
($\forall s\ cmd.\ P\ s$ (trap (PL cmd))) $\wedge$
($\forall s\ cmd.\ P\ s$ (trap (PSG cmd))) $\wedge$
($\forall s\ cmd.\ P\ s$ (discard (PL cmd))) $\wedge$
($\forall s\ cmd.\ P\ s$ (discard (PSG cmd))) $\wedge$
P CONDUCT_PB (exec (PL withdrawPB)) $\wedge$
P CONDUCT_PB (exec (PL completePB)) $\wedge$
($\forall v_{11}.\ P$ CONDUCT_PB (exec (PSG v_{11}))) $\wedge$
($\forall v_{13}.\ P$ SECURE_PB (exec (PL v_{13}))) $\wedge$
P ACTIONS_IN_PB (exec (PL securePB)) $\wedge$
P ACTIONS_IN_PB (exec (PL completePB)) $\wedge$
($\forall v_{17}.\ P$ ACTIONS_IN_PB (exec (PSG v_{17}))) $\wedge$
P WITHDRAW_PB (exec (PL securePB)) $\wedge$
P WITHDRAW_PB (exec (PL withdrawPB)) $\wedge$
($\forall v_{20}.\ P$ WITHDRAW_PB (exec (PSG v_{20}))) $\wedge$
($\forall v_{21}.\ P$ COMPLETE_PB (exec v_{21})) $\Rightarrow$
$\forall v\ v_1.\ P\ v\ v_1$

[conductPBOut_def]

$$
\begin{aligned}
&\vdash (\texttt{conductPBOut CONDUCT\_PB (exec (PL securePB))} = \texttt{ConductPB}) \wedge \\
&\quad (\texttt{conductPBOut CONDUCT\_PB (exec (PL plIncompletePB))} = \\
&\quad\ \texttt{ConductPB}) \wedge \\
&\quad (\texttt{conductPBOut SECURE\_PB (exec (PSG actionsInPB))} = \\
&\quad\ \texttt{SecurePB}) \wedge \\
&\quad (\texttt{conductPBOut SECURE\_PB (exec (PSG psgIncompletePB))} = \\
&\quad\ \texttt{SecurePB}) \wedge \\
&\quad (\texttt{conductPBOut ACTIONS\_IN\_PB (exec (PL withdrawPB))} = \\
&\quad\ \texttt{ActionsInPB}) \wedge \\
&\quad (\texttt{conductPBOut ACTIONS\_IN\_PB (exec (PL plIncompletePB))} = \\
&\quad\ \texttt{ActionsInPB}) \wedge \\
&\quad (\texttt{conductPBOut WITHDRAW\_PB (exec (PL completePB))} = \\
&\quad\ \texttt{WithdrawPB}) \wedge \\
&\quad (\texttt{conductPBOut WITHDRAW\_PB (exec (PL plIncompletePB))} = \\
&\quad\ \texttt{WithdrawPB}) \wedge \\
&\quad (\texttt{conductPBOut } s\ \texttt{(trap (PL } cmd'\texttt{))} = \texttt{unAuthorized}) \wedge \\
&\quad (\texttt{conductPBOut } s\ \texttt{(trap (PSG } cmd\texttt{))} = \texttt{unAuthorized}) \wedge \\
&\quad (\texttt{conductPBOut } s\ \texttt{(discard (PL } cmd'\texttt{))} = \texttt{unAuthenticated}) \wedge \\
&\quad (\texttt{conductPBOut } s\ \texttt{(discard (PSG } cmd\texttt{))} = \texttt{unAuthenticated})
\end{aligned}
$$

[conductPBOut_ind]

$$
\begin{aligned}
&\vdash \forall P. \\
&\quad P\ \texttt{CONDUCT\_PB (exec (PL securePB))} \wedge \\
&\quad P\ \texttt{CONDUCT\_PB (exec (PL plIncompletePB))} \wedge \\
&\quad P\ \texttt{SECURE\_PB (exec (PSG actionsInPB))} \wedge \\
&\quad P\ \texttt{SECURE\_PB (exec (PSG psgIncompletePB))} \wedge \\
&\quad P\ \texttt{ACTIONS\_IN\_PB (exec (PL withdrawPB))} \wedge \\
&\quad P\ \texttt{ACTIONS\_IN\_PB (exec (PL plIncompletePB))} \wedge \\
&\quad P\ \texttt{WITHDRAW\_PB (exec (PL completePB))} \wedge \\
&\quad P\ \texttt{WITHDRAW\_PB (exec (PL plIncompletePB))} \wedge \\
&\quad (\forall s\ cmd.\ P\ s\ \texttt{(trap (PL } cmd\texttt{))}) \wedge \\
&\quad (\forall s\ cmd.\ P\ s\ \texttt{(trap (PSG } cmd\texttt{))}) \wedge \\
&\quad (\forall s\ cmd.\ P\ s\ \texttt{(discard (PL } cmd\texttt{))}) \wedge \\
&\quad (\forall s\ cmd.\ P\ s\ \texttt{(discard (PSG } cmd\texttt{))}) \wedge \\
&\quad P\ \texttt{CONDUCT\_PB (exec (PL withdrawPB))} \wedge \\
&\quad P\ \texttt{CONDUCT\_PB (exec (PL completePB))} \wedge \\
&\quad (\forall v_{11}.\ P\ \texttt{CONDUCT\_PB (exec (PSG } v_{11}\texttt{))}) \wedge \\
&\quad (\forall v_{13}.\ P\ \texttt{SECURE\_PB (exec (PL } v_{13}\texttt{))}) \wedge \\
&\quad P\ \texttt{ACTIONS\_IN\_PB (exec (PL securePB))} \wedge \\
&\quad P\ \texttt{ACTIONS\_IN\_PB (exec (PL completePB))} \wedge \\
&\quad (\forall v_{17}.\ P\ \texttt{ACTIONS\_IN\_PB (exec (PSG } v_{17}\texttt{))}) \wedge \\
&\quad P\ \texttt{WITHDRAW\_PB (exec (PL securePB))} \wedge \\
&\quad P\ \texttt{WITHDRAW\_PB (exec (PL withdrawPB))} \wedge \\
&\quad (\forall v_{20}.\ P\ \texttt{WITHDRAW\_PB (exec (PSG } v_{20}\texttt{))}) \wedge \\
&\quad (\forall v_{21}.\ P\ \texttt{COMPLETE\_PB (exec } v_{21}\texttt{)}) \Rightarrow \\
&\quad \forall v\ v_1.\ P\ v\ v_1
\end{aligned}
$$

[PlatoonLeader_exec_plCommandPB_justified_thm]

```
⊢ ∀NS Out M Oi Os.
    TR (M,Oi,Os) (exec (SLc (PL plCommand)))
      (CFG authTestConductPB ssmConductPBStateInterp
         (secContextConductPB plCommand psgCommand incomplete)
         (Name PlatoonLeader says
          prop (SOME (SLc (PL plCommand)))::ins) s outs)
      (CFG authTestConductPB ssmConductPBStateInterp
         (secContextConductPB plCommand psgCommand incomplete)
         ins (NS s (exec (SLc (PL plCommand))))
         (Out s (exec (SLc (PL plCommand)))::outs)) ⟺
    authTestConductPB
      (Name PlatoonLeader says
       prop (SOME (SLc (PL plCommand)))) ∧
    CFGInterpret (M,Oi,Os)
      (CFG authTestConductPB ssmConductPBStateInterp
         (secContextConductPB plCommand psgCommand incomplete)
         (Name PlatoonLeader says
          prop (SOME (SLc (PL plCommand)))::ins) s outs) ∧
    (M,Oi,Os) sat prop (SOME (SLc (PL plCommand)))
```

[PlatoonLeader_plCommandPB_lemma]

```
⊢ CFGInterpret (M,Oi,Os)
    (CFG authTestConductPB ssmConductPBStateInterp
       (secContextConductPB plCommand psgCommand incomplete)
       (Name PlatoonLeader says
        prop (SOME (SLc (PL plCommand)))::ins) s outs) ⇒
  (M,Oi,Os) sat prop (SOME (SLc (PL plCommand)))
```

[PlatoonSergeant_exec_psgCommandPB_justified_thm]

```
⊢ ∀NS Out M Oi Os.
    TR (M,Oi,Os) (exec (SLc (PSG psgCommand)))
      (CFG authTestConductPB ssmConductPBStateInterp
         (secContextConductPB plCommand psgCommand incomplete)
         (Name PlatoonSergeant says
          prop (SOME (SLc (PSG psgCommand)))::ins) s outs)
      (CFG authTestConductPB ssmConductPBStateInterp
         (secContextConductPB plCommand psgCommand incomplete)
         ins (NS s (exec (SLc (PSG psgCommand))))
         (Out s (exec (SLc (PSG psgCommand)))::outs)) ⟺
    authTestConductPB
      (Name PlatoonSergeant says
       prop (SOME (SLc (PSG psgCommand)))) ∧
    CFGInterpret (M,Oi,Os)
      (CFG authTestConductPB ssmConductPBStateInterp
         (secContextConductPB plCommand psgCommand incomplete)
         (Name PlatoonSergeant says
          prop (SOME (SLc (PSG psgCommand)))::ins) s outs) ∧
    (M,Oi,Os) sat prop (SOME (SLc (PSG psgCommand)))
```

[PlatoonSergeant_psgCommandPB_lemma]

```
⊢ CFGInterpret (M, Oi, Os)
    (CFG authTestConductPB ssmConductPBStateInterp
       (secContextConductPB plCommand psgCommand incomplete)
       (Name PlatoonSergeant says
        prop (SOME (SLc (PSG psgCommand)))::ins) s outs) ⇒
  (M, Oi, Os) sat prop (SOME (SLc (PSG psgCommand)))
```

11 ConductPBType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

11.1 Datatypes

```
plCommandPB = securePB | withdrawPB | completePB
            | plIncompletePB

psgCommandPB = actionsInPB | psgIncompletePB

slCommand = PL plCommandPB | PSG psgCommandPB

slOutput = ConductPB | SecurePB | ActionsInPB | WithdrawPB
         | CompletePB | unAuthenticated | unAuthorized

slState = CONDUCT_PB | SECURE_PB | ACTIONS_IN_PB | WITHDRAW_PB
        | COMPLETE_PB

stateRole = PlatoonLeader | PlatoonSergeant
```

11.2 Theorems

[plCommandPB_distinct_clauses]

```
⊢ securePB ≠ withdrawPB ∧ securePB ≠ completePB ∧
  securePB ≠ plIncompletePB ∧ withdrawPB ≠ completePB ∧
  withdrawPB ≠ plIncompletePB ∧ completePB ≠ plIncompletePB
```

[psgCommandPB_distinct_clauses]

```
⊢ actionsInPB ≠ psgIncompletePB
```

[slCommand_distinct_clauses]

```
⊢ ∀a' a. PL a ≠ PSG a'
```

[slCommand_one_one]

```
⊢ (∀a a'. (PL a = PL a') ⟺ (a = a')) ∧
  ∀a a'. (PSG a = PSG a') ⟺ (a = a')
```

[slOutput_distinct_clauses]

⊢ ConductPB ≠ SecurePB ∧ ConductPB ≠ ActionsInPB ∧
ConductPB ≠ WithdrawPB ∧ ConductPB ≠ CompletePB ∧
ConductPB ≠ unAuthenticated ∧ ConductPB ≠ unAuthorized ∧
SecurePB ≠ ActionsInPB ∧ SecurePB ≠ WithdrawPB ∧
SecurePB ≠ CompletePB ∧ SecurePB ≠ unAuthenticated ∧
SecurePB ≠ unAuthorized ∧ ActionsInPB ≠ WithdrawPB ∧
ActionsInPB ≠ CompletePB ∧ ActionsInPB ≠ unAuthenticated ∧
ActionsInPB ≠ unAuthorized ∧ WithdrawPB ≠ CompletePB ∧
WithdrawPB ≠ unAuthenticated ∧ WithdrawPB ≠ unAuthorized ∧
CompletePB ≠ unAuthenticated ∧ CompletePB ≠ unAuthorized ∧
unAuthenticated ≠ unAuthorized

[slRole_distinct_clauses]

⊢ PlatoonLeader ≠ PlatoonSergeant

[slState_distinct_clauses]

⊢ CONDUCT_PB ≠ SECURE_PB ∧ CONDUCT_PB ≠ ACTIONS_IN_PB ∧
CONDUCT_PB ≠ WITHDRAW_PB ∧ CONDUCT_PB ≠ COMPLETE_PB ∧
SECURE_PB ≠ ACTIONS_IN_PB ∧ SECURE_PB ≠ WITHDRAW_PB ∧
SECURE_PB ≠ COMPLETE_PB ∧ ACTIONS_IN_PB ≠ WITHDRAW_PB ∧
ACTIONS_IN_PB ≠ COMPLETE_PB ∧ WITHDRAW_PB ≠ COMPLETE_PB

12 ssmMoveToORP Theory

Built: 13 May 2018
Parent Theories: MoveToORPType, ssm11, OMNIType

12.1 Definitions

[secContextMoveToORP_def]

⊢ ∀ *cmd*.
secContextMoveToORP *cmd* =
[Name PlatoonLeader controls prop (SOME (SLc *cmd*))]

[ssmMoveToORPStateInterp_def]

⊢ ∀ *state*. ssmMoveToORPStateInterp *state* = TT

12.2 Theorems

[authTestMoveToORP_cmd_reject_lemma]

⊢ ∀ *cmd*. ¬authTestMoveToORP (prop (SOME *cmd*))

[authTestMoveToORP_def]

$\vdash$ (authTestMoveToORP (Name PlatoonLeader says prop cmd) $\iff$ T) $\wedge$
(authTestMoveToORP TT $\iff$ F) $\wedge$ (authTestMoveToORP FF $\iff$ F) $\wedge$
(authTestMoveToORP (prop v) $\iff$ F) $\wedge$
(authTestMoveToORP (notf v_1) $\iff$ F) $\wedge$
(authTestMoveToORP (v_2 andf v_3) $\iff$ F) $\wedge$
(authTestMoveToORP (v_4 orf v_5) $\iff$ F) $\wedge$
(authTestMoveToORP (v_6 impf v_7) $\iff$ F) $\wedge$
(authTestMoveToORP (v_8 eqf v_9) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says TT) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says FF) $\iff$ F) $\wedge$
(authTestMoveToORP ($v133$ meet $v134$ says prop v_{66}) $\iff$ F) $\wedge$
(authTestMoveToORP ($v135$ quoting $v136$ says prop v_{66}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says notf v_{67}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says (v_{68} andf v_{69})) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says (v_{70} orf v_{71})) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says (v_{72} impf v_{73})) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says (v_{74} eqf v_{75})) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{76} says v_{77}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{78} speaks_for v_{79}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{80} controls v_{81}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says reps v_{82} v_{83} v_{84}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{85} domi v_{86}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{87} eqi v_{88}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{89} doms v_{90}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{91} eqs v_{92}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{93} eqn v_{94}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{95} lte v_{96}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{10} says v_{97} lt v_{98}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{12} speaks_for v_{13}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{14} controls v_{15}) $\iff$ F) $\wedge$
(authTestMoveToORP (reps v_{16} v_{17} v_{18}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{19} domi v_{20}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{21} eqi v_{22}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{23} doms v_{24}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{25} eqs v_{26}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{27} eqn v_{28}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{29} lte v_{30}) $\iff$ F) $\wedge$
(authTestMoveToORP (v_{31} lt v_{32}) $\iff$ F)

[authTestMoveToORP_ind]

$\vdash$ $\forall P$.
($\forall cmd$. P (Name PlatoonLeader says prop cmd)) $\wedge$ P TT $\wedge$
P FF $\wedge$ ($\forall v$. P (prop v)) $\wedge$ ($\forall v_1$. P (notf v_1)) $\wedge$
($\forall v_2$ v_3. P (v_2 andf v_3)) $\wedge$ ($\forall v_4$ v_5. P (v_4 orf v_5)) $\wedge$
($\forall v_6$ v_7. P (v_6 impf v_7)) $\wedge$ ($\forall v_8$ v_9. P (v_8 eqf v_9)) $\wedge$
($\forall v_{10}$. P (v_{10} says TT)) $\wedge$ ($\forall v_{10}$. P (v_{10} says FF)) $\wedge$
($\forall v133$ $v134$ v_{66}. P ($v133$ meet $v134$ says prop v_{66})) $\wedge$
($\forall v135$ $v136$ v_{66}. P ($v135$ quoting $v136$ says prop v_{66})) $\wedge$

$(\forall\, v_{10}\ v_{67}.\ P\ (v_{10}\ \texttt{says notf}\ v_{67})) \wedge$
$(\forall\, v_{10}\ v_{68}\ v_{69}.\ P\ (v_{10}\ \texttt{says}\ (v_{68}\ \texttt{andf}\ v_{69}))) \wedge$
$(\forall\, v_{10}\ v_{70}\ v_{71}.\ P\ (v_{10}\ \texttt{says}\ (v_{70}\ \texttt{orf}\ v_{71}))) \wedge$
$(\forall\, v_{10}\ v_{72}\ v_{73}.\ P\ (v_{10}\ \texttt{says}\ (v_{72}\ \texttt{impf}\ v_{73}))) \wedge$
$(\forall\, v_{10}\ v_{74}\ v_{75}.\ P\ (v_{10}\ \texttt{says}\ (v_{74}\ \texttt{eqf}\ v_{75}))) \wedge$
$(\forall\, v_{10}\ v_{76}\ v_{77}.\ P\ (v_{10}\ \texttt{says}\ v_{76}\ \texttt{says}\ v_{77})) \wedge$
$(\forall\, v_{10}\ v_{78}\ v_{79}.\ P\ (v_{10}\ \texttt{says}\ v_{78}\ \texttt{speaks\_for}\ v_{79})) \wedge$
$(\forall\, v_{10}\ v_{80}\ v_{81}.\ P\ (v_{10}\ \texttt{says}\ v_{80}\ \texttt{controls}\ v_{81})) \wedge$
$(\forall\, v_{10}\ v_{82}\ v_{83}\ v_{84}.\ P\ (v_{10}\ \texttt{says reps}\ v_{82}\ v_{83}\ v_{84})) \wedge$
$(\forall\, v_{10}\ v_{85}\ v_{86}.\ P\ (v_{10}\ \texttt{says}\ v_{85}\ \texttt{domi}\ v_{86})) \wedge$
$(\forall\, v_{10}\ v_{87}\ v_{88}.\ P\ (v_{10}\ \texttt{says}\ v_{87}\ \texttt{eqi}\ v_{88})) \wedge$
$(\forall\, v_{10}\ v_{89}\ v_{90}.\ P\ (v_{10}\ \texttt{says}\ v_{89}\ \texttt{doms}\ v_{90})) \wedge$
$(\forall\, v_{10}\ v_{91}\ v_{92}.\ P\ (v_{10}\ \texttt{says}\ v_{91}\ \texttt{eqs}\ v_{92})) \wedge$
$(\forall\, v_{10}\ v_{93}\ v_{94}.\ P\ (v_{10}\ \texttt{says}\ v_{93}\ \texttt{eqn}\ v_{94})) \wedge$
$(\forall\, v_{10}\ v_{95}\ v_{96}.\ P\ (v_{10}\ \texttt{says}\ v_{95}\ \texttt{lte}\ v_{96})) \wedge$
$(\forall\, v_{10}\ v_{97}\ v_{98}.\ P\ (v_{10}\ \texttt{says}\ v_{97}\ \texttt{lt}\ v_{98})) \wedge$
$(\forall\, v_{12}\ v_{13}.\ P\ (v_{12}\ \texttt{speaks\_for}\ v_{13})) \wedge$
$(\forall\, v_{14}\ v_{15}.\ P\ (v_{14}\ \texttt{controls}\ v_{15})) \wedge$
$(\forall\, v_{16}\ v_{17}\ v_{18}.\ P\ (\texttt{reps}\ v_{16}\ v_{17}\ v_{18})) \wedge$
$(\forall\, v_{19}\ v_{20}.\ P\ (v_{19}\ \texttt{domi}\ v_{20})) \wedge$
$(\forall\, v_{21}\ v_{22}.\ P\ (v_{21}\ \texttt{eqi}\ v_{22})) \wedge$
$(\forall\, v_{23}\ v_{24}.\ P\ (v_{23}\ \texttt{doms}\ v_{24})) \wedge$
$(\forall\, v_{25}\ v_{26}.\ P\ (v_{25}\ \texttt{eqs}\ v_{26})) \wedge (\forall\, v_{27}\ v_{28}.\ P\ (v_{27}\ \texttt{eqn}\ v_{28})) \wedge$
$(\forall\, v_{29}\ v_{30}.\ P\ (v_{29}\ \texttt{lte}\ v_{30})) \wedge (\forall\, v_{31}\ v_{32}.\ P\ (v_{31}\ \texttt{lt}\ v_{32})) \Rightarrow$
$\forall\, v.\ P\ v$

[moveToORPNS_def]

$\vdash$ (moveToORPNS MOVE_TO_ORP (exec (SLc pltForm)) = PLT_FORM) $\wedge$
(moveToORPNS MOVE_TO_ORP (exec (SLc incomplete)) =
MOVE_TO_ORP) $\wedge$
(moveToORPNS PLT_FORM (exec (SLc pltMove)) = PLT_MOVE) $\wedge$
(moveToORPNS PLT_FORM (exec (SLc incomplete)) = PLT_FORM) $\wedge$
(moveToORPNS PLT_MOVE (exec (SLc pltSecureHalt)) =
PLT_SECURE_HALT) $\wedge$
(moveToORPNS PLT_MOVE (exec (SLc incomplete)) = PLT_MOVE) $\wedge$
(moveToORPNS PLT_SECURE_HALT (exec (SLc complete)) =
COMPLETE) $\wedge$
(moveToORPNS PLT_SECURE_HALT (exec (SLc incomplete)) =
PLT_SECURE_HALT) $\wedge$ (moveToORPNS s (trap (SLc cmd)) = s) $\wedge$
(moveToORPNS s (discard (SLc cmd)) = s)

[moveToORPNS_ind]

$\vdash \forall\, P.$
P MOVE_TO_ORP (exec (SLc pltForm)) $\wedge$
P MOVE_TO_ORP (exec (SLc incomplete)) $\wedge$
P PLT_FORM (exec (SLc pltMove)) $\wedge$
P PLT_FORM (exec (SLc incomplete)) $\wedge$
P PLT_MOVE (exec (SLc pltSecureHalt)) $\wedge$
P PLT_MOVE (exec (SLc incomplete)) $\wedge$

P PLT_SECURE_HALT (exec (SLc complete)) ∧
P PLT_SECURE_HALT (exec (SLc incomplete)) ∧
(∀ s cmd. P s (trap (SLc cmd))) ∧
(∀ s cmd. P s (discard (SLc cmd))) ∧
(∀ s v_6. P s (discard (ESCc v_6))) ∧
(∀ s v_9. P s (trap (ESCc v_9))) ∧
(∀ v_{12}. P MOVE_TO_ORP (exec (ESCc v_{12}))) ∧
P MOVE_TO_ORP (exec (SLc pltMove)) ∧
P MOVE_TO_ORP (exec (SLc pltSecureHalt)) ∧
P MOVE_TO_ORP (exec (SLc complete)) ∧
(∀ v_{15}. P PLT_FORM (exec (ESCc v_{15}))) ∧
P PLT_FORM (exec (SLc pltForm)) ∧
P PLT_FORM (exec (SLc pltSecureHalt)) ∧
P PLT_FORM (exec (SLc complete)) ∧
(∀ v_{18}. P PLT_MOVE (exec (ESCc v_{18}))) ∧
P PLT_MOVE (exec (SLc pltForm)) ∧
P PLT_MOVE (exec (SLc pltMove)) ∧
P PLT_MOVE (exec (SLc complete)) ∧
(∀ v_{21}. P PLT_SECURE_HALT (exec (ESCc v_{21}))) ∧
P PLT_SECURE_HALT (exec (SLc pltForm)) ∧
P PLT_SECURE_HALT (exec (SLc pltMove)) ∧
P PLT_SECURE_HALT (exec (SLc pltSecureHalt)) ∧
(∀ v_{23}. P COMPLETE (exec v_{23})) ⇒
∀ v v_1. P v v_1

[moveToORPOut_def]

⊢ (moveToORPOut MOVE_TO_ORP (exec (SLc pltForm)) = PLTForm) ∧
(moveToORPOut MOVE_TO_ORP (exec (SLc incomplete)) =
MoveToORP) ∧
(moveToORPOut PLT_FORM (exec (SLc pltMove)) = PLTMove) ∧
(moveToORPOut PLT_FORM (exec (SLc incomplete)) = PLTForm) ∧
(moveToORPOut PLT_MOVE (exec (SLc pltSecureHalt)) =
PLTSecureHalt) ∧
(moveToORPOut PLT_MOVE (exec (SLc incomplete)) = PLTMove) ∧
(moveToORPOut PLT_SECURE_HALT (exec (SLc complete)) =
Complete) ∧
(moveToORPOut PLT_SECURE_HALT (exec (SLc incomplete)) =
PLTSecureHalt) ∧
(moveToORPOut s (trap (SLc cmd)) = unAuthorized) ∧
(moveToORPOut s (discard (SLc cmd)) = unAuthenticated)

[moveToORPOut_ind]

⊢ ∀ P.
P MOVE_TO_ORP (exec (SLc pltForm)) ∧
P MOVE_TO_ORP (exec (SLc incomplete)) ∧
P PLT_FORM (exec (SLc pltMove)) ∧
P PLT_FORM (exec (SLc incomplete)) ∧
P PLT_MOVE (exec (SLc pltSecureHalt)) ∧
P PLT_MOVE (exec (SLc incomplete)) ∧

```
     P PLT_SECURE_HALT (exec (SLc complete)) ∧
     P PLT_SECURE_HALT (exec (SLc incomplete)) ∧
     (∀ s cmd. P s (trap (SLc cmd))) ∧
     (∀ s cmd. P s (discard (SLc cmd))) ∧
     (∀ s v6. P s (discard (ESCc v6))) ∧
     (∀ s v9. P s (trap (ESCc v9))) ∧
     (∀ v12. P MOVE_TO_ORP (exec (ESCc v12))) ∧
     P MOVE_TO_ORP (exec (SLc pltMove)) ∧
     P MOVE_TO_ORP (exec (SLc pltSecureHalt)) ∧
     P MOVE_TO_ORP (exec (SLc complete)) ∧
     (∀ v15. P PLT_FORM (exec (ESCc v15))) ∧
     P PLT_FORM (exec (SLc pltForm)) ∧
     P PLT_FORM (exec (SLc pltSecureHalt)) ∧
     P PLT_FORM (exec (SLc complete)) ∧
     (∀ v18. P PLT_MOVE (exec (ESCc v18))) ∧
     P PLT_MOVE (exec (SLc pltForm)) ∧
     P PLT_MOVE (exec (SLc pltMove)) ∧
     P PLT_MOVE (exec (SLc complete)) ∧
     (∀ v21. P PLT_SECURE_HALT (exec (ESCc v21))) ∧
     P PLT_SECURE_HALT (exec (SLc pltForm)) ∧
     P PLT_SECURE_HALT (exec (SLc pltMove)) ∧
     P PLT_SECURE_HALT (exec (SLc pltSecureHalt)) ∧
     (∀ v23. P COMPLETE (exec v23)) ⇒
     ∀ v v1. P v v1
```

[PlatoonLeader_exec_slCommand_justified_thm]

```
 ⊢ ∀ NS Out M Oi Os.
     TR (M,Oi,Os) (exec (SLc slCommand))
       (CFG authTestMoveToORP ssmMoveToORPStateInterp
          (secContextMoveToORP slCommand)
          (Name PlatoonLeader says prop (SOME (SLc slCommand))::
               ins) s outs)
       (CFG authTestMoveToORP ssmMoveToORPStateInterp
          (secContextMoveToORP slCommand) ins
          (NS s (exec (SLc slCommand)))
          (Out s (exec (SLc slCommand))::outs)) ⟺
     authTestMoveToORP
       (Name PlatoonLeader says prop (SOME (SLc slCommand))) ∧
     CFGInterpret (M,Oi,Os)
       (CFG authTestMoveToORP ssmMoveToORPStateInterp
          (secContextMoveToORP slCommand)
          (Name PlatoonLeader says prop (SOME (SLc slCommand))::
               ins) s outs) ∧
     (M,Oi,Os) sat prop (SOME (SLc slCommand))
```

[PlatoonLeader_slCommand_lemma]

```
 ⊢ CFGInterpret (M,Oi,Os)
     (CFG authTestMoveToORP ssmMoveToORPStateInterp
        (secContextMoveToORP slCommand)
```

```
        (Name PlatoonLeader says prop (SOME (SLc slCommand))::
             ins) s outs) ⇒
   (M,Oi,Os) sat prop (SOME (SLc slCommand))
```

13 MoveToORPType Theory

Built: 13 May 2018

Parent Theories: indexedLists, patternMatches

13.1 Datatypes

slCommand = pltForm | pltMove | pltSecureHalt | complete
| incomplete

slOutput = MoveToORP | PLTForm | PLTMove | PLTSecureHalt
| Complete | unAuthorized | unAuthenticated

slState = MOVE_TO_ORP | PLT_FORM | PLT_MOVE | PLT_SECURE_HALT
| COMPLETE

stateRole = PlatoonLeader

13.2 Theorems

[slCommand_distinct_clauses]

```
⊢ pltForm ≠ pltMove ∧ pltForm ≠ pltSecureHalt ∧
  pltForm ≠ complete ∧ pltForm ≠ incomplete ∧
  pltMove ≠ pltSecureHalt ∧ pltMove ≠ complete ∧
  pltMove ≠ incomplete ∧ pltSecureHalt ≠ complete ∧
  pltSecureHalt ≠ incomplete ∧ complete ≠ incomplete
```

[slOutput_distinct_clauses]

```
⊢ MoveToORP ≠ PLTForm ∧ MoveToORP ≠ PLTMove ∧
  MoveToORP ≠ PLTSecureHalt ∧ MoveToORP ≠ Complete ∧
  MoveToORP ≠ unAuthorized ∧ MoveToORP ≠ unAuthenticated ∧
  PLTForm ≠ PLTMove ∧ PLTForm ≠ PLTSecureHalt ∧
  PLTForm ≠ Complete ∧ PLTForm ≠ unAuthorized ∧
  PLTForm ≠ unAuthenticated ∧ PLTMove ≠ PLTSecureHalt ∧
  PLTMove ≠ Complete ∧ PLTMove ≠ unAuthorized ∧
  PLTMove ≠ unAuthenticated ∧ PLTSecureHalt ≠ Complete ∧
  PLTSecureHalt ≠ unAuthorized ∧
  PLTSecureHalt ≠ unAuthenticated ∧ Complete ≠ unAuthorized ∧
  Complete ≠ unAuthenticated ∧ unAuthorized ≠ unAuthenticated
```

[slState_distinct_clauses]

```
⊢ MOVE_TO_ORP ≠ PLT_FORM ∧ MOVE_TO_ORP ≠ PLT_MOVE ∧
  MOVE_TO_ORP ≠ PLT_SECURE_HALT ∧ MOVE_TO_ORP ≠ COMPLETE ∧
  PLT_FORM ≠ PLT_MOVE ∧ PLT_FORM ≠ PLT_SECURE_HALT ∧
  PLT_FORM ≠ COMPLETE ∧ PLT_MOVE ≠ PLT_SECURE_HALT ∧
  PLT_MOVE ≠ COMPLETE ∧ PLT_SECURE_HALT ≠ COMPLETE
```

14 ssmMoveToPB Theory

Built: 13 May 2018
Parent Theories: MoveToPBType, ssm11, OMNIType

14.1 Definitions

[secContextMoveToPB_def]

$\vdash \forall cmd.$
secContextMoveToPB cmd =
[Name PlatoonLeader controls prop (SOME (SLc cmd))]

[ssmMoveToPBStateInterp_def]

$\vdash \forall state.$ ssmMoveToPBStateInterp $state$ = TT

14.2 Theorems

[authTestMoveToPB_cmd_reject_lemma]

$\vdash \forall cmd.$ ¬authTestMoveToPB (prop (SOME cmd))

[authTestMoveToPB_def]

$\vdash$ (authTestMoveToPB (Name PlatoonLeader says prop cmd) $\iff$ T) $\wedge$
(authTestMoveToPB TT $\iff$ F) $\wedge$ (authTestMoveToPB FF $\iff$ F) $\wedge$
(authTestMoveToPB (prop v) $\iff$ F) $\wedge$
(authTestMoveToPB (notf v_1) $\iff$ F) $\wedge$
(authTestMoveToPB (v_2 andf v_3) $\iff$ F) $\wedge$
(authTestMoveToPB (v_4 orf v_5) $\iff$ F) $\wedge$
(authTestMoveToPB (v_6 impf v_7) $\iff$ F) $\wedge$
(authTestMoveToPB (v_8 eqf v_9) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says TT) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says FF) $\iff$ F) $\wedge$
(authTestMoveToPB (*v133* meet *v134* says prop v_{66}) $\iff$ F) $\wedge$
(authTestMoveToPB (*v135* quoting *v136* says prop v_{66}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says notf v_{67}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says (v_{68} andf v_{69})) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says (v_{70} orf v_{71})) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says (v_{72} impf v_{73})) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says (v_{74} eqf v_{75})) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says v_{76} says v_{77}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says v_{78} speaks_for v_{79}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says v_{80} controls v_{81}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says reps v_{82} v_{83} v_{84}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says v_{85} domi v_{86}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says v_{87} eqi v_{88}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says v_{89} doms v_{90}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says v_{91} eqs v_{92}) $\iff$ F) $\wedge$
(authTestMoveToPB (v_{10} says v_{93} eqn v_{94}) $\iff$ F) $\wedge$

(authTestMoveToPB (v_{10} says v_{95} lte v_{96}) ⟺ F) ∧
(authTestMoveToPB (v_{10} says v_{97} lt v_{98}) ⟺ F) ∧
(authTestMoveToPB (v_{12} speaks_for v_{13}) ⟺ F) ∧
(authTestMoveToPB (v_{14} controls v_{15}) ⟺ F) ∧
(authTestMoveToPB (reps v_{16} v_{17} v_{18}) ⟺ F) ∧
(authTestMoveToPB (v_{19} domi v_{20}) ⟺ F) ∧
(authTestMoveToPB (v_{21} eqi v_{22}) ⟺ F) ∧
(authTestMoveToPB (v_{23} doms v_{24}) ⟺ F) ∧
(authTestMoveToPB (v_{25} eqs v_{26}) ⟺ F) ∧
(authTestMoveToPB (v_{27} eqn v_{28}) ⟺ F) ∧
(authTestMoveToPB (v_{29} lte v_{30}) ⟺ F) ∧
(authTestMoveToPB (v_{31} lt v_{32}) ⟺ F)

[authTestMoveToPB_ind]

⊢ ∀ P.
(∀ cmd. P (Name PlatoonLeader says prop cmd)) ∧ P TT ∧
P FF ∧ (∀ v. P (prop v)) ∧ (∀ v_1. P (notf v_1)) ∧
(∀ v_2 v_3. P (v_2 andf v_3)) ∧ (∀ v_4 v_5. P (v_4 orf v_5)) ∧
(∀ v_6 v_7. P (v_6 impf v_7)) ∧ (∀ v_8 v_9. P (v_8 eqf v_9)) ∧
(∀ v_{10}. P (v_{10} says TT)) ∧ (∀ v_{10}. P (v_{10} says FF)) ∧
(∀ *v133* *v134* v_{66}. P (*v133* meet *v134* says prop v_{66})) ∧
(∀ *v135* *v136* v_{66}. P (*v135* quoting *v136* says prop v_{66})) ∧
(∀ v_{10} v_{67}. P (v_{10} says notf v_{67})) ∧
(∀ v_{10} v_{68} v_{69}. P (v_{10} says (v_{68} andf v_{69}))) ∧
(∀ v_{10} v_{70} v_{71}. P (v_{10} says (v_{70} orf v_{71}))) ∧
(∀ v_{10} v_{72} v_{73}. P (v_{10} says (v_{72} impf v_{73}))) ∧
(∀ v_{10} v_{74} v_{75}. P (v_{10} says (v_{74} eqf v_{75}))) ∧
(∀ v_{10} v_{76} v_{77}. P (v_{10} says v_{76} says v_{77})) ∧
(∀ v_{10} v_{78} v_{79}. P (v_{10} says v_{78} speaks_for v_{79})) ∧
(∀ v_{10} v_{80} v_{81}. P (v_{10} says v_{80} controls v_{81})) ∧
(∀ v_{10} v_{82} v_{83} v_{84}. P (v_{10} says reps v_{82} v_{83} v_{84})) ∧
(∀ v_{10} v_{85} v_{86}. P (v_{10} says v_{85} domi v_{86})) ∧
(∀ v_{10} v_{87} v_{88}. P (v_{10} says v_{87} eqi v_{88})) ∧
(∀ v_{10} v_{89} v_{90}. P (v_{10} says v_{89} doms v_{90})) ∧
(∀ v_{10} v_{91} v_{92}. P (v_{10} says v_{91} eqs v_{92})) ∧
(∀ v_{10} v_{93} v_{94}. P (v_{10} says v_{93} eqn v_{94})) ∧
(∀ v_{10} v_{95} v_{96}. P (v_{10} says v_{95} lte v_{96})) ∧
(∀ v_{10} v_{97} v_{98}. P (v_{10} says v_{97} lt v_{98})) ∧
(∀ v_{12} v_{13}. P (v_{12} speaks_for v_{13})) ∧
(∀ v_{14} v_{15}. P (v_{14} controls v_{15})) ∧
(∀ v_{16} v_{17} v_{18}. P (reps v_{16} v_{17} v_{18})) ∧
(∀ v_{19} v_{20}. P (v_{19} domi v_{20})) ∧
(∀ v_{21} v_{22}. P (v_{21} eqi v_{22})) ∧
(∀ v_{23} v_{24}. P (v_{23} doms v_{24})) ∧
(∀ v_{25} v_{26}. P (v_{25} eqs v_{26})) ∧ (∀ v_{27} v_{28}. P (v_{27} eqn v_{28})) ∧
(∀ v_{29} v_{30}. P (v_{29} lte v_{30})) ∧ (∀ v_{31} v_{32}. P (v_{31} lt v_{32})) ⇒
∀ v. P v

[moveToPBNS_def]

⊢ (moveToPBNS MOVE_TO_PB (exec (SLc pltForm)) = PLT_FORM) ∧
(moveToPBNS MOVE_TO_PB (exec (SLc incomplete)) =
MOVE_TO_PB) ∧
(moveToPBNS PLT_FORM (exec (SLc pltMove)) = PLT_MOVE) ∧
(moveToPBNS PLT_FORM (exec (SLc incomplete)) = PLT_FORM) ∧
(moveToPBNS PLT_MOVE (exec (SLc pltHalt)) = PLT_HALT) ∧
(moveToPBNS PLT_MOVE (exec (SLc incomplete)) = PLT_MOVE) ∧
(moveToPBNS PLT_HALT (exec (SLc complete)) = COMPLETE) ∧
(moveToPBNS PLT_HALT (exec (SLc incomplete)) = PLT_HALT) ∧
(moveToPBNS s (trap (SLc cmd)) = s) ∧
(moveToPBNS s (discard (SLc cmd)) = s)

[moveToPBNS_ind]

⊢ ∀ P.
P MOVE_TO_PB (exec (SLc pltForm)) ∧
P MOVE_TO_PB (exec (SLc incomplete)) ∧
P PLT_FORM (exec (SLc pltMove)) ∧
P PLT_FORM (exec (SLc incomplete)) ∧
P PLT_MOVE (exec (SLc pltHalt)) ∧
P PLT_MOVE (exec (SLc incomplete)) ∧
P PLT_HALT (exec (SLc complete)) ∧
P PLT_HALT (exec (SLc incomplete)) ∧
(∀ s cmd. P s (trap (SLc cmd))) ∧
(∀ s cmd. P s (discard (SLc cmd))) ∧
(∀ s v_6. P s (discard (ESCc v_6))) ∧
(∀ s v_9. P s (trap (ESCc v_9))) ∧
(∀ v_{12}. P MOVE_TO_PB (exec (ESCc v_{12}))) ∧
P MOVE_TO_PB (exec (SLc pltMove)) ∧
P MOVE_TO_PB (exec (SLc pltHalt)) ∧
P MOVE_TO_PB (exec (SLc complete)) ∧
(∀ v_{15}. P PLT_FORM (exec (ESCc v_{15}))) ∧
P PLT_FORM (exec (SLc pltForm)) ∧
P PLT_FORM (exec (SLc pltHalt)) ∧
P PLT_FORM (exec (SLc complete)) ∧
(∀ v_{18}. P PLT_MOVE (exec (ESCc v_{18}))) ∧
P PLT_MOVE (exec (SLc pltForm)) ∧
P PLT_MOVE (exec (SLc pltMove)) ∧
P PLT_MOVE (exec (SLc complete)) ∧
(∀ v_{21}. P PLT_HALT (exec (ESCc v_{21}))) ∧
P PLT_HALT (exec (SLc pltForm)) ∧
P PLT_HALT (exec (SLc pltMove)) ∧
P PLT_HALT (exec (SLc pltHalt)) ∧
(∀ v_{23}. P COMPLETE (exec v_{23})) ⇒
∀ v v_1. P v v_1

[moveToPBOut_def]

⊢ (moveToPBOut MOVE_TO_PB (exec (SLc pltForm)) = PLTForm) ∧
(moveToPBOut MOVE_TO_PB (exec (SLc incomplete)) = MoveToPB) ∧
(moveToPBOut PLT_FORM (exec (SLc pltMove)) = PLTMove) ∧

```
(moveToPBOut PLT_FORM (exec (SLc incomplete)) = PLTForm) ∧
(moveToPBOut PLT_MOVE (exec (SLc pltHalt)) = PLTHalt) ∧
(moveToPBOut PLT_MOVE (exec (SLc incomplete)) = PLTMove) ∧
(moveToPBOut PLT_HALT (exec (SLc complete)) = Complete) ∧
(moveToPBOut PLT_HALT (exec (SLc incomplete)) = PLTHalt) ∧
(moveToPBOut s (trap (SLc cmd)) = unAuthorized) ∧
(moveToPBOut s (discard (SLc cmd)) = unAuthenticated)
```

[moveToPBOut_ind]

```
⊢ ∀P.
    P MOVE_TO_PB (exec (SLc pltForm)) ∧
    P MOVE_TO_PB (exec (SLc incomplete)) ∧
    P PLT_FORM (exec (SLc pltMove)) ∧
    P PLT_FORM (exec (SLc incomplete)) ∧
    P PLT_MOVE (exec (SLc pltHalt)) ∧
    P PLT_MOVE (exec (SLc incomplete)) ∧
    P PLT_HALT (exec (SLc complete)) ∧
    P PLT_HALT (exec (SLc incomplete)) ∧
    (∀s cmd. P s (trap (SLc cmd))) ∧
    (∀s cmd. P s (discard (SLc cmd))) ∧
    (∀s v6. P s (discard (ESCc v6))) ∧
    (∀s v9. P s (trap (ESCc v9))) ∧
    (∀v12. P MOVE_TO_PB (exec (ESCc v12))) ∧
    P MOVE_TO_PB (exec (SLc pltMove)) ∧
    P MOVE_TO_PB (exec (SLc pltHalt)) ∧
    P MOVE_TO_PB (exec (SLc complete)) ∧
    (∀v15. P PLT_FORM (exec (ESCc v15))) ∧
    P PLT_FORM (exec (SLc pltForm)) ∧
    P PLT_FORM (exec (SLc pltHalt)) ∧
    P PLT_FORM (exec (SLc complete)) ∧
    (∀v18. P PLT_MOVE (exec (ESCc v18))) ∧
    P PLT_MOVE (exec (SLc pltForm)) ∧
    P PLT_MOVE (exec (SLc pltMove)) ∧
    P PLT_MOVE (exec (SLc complete)) ∧
    (∀v21. P PLT_HALT (exec (ESCc v21))) ∧
    P PLT_HALT (exec (SLc pltForm)) ∧
    P PLT_HALT (exec (SLc pltMove)) ∧
    P PLT_HALT (exec (SLc pltHalt)) ∧
    (∀v23. P COMPLETE (exec v23)) ⇒
    ∀v v1. P v v1
```

[PlatoonLeader_exec_slCommand_justified_thm]

```
⊢ ∀NS Out M Oi Os.
    TR (M,Oi,Os) (exec (SLc slCommand))
      (CFG authTestMoveToPB ssmMoveToPBStateInterp
         (secContextMoveToPB slCommand)
         (Name PlatoonLeader says prop (SOME (SLc slCommand))::
              ins) s outs)
      (CFG authTestMoveToPB ssmMoveToPBStateInterp
```

```
        (secContextMoveToPB slCommand) ins
        (NS s (exec (SLc slCommand)))
        (Out s (exec (SLc slCommand))::outs)) ⟺
    authTestMoveToPB
      (Name PlatoonLeader says prop (SOME (SLc slCommand))) ∧
    CFGInterpret (M,Oi,Os)
      (CFG authTestMoveToPB ssmMoveToPBStateInterp
        (secContextMoveToPB slCommand)
        (Name PlatoonLeader says prop (SOME (SLc slCommand))::
             ins) s outs) ∧
    (M,Oi,Os) sat prop (SOME (SLc slCommand))
```

[PlatoonLeader_slCommand_lemma]

```
⊢ CFGInterpret (M,Oi,Os)
    (CFG authTestMoveToPB ssmMoveToPBStateInterp
       (secContextMoveToPB slCommand)
       (Name PlatoonLeader says prop (SOME (SLc slCommand))::
            ins) s outs) ⇒
  (M,Oi,Os) sat prop (SOME (SLc slCommand))
```

15 MoveToPBType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

15.1 Datatypes

```
slCommand = pltForm | pltMove | pltHalt | complete | incomplete

slOutput = MoveToPB | PLTForm | PLTMove | PLTHalt | Complete
         | unAuthorized | unAuthenticated

slState = MOVE_TO_PB | PLT_FORM | PLT_MOVE | PLT_HALT | COMPLETE

stateRole = PlatoonLeader
```

15.2 Theorems

[slCommand_distinct_clauses]

```
⊢ pltForm ≠ pltMove ∧ pltForm ≠ pltHalt ∧ pltForm ≠ complete ∧
  pltForm ≠ incomplete ∧ pltMove ≠ pltHalt ∧
  pltMove ≠ complete ∧ pltMove ≠ incomplete ∧
  pltHalt ≠ complete ∧ pltHalt ≠ incomplete ∧
  complete ≠ incomplete
```

[slOutput_distinct_clauses]

⊢ MoveToPB ≠ PLTForm ∧ MoveToPB ≠ PLTMove ∧
MoveToPB ≠ PLTHalt ∧ MoveToPB ≠ Complete ∧
MoveToPB ≠ unAuthorized ∧ MoveToPB ≠ unAuthenticated ∧
PLTForm ≠ PLTMove ∧ PLTForm ≠ PLTHalt ∧ PLTForm ≠ Complete ∧
PLTForm ≠ unAuthorized ∧ PLTForm ≠ unAuthenticated ∧
PLTMove ≠ PLTHalt ∧ PLTMove ≠ Complete ∧
PLTMove ≠ unAuthorized ∧ PLTMove ≠ unAuthenticated ∧
PLTHalt ≠ Complete ∧ PLTHalt ≠ unAuthorized ∧
PLTHalt ≠ unAuthenticated ∧ Complete ≠ unAuthorized ∧
Complete ≠ unAuthenticated ∧ unAuthorized ≠ unAuthenticated

[slState_distinct_clauses]

⊢ MOVE_TO_PB ≠ PLT_FORM ∧ MOVE_TO_PB ≠ PLT_MOVE ∧
MOVE_TO_PB ≠ PLT_HALT ∧ MOVE_TO_PB ≠ COMPLETE ∧
PLT_FORM ≠ PLT_MOVE ∧ PLT_FORM ≠ PLT_HALT ∧
PLT_FORM ≠ COMPLETE ∧ PLT_MOVE ≠ PLT_HALT ∧
PLT_MOVE ≠ COMPLETE ∧ PLT_HALT ≠ COMPLETE

16 ssmPlanPB Theory

Built: 13 May 2018
Parent Theories: PlanPBDef, ssm

16.1 Theorems

[inputOK_def]

⊢ (inputOK (Name PlatoonLeader says prop *cmd*) ⟺ T) ∧
(inputOK (Name PlatoonSergeant says prop *cmd*) ⟺ T) ∧
(inputOK TT ⟺ F) ∧ (inputOK FF ⟺ F) ∧
(inputOK (prop v) ⟺ F) ∧ (inputOK (notf v_1) ⟺ F) ∧
(inputOK (v_2 andf v_3) ⟺ F) ∧ (inputOK (v_4 orf v_5) ⟺ F) ∧
(inputOK (v_6 impf v_7) ⟺ F) ∧ (inputOK (v_8 eqf v_9) ⟺ F) ∧
(inputOK (v_{10} says TT) ⟺ F) ∧ (inputOK (v_{10} says FF) ⟺ F) ∧
(inputOK (*v133* meet *v134* says prop v_{66}) ⟺ F) ∧
(inputOK (*v135* quoting *v136* says prop v_{66}) ⟺ F) ∧
(inputOK (v_{10} says notf v_{67}) ⟺ F) ∧
(inputOK (v_{10} says (v_{68} andf v_{69})) ⟺ F) ∧
(inputOK (v_{10} says (v_{70} orf v_{71})) ⟺ F) ∧
(inputOK (v_{10} says (v_{72} impf v_{73})) ⟺ F) ∧
(inputOK (v_{10} says (v_{74} eqf v_{75})) ⟺ F) ∧
(inputOK (v_{10} says v_{76} says v_{77}) ⟺ F) ∧
(inputOK (v_{10} says v_{78} speaks_for v_{79}) ⟺ F) ∧
(inputOK (v_{10} says v_{80} controls v_{81}) ⟺ F) ∧
(inputOK (v_{10} says reps v_{82} v_{83} v_{84}) ⟺ F) ∧
(inputOK (v_{10} says v_{85} domi v_{86}) ⟺ F) ∧
(inputOK (v_{10} says v_{87} eqi v_{88}) ⟺ F) ∧

(inputOK (v_{10} says v_{89} doms v_{90}) $\iff$ F) $\wedge$
(inputOK (v_{10} says v_{91} eqs v_{92}) $\iff$ F) $\wedge$
(inputOK (v_{10} says v_{93} eqn v_{94}) $\iff$ F) $\wedge$
(inputOK (v_{10} says v_{95} lte v_{96}) $\iff$ F) $\wedge$
(inputOK (v_{10} says v_{97} lt v_{98}) $\iff$ F) $\wedge$
(inputOK (v_{12} speaks_for v_{13}) $\iff$ F) $\wedge$
(inputOK (v_{14} controls v_{15}) $\iff$ F) $\wedge$
(inputOK (reps v_{16} v_{17} v_{18}) $\iff$ F) $\wedge$
(inputOK (v_{19} domi v_{20}) $\iff$ F) $\wedge$
(inputOK (v_{21} eqi v_{22}) $\iff$ F) $\wedge$
(inputOK (v_{23} doms v_{24}) $\iff$ F) $\wedge$
(inputOK (v_{25} eqs v_{26}) $\iff$ F) $\wedge$ (inputOK (v_{27} eqn v_{28}) $\iff$ F) $\wedge$
(inputOK (v_{29} lte v_{30}) $\iff$ F) $\wedge$ (inputOK (v_{31} lt v_{32}) $\iff$ F)

[inputOK_ind]

$\vdash \forall P.$
(∀ *cmd*. P (Name PlatoonLeader says prop *cmd*)) $\wedge$
(∀ *cmd*. P (Name PlatoonSergeant says prop *cmd*)) $\wedge$ P TT $\wedge$
P FF $\wedge$ (∀ v. P (prop v)) $\wedge$ (∀ v_1. P (notf v_1)) $\wedge$
(∀ v_2 v_3. P (v_2 andf v_3)) $\wedge$ (∀ v_4 v_5. P (v_4 orf v_5)) $\wedge$
(∀ v_6 v_7. P (v_6 impf v_7)) $\wedge$ (∀ v_8 v_9. P (v_8 eqf v_9)) $\wedge$
(∀ v_{10}. P (v_{10} says TT)) $\wedge$ (∀ v_{10}. P (v_{10} says FF)) $\wedge$
(∀ *v133* *v134* v_{66}. P (*v133* meet *v134* says prop v_{66})) $\wedge$
(∀ *v135* *v136* v_{66}. P (*v135* quoting *v136* says prop v_{66})) $\wedge$
(∀ v_{10} v_{67}. P (v_{10} says notf v_{67})) $\wedge$
(∀ v_{10} v_{68} v_{69}. P (v_{10} says (v_{68} andf v_{69}))) $\wedge$
(∀ v_{10} v_{70} v_{71}. P (v_{10} says (v_{70} orf v_{71}))) $\wedge$
(∀ v_{10} v_{72} v_{73}. P (v_{10} says (v_{72} impf v_{73}))) $\wedge$
(∀ v_{10} v_{74} v_{75}. P (v_{10} says (v_{74} eqf v_{75}))) $\wedge$
(∀ v_{10} v_{76} v_{77}. P (v_{10} says v_{76} says v_{77})) $\wedge$
(∀ v_{10} v_{78} v_{79}. P (v_{10} says v_{78} speaks_for v_{79})) $\wedge$
(∀ v_{10} v_{80} v_{81}. P (v_{10} says v_{80} controls v_{81})) $\wedge$
(∀ v_{10} v_{82} v_{83} v_{84}. P (v_{10} says reps v_{82} v_{83} v_{84})) $\wedge$
(∀ v_{10} v_{85} v_{86}. P (v_{10} says v_{85} domi v_{86})) $\wedge$
(∀ v_{10} v_{87} v_{88}. P (v_{10} says v_{87} eqi v_{88})) $\wedge$
(∀ v_{10} v_{89} v_{90}. P (v_{10} says v_{89} doms v_{90})) $\wedge$
(∀ v_{10} v_{91} v_{92}. P (v_{10} says v_{91} eqs v_{92})) $\wedge$
(∀ v_{10} v_{93} v_{94}. P (v_{10} says v_{93} eqn v_{94})) $\wedge$
(∀ v_{10} v_{95} v_{96}. P (v_{10} says v_{95} lte v_{96})) $\wedge$
(∀ v_{10} v_{97} v_{98}. P (v_{10} says v_{97} lt v_{98})) $\wedge$
(∀ v_{12} v_{13}. P (v_{12} speaks_for v_{13})) $\wedge$
(∀ v_{14} v_{15}. P (v_{14} controls v_{15})) $\wedge$
(∀ v_{16} v_{17} v_{18}. P (reps v_{16} v_{17} v_{18})) $\wedge$
(∀ v_{19} v_{20}. P (v_{19} domi v_{20})) $\wedge$
(∀ v_{21} v_{22}. P (v_{21} eqi v_{22})) $\wedge$
(∀ v_{23} v_{24}. P (v_{23} doms v_{24})) $\wedge$
(∀ v_{25} v_{26}. P (v_{25} eqs v_{26})) $\wedge$ (∀ v_{27} v_{28}. P (v_{27} eqn v_{28})) $\wedge$
(∀ v_{29} v_{30}. P (v_{29} lte v_{30})) $\wedge$ (∀ v_{31} v_{32}. P (v_{31} lt v_{32})) $\Rightarrow$
∀ v. P v

[planPBNS_def]

```
⊢ (planPBNS WARNO (exec x) =
   if
     (getRecon x = [SOME (SLc (PL recon))]) ∧
     (getTenativePlan x = [SOME (SLc (PL tentativePlan))]) ∧
     (getReport x = [SOME (SLc (PL report1))]) ∧
     (getInitMove x = [SOME (SLc (PSG initiateMovement))])
   then
     REPORT1
   else WARNO) ∧
  (planPBNS PLAN_PB (exec x) =
   if getPlCom x = receiveMission then RECEIVE_MISSION
   else PLAN_PB) ∧
  (planPBNS RECEIVE_MISSION (exec x) =
   if getPlCom x = warno then WARNO else RECEIVE_MISSION) ∧
  (planPBNS REPORT1 (exec x) =
   if getPlCom x = completePlan then COMPLETE_PLAN
   else REPORT1) ∧
  (planPBNS COMPLETE_PLAN (exec x) =
   if getPlCom x = opoid then OPOID else COMPLETE_PLAN) ∧
  (planPBNS OPOID (exec x) =
   if getPlCom x = supervise then SUPERVISE else OPOID) ∧
  (planPBNS SUPERVISE (exec x) =
   if getPlCom x = report2 then REPORT2 else SUPERVISE) ∧
  (planPBNS REPORT2 (exec x) =
   if getPlCom x = complete then COMPLETE else REPORT2) ∧
  (planPBNS s (trap v0) = s) ∧ (planPBNS s (discard v1) = s)
```

[planPBNS_ind]

```
⊢ ∀P.
    (∀x. P WARNO (exec x)) ∧ (∀x. P PLAN_PB (exec x)) ∧
    (∀x. P RECEIVE_MISSION (exec x)) ∧
    (∀x. P REPORT1 (exec x)) ∧ (∀x. P COMPLETE_PLAN (exec x)) ∧
    (∀x. P OPOID (exec x)) ∧ (∀x. P SUPERVISE (exec x)) ∧
    (∀x. P REPORT2 (exec x)) ∧ (∀s v0. P s (trap v0)) ∧
    (∀s v1. P s (discard v1)) ∧
    (∀v6. P TENTATIVE_PLAN (exec v6)) ∧
    (∀v7. P INITIATE_MOVEMENT (exec v7)) ∧
    (∀v8. P RECON (exec v8)) ∧ (∀v9. P COMPLETE (exec v9)) ⇒
    ∀v v1. P v v1
```

[planPBOut_def]

```
⊢ (planPBOut WARNO (exec x) =
   if
     (getRecon x = [SOME (SLc (PL recon))]) ∧
     (getTenativePlan x = [SOME (SLc (PL tentativePlan))]) ∧
     (getReport x = [SOME (SLc (PL report1))]) ∧
     (getInitMove x = [SOME (SLc (PSG initiateMovement))])
```

```
     then
       Report1
     else unAuthorized) ∧
    (planPBOut PLAN_PB (exec x) =
     if getPlCom x = receiveMission then ReceiveMission
     else unAuthorized) ∧
    (planPBOut RECEIVE_MISSION (exec x) =
     if getPlCom x = warno then Warno else unAuthorized) ∧
    (planPBOut REPORT1 (exec x) =
     if getPlCom x = completePlan then CompletePlan
     else unAuthorized) ∧
    (planPBOut COMPLETE_PLAN (exec x) =
     if getPlCom x = opoid then Opoid else unAuthorized) ∧
    (planPBOut OPOID (exec x) =
     if getPlCom x = supervise then Supervise
     else unAuthorized) ∧
    (planPBOut SUPERVISE (exec x) =
     if getPlCom x = report2 then Report2 else unAuthorized) ∧
    (planPBOut REPORT2 (exec x) =
     if getPlCom x = complete then Complete else unAuthorized) ∧
    (planPBOut s (trap v0) = unAuthorized) ∧
    (planPBOut s (discard v1) = unAuthenticated)
```

[planPBOut_ind]

```
⊢ ∀P.
    (∀x. P WARNO (exec x)) ∧ (∀x. P PLAN_PB (exec x)) ∧
    (∀x. P RECEIVE_MISSION (exec x)) ∧
    (∀x. P REPORT1 (exec x)) ∧ (∀x. P COMPLETE_PLAN (exec x)) ∧
    (∀x. P OPOID (exec x)) ∧ (∀x. P SUPERVISE (exec x)) ∧
    (∀x. P REPORT2 (exec x)) ∧ (∀s v0. P s (trap v0)) ∧
    (∀s v1. P s (discard v1)) ∧
    (∀v6. P TENTATIVE_PLAN (exec v6)) ∧
    (∀v7. P INITIATE_MOVEMENT (exec v7)) ∧
    (∀v8. P RECON (exec v8)) ∧ (∀v9. P COMPLETE (exec v9)) ⇒
    ∀v v1. P v v1
```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma]

```
⊢ s ≠ WARNO ⇒
  plCommand ≠ invalidPlCommand ⇒
  plCommand ≠ report1 ⇒
  ∀NS Out M Oi Os.
    TR (M,Oi,Os)
      (exec
         (inputList
            [Name PlatoonLeader says
             prop (SOME (SLc (PL plCommand)))]))
      (CFG inputOK secContext secContextNull
         ([Name PlatoonLeader says
           prop (SOME (SLc (PL plCommand)))]::ins) s outs)
```

```
      (CFG inputOK secContext secContextNull ins
         (NS s
            (exec
               (inputList
                  [Name PlatoonLeader says
                   prop (SOME (SLc (PL plCommand)))])))
         (Out s
            (exec
               (inputList
                  [Name PlatoonLeader says
                   prop (SOME (SLc (PL plCommand)))]))::
           outs)) ⟺
    authenticationTest inputOK
      [Name PlatoonLeader says
       prop (SOME (SLc (PL plCommand)))] ∧
    CFGInterpret (M,Oi,Os)
      (CFG inputOK secContext secContextNull
         ([Name PlatoonLeader says
           prop (SOME (SLc (PL plCommand)))]::ins) s outs) ∧
    (M,Oi,Os) satList
    propCommandList
      [Name PlatoonLeader says
       prop (SOME (SLc (PL plCommand)))]
```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]

```
⊢ s ≠ WARNO ⇒
  plCommand ≠ invalidPlCommand ⇒
  plCommand ≠ report1 ⇒
  ∀ NS Out M Oi Os.
    TR (M,Oi,Os) (exec [SOME (SLc (PL plCommand))])
      (CFG inputOK secContext secContextNull
         ([Name PlatoonLeader says
           prop (SOME (SLc (PL plCommand)))]::ins) s outs)
      (CFG inputOK secContext secContextNull ins
         (NS s (exec [SOME (SLc (PL plCommand))]))
         (Out s (exec [SOME (SLc (PL plCommand))])::outs)) ⟺
    authenticationTest inputOK
      [Name PlatoonLeader says
       prop (SOME (SLc (PL plCommand)))] ∧
    CFGInterpret (M,Oi,Os)
      (CFG inputOK secContext secContextNull
         ([Name PlatoonLeader says
           prop (SOME (SLc (PL plCommand)))]::ins) s outs) ∧
    (M,Oi,Os) satList [prop (SOME (SLc (PL plCommand)))]
```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma]

```
⊢ s ≠ WARNO ⇒
  plCommand ≠ invalidPlCommand ⇒
  plCommand ≠ report1 ⇒
```

```
   ∀ M  Oi  Os.
     CFGInterpret (M, Oi, Os)
       (CFG inputOK secContext secContextNull
          ([Name PlatoonLeader says
            prop (SOME (SLc (PL plCommand)))]::ins) s outs) ⇒
     (M, Oi, Os) satList
     propCommandList
       [Name PlatoonLeader says
        prop (SOME (SLc (PL plCommand)))]
```

[PlatoonLeader_psgCommand_notDiscard_thm]

```
 ⊢ ∀ NS  Out  M  Oi  Os.
     ¬TR (M, Oi, Os)
        (discard
           (inputList
              [Name PlatoonLeader says
               prop (SOME (SLc (PSG psgCommand)))]))
        (CFG inputOK secContext secContextNull
           ([Name PlatoonLeader says
             prop (SOME (SLc (PSG psgCommand)))]::ins) s outs)
        (CFG inputOK secContext secContextNull ins
           (NS s
              (discard
                 (inputList
                    [Name PlatoonLeader says
                     prop (SOME (SLc (PSG psgCommand)))])))
           (Out s
              (discard
                 (inputList
                    [Name PlatoonLeader says
                     prop (SOME (SLc (PSG psgCommand)))]))::
               outs))
```

[PlatoonLeader_trap_psgCommand_justified_lemma]

```
 ⊢ ∀ NS  Out  M  Oi  Os.
     TR (M, Oi, Os)
       (trap
          (inputList
             [Name PlatoonLeader says
              prop (SOME (SLc (PSG psgCommand)))]))
       (CFG inputOK secContext secContextNull
          ([Name PlatoonLeader says
            prop (SOME (SLc (PSG psgCommand)))]::ins) s outs)
       (CFG inputOK secContext secContextNull ins
          (NS s
             (trap
                (inputList
                   [Name PlatoonLeader says
                    prop (SOME (SLc (PSG psgCommand)))]))))
```

```
          (Out s
             (trap
                (inputList
                   [Name PlatoonLeader says
                    prop (SOME (SLc (PSG psgCommand)))]))::
              outs)) ⟺
     authenticationTest inputOK
       [Name PlatoonLeader says
        prop (SOME (SLc (PSG psgCommand)))] ∧
     CFGInterpret (M,Oi,Os)
       (CFG inputOK secContext secContextNull
          ([Name PlatoonLeader says
            prop (SOME (SLc (PSG psgCommand)))]::ins) s outs) ∧
     (M,Oi,Os) sat prop NONE
```

[PlatoonLeader_trap_psgCommand_lemma]

```
⊢ ∀M Oi Os.
     CFGInterpret (M,Oi,Os)
       (CFG inputOK secContext secContextNull
          ([Name PlatoonLeader says
            prop (SOME (SLc (PSG psgCommand)))]::ins) s outs) ⇒
     (M,Oi,Os) sat prop NONE
```

[PlatoonLeader_WARNO_exec_report1_justified_lemma]

```
⊢ ∀NS Out M Oi Os.
     TR (M,Oi,Os)
       (exec
          (inputList
             [Name PlatoonLeader says
              prop (SOME (SLc (PL recon)));
              Name PlatoonLeader says
              prop (SOME (SLc (PL tentativePlan)));
              Name PlatoonSergeant says
              prop (SOME (SLc (PSG initiateMovement)));
              Name PlatoonLeader says
              prop (SOME (SLc (PL report1)))]))
       (CFG inputOK secContext secContextNull
          ([Name PlatoonLeader says
            prop (SOME (SLc (PL recon)));
            Name PlatoonLeader says
            prop (SOME (SLc (PL tentativePlan)));
            Name PlatoonSergeant says
            prop (SOME (SLc (PSG initiateMovement)));
            Name PlatoonLeader says
            prop (SOME (SLc (PL report1)))]::ins) WARNO outs)
       (CFG inputOK secContext secContextNull ins
          (NS WARNO
             (exec
                (inputList
```

```
                  [Name PlatoonLeader says
                   prop (SOME (SLc (PL recon)));
                   Name PlatoonLeader says
                   prop (SOME (SLc (PL tentativePlan)));
                   Name PlatoonSergeant says
                   prop (SOME (SLc (PSG initiateMovement)));
                   Name PlatoonLeader says
                   prop (SOME (SLc (PL report1)))])))
        (Out WARNO
           (exec
              (inputList
                 [Name PlatoonLeader says
                  prop (SOME (SLc (PL recon)));
                  Name PlatoonLeader says
                  prop (SOME (SLc (PL tentativePlan)));
                  Name PlatoonSergeant says
                  prop (SOME (SLc (PSG initiateMovement)));
                  Name PlatoonLeader says
                  prop (SOME (SLc (PL report1)))]))::outs)) ⟺
    authenticationTest inputOK
      [Name PlatoonLeader says prop (SOME (SLc (PL recon)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL tentativePlan)));
       Name PlatoonSergeant says
       prop (SOME (SLc (PSG initiateMovement)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL report1)))] ∧
    CFGInterpret (M,Oi,Os)
      (CFG inputOK secContext secContextNull
         ([Name PlatoonLeader says
           prop (SOME (SLc (PL recon)));
           Name PlatoonLeader says
           prop (SOME (SLc (PL tentativePlan)));
           Name PlatoonSergeant says
           prop (SOME (SLc (PSG initiateMovement)));
           Name PlatoonLeader says
           prop (SOME (SLc (PL report1)))]::ins) WARNO outs) ∧
    (M,Oi,Os) satList
    propCommandList
      [Name PlatoonLeader says prop (SOME (SLc (PL recon)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL tentativePlan)));
       Name PlatoonSergeant says
       prop (SOME (SLc (PSG initiateMovement)));
       Name PlatoonLeader says prop (SOME (SLc (PL report1)))]
```

[PlatoonLeader_WARNO_exec_report1_justified_thm]

```
⊢ ∀NS Out M Oi Os.
    TR (M,Oi,Os)
```

```
  (exec
     [SOME (SLc (PL recon)); SOME (SLc (PL tentativePlan));
      SOME (SLc (PSG initiateMovement));
      SOME (SLc (PL report1))])
  (CFG inputOK secContext secContextNull
     ([Name PlatoonLeader says
       prop (SOME (SLc (PL recon)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL tentativePlan)));
       Name PlatoonSergeant says
       prop (SOME (SLc (PSG initiateMovement)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL report1)))]::ins) WARNO outs)
  (CFG inputOK secContext secContextNull ins
     (NS WARNO
        (exec
           [SOME (SLc (PL recon));
            SOME (SLc (PL tentativePlan));
            SOME (SLc (PSG initiateMovement));
            SOME (SLc (PL report1))]))
     (Out WARNO
        (exec
           [SOME (SLc (PL recon));
            SOME (SLc (PL tentativePlan));
            SOME (SLc (PSG initiateMovement));
            SOME (SLc (PL report1))]::outs)) ⟺
authenticationTest inputOK
  [Name PlatoonLeader says prop (SOME (SLc (PL recon)));
   Name PlatoonLeader says
   prop (SOME (SLc (PL tentativePlan)));
   Name PlatoonSergeant says
   prop (SOME (SLc (PSG initiateMovement)));
   Name PlatoonLeader says
   prop (SOME (SLc (PL report1)))] ∧
CFGInterpret (M,Oi,Os)
  (CFG inputOK secContext secContextNull
     ([Name PlatoonLeader says
       prop (SOME (SLc (PL recon)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL tentativePlan)));
       Name PlatoonSergeant says
       prop (SOME (SLc (PSG initiateMovement)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL report1)))]::ins) WARNO outs) ∧
(M,Oi,Os) satList
[prop (SOME (SLc (PL recon)));
 prop (SOME (SLc (PL tentativePlan)));
 prop (SOME (SLc (PSG initiateMovement)));
 prop (SOME (SLc (PL report1)))]
```

[PlatoonLeader_WARNO_exec_report1_lemma]

```
⊢ ∀M Oi Os.
    CFGInterpret (M,Oi,Os)
      (CFG inputOK secContext secContextNull
         ([Name PlatoonLeader says
           prop (SOME (SLc (PL recon)));
           Name PlatoonLeader says
           prop (SOME (SLc (PL tentativePlan)));
           Name PlatoonSergeant says
           prop (SOME (SLc (PSG initiateMovement)));
           Name PlatoonLeader says
           prop (SOME (SLc (PL report1)))]::ins) WARNO outs) ⇒
    (M,Oi,Os) satList
    propCommandList
      [Name PlatoonLeader says prop (SOME (SLc (PL recon)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL tentativePlan)));
       Name PlatoonSergeant says
       prop (SOME (SLc (PSG initiateMovement)));
       Name PlatoonLeader says prop (SOME (SLc (PL report1)))]
```

[PlatoonSergeant_trap_plCommand_justified_lemma]

```
⊢ ∀NS Out M Oi Os.
    TR (M,Oi,Os)
      (trap
         (inputList
            [Name PlatoonSergeant says
             prop (SOME (SLc (PL plCommand)))]))
      (CFG inputOK secContext secContextNull
         ([Name PlatoonSergeant says
           prop (SOME (SLc (PL plCommand)))]::ins) s outs)
      (CFG inputOK secContext secContextNull ins
         (NS s
            (trap
               (inputList
                  [Name PlatoonSergeant says
                   prop (SOME (SLc (PL plCommand)))])))
         (Out s
            (trap
               (inputList
                  [Name PlatoonSergeant says
                   prop (SOME (SLc (PL plCommand)))]))::
              outs)) ⟺
    authenticationTest inputOK
      [Name PlatoonSergeant says
       prop (SOME (SLc (PL plCommand)))] ∧
    CFGInterpret (M,Oi,Os)
      (CFG inputOK secContext secContextNull
         ([Name PlatoonSergeant says
```

```
            prop (SOME (SLc (PL plCommand)))]::ins) s outs) ∧
     (M,Oi,Os) sat prop NONE
```

[PlatoonSergeant_trap_plCommand_justified_thm]

```
 ⊢ ∀NS Out M Oi Os.
     TR (M,Oi,Os) (trap [SOME (SLc (PL plCommand))])
       (CFG inputOK secContext secContextNull
          ([Name PlatoonSergeant says
            prop (SOME (SLc (PL plCommand)))]::ins) s outs)
       (CFG inputOK secContext secContextNull ins
          (NS s (trap [SOME (SLc (PL plCommand))]))
          (Out s (trap [SOME (SLc (PL plCommand))])::outs)) ⟺
     authenticationTest inputOK
       [Name PlatoonSergeant says
        prop (SOME (SLc (PL plCommand)))] ∧
     CFGInterpret (M,Oi,Os)
       (CFG inputOK secContext secContextNull
          ([Name PlatoonSergeant says
            prop (SOME (SLc (PL plCommand)))]::ins) s outs) ∧
     (M,Oi,Os) sat prop NONE
```

[PlatoonSergeant_trap_plCommand_lemma]

```
 ⊢ ∀M Oi Os.
     CFGInterpret (M,Oi,Os)
       (CFG inputOK secContext secContextNull
          ([Name PlatoonSergeant says
            prop (SOME (SLc (PL plCommand)))]::ins) s outs) ⇒
     (M,Oi,Os) sat prop NONE
```

17 PlanPBType Theory

Built: 13 May 2018

Parent Theories: indexedLists, patternMatches

17.1 Datatypes

```
plCommand = receiveMission | warno | tentativePlan | recon
          | report1 | completePlan | opoid | supervise | report2
          | complete | plIncomplete | invalidPlCommand

psgCommand = initiateMovement | psgIncomplete
           | invalidPsgCommand

slCommand = PL plCommand | PSG psgCommand

slOutput = PlanPB | ReceiveMission | Warno | TentativePlan
         | InitiateMovement | Recon | Report1 | CompletePlan
         | Opoid | Supervise | Report2 | Complete
         | unAuthenticated | unAuthorized
```

slState = PLAN_PB | RECEIVE_MISSION | WARNO | TENTATIVE_PLAN
| INITIATE_MOVEMENT | RECON | REPORT1 | COMPLETE_PLAN
| OPOID | SUPERVISE | REPORT2 | COMPLETE

stateRole = PlatoonLeader | PlatoonSergeant

17.2 Theorems

[plCommand_distinct_clauses]

⊢ receiveMission ≠ warno ∧ receiveMission ≠ tentativePlan ∧
receiveMission ≠ recon ∧ receiveMission ≠ report1 ∧
receiveMission ≠ completePlan ∧ receiveMission ≠ opoid ∧
receiveMission ≠ supervise ∧ receiveMission ≠ report2 ∧
receiveMission ≠ complete ∧ receiveMission ≠ plIncomplete ∧
receiveMission ≠ invalidPlCommand ∧ warno ≠ tentativePlan ∧
warno ≠ recon ∧ warno ≠ report1 ∧ warno ≠ completePlan ∧
warno ≠ opoid ∧ warno ≠ supervise ∧ warno ≠ report2 ∧
warno ≠ complete ∧ warno ≠ plIncomplete ∧
warno ≠ invalidPlCommand ∧ tentativePlan ≠ recon ∧
tentativePlan ≠ report1 ∧ tentativePlan ≠ completePlan ∧
tentativePlan ≠ opoid ∧ tentativePlan ≠ supervise ∧
tentativePlan ≠ report2 ∧ tentativePlan ≠ complete ∧
tentativePlan ≠ plIncomplete ∧
tentativePlan ≠ invalidPlCommand ∧ recon ≠ report1 ∧
recon ≠ completePlan ∧ recon ≠ opoid ∧ recon ≠ supervise ∧
recon ≠ report2 ∧ recon ≠ complete ∧ recon ≠ plIncomplete ∧
recon ≠ invalidPlCommand ∧ report1 ≠ completePlan ∧
report1 ≠ opoid ∧ report1 ≠ supervise ∧ report1 ≠ report2 ∧
report1 ≠ complete ∧ report1 ≠ plIncomplete ∧
report1 ≠ invalidPlCommand ∧ completePlan ≠ opoid ∧
completePlan ≠ supervise ∧ completePlan ≠ report2 ∧
completePlan ≠ complete ∧ completePlan ≠ plIncomplete ∧
completePlan ≠ invalidPlCommand ∧ opoid ≠ supervise ∧
opoid ≠ report2 ∧ opoid ≠ complete ∧ opoid ≠ plIncomplete ∧
opoid ≠ invalidPlCommand ∧ supervise ≠ report2 ∧
supervise ≠ complete ∧ supervise ≠ plIncomplete ∧
supervise ≠ invalidPlCommand ∧ report2 ≠ complete ∧
report2 ≠ plIncomplete ∧ report2 ≠ invalidPlCommand ∧
complete ≠ plIncomplete ∧ complete ≠ invalidPlCommand ∧
plIncomplete ≠ invalidPlCommand

[psgCommand_distinct_clauses]

⊢ initiateMovement ≠ psgIncomplete ∧
initiateMovement ≠ invalidPsgCommand ∧
psgIncomplete ≠ invalidPsgCommand

[slCommand_distinct_clauses]

⊢ ∀ a' a. PL a ≠ PSG a'

[slCommand_one_one]

$\vdash$ $(\forall a\ a'.\ (\mathtt{PL}\ a = \mathtt{PL}\ a') \iff (a = a')) \wedge$
$\forall a\ a'.\ (\mathtt{PSG}\ a = \mathtt{PSG}\ a') \iff (a = a')$

[slOutput_distinct_clauses]

$\vdash$ PlanPB $\neq$ ReceiveMission $\wedge$ PlanPB $\neq$ Warno $\wedge$
PlanPB $\neq$ TentativePlan $\wedge$ PlanPB $\neq$ InitiateMovement $\wedge$
PlanPB $\neq$ Recon $\wedge$ PlanPB $\neq$ Report1 $\wedge$ PlanPB $\neq$ CompletePlan $\wedge$
PlanPB $\neq$ Opoid $\wedge$ PlanPB $\neq$ Supervise $\wedge$ PlanPB $\neq$ Report2 $\wedge$
PlanPB $\neq$ Complete $\wedge$ PlanPB $\neq$ unAuthenticated $\wedge$
PlanPB $\neq$ unAuthorized $\wedge$ ReceiveMission $\neq$ Warno $\wedge$
ReceiveMission $\neq$ TentativePlan $\wedge$
ReceiveMission $\neq$ InitiateMovement $\wedge$ ReceiveMission $\neq$ Recon $\wedge$
ReceiveMission $\neq$ Report1 $\wedge$ ReceiveMission $\neq$ CompletePlan $\wedge$
ReceiveMission $\neq$ Opoid $\wedge$ ReceiveMission $\neq$ Supervise $\wedge$
ReceiveMission $\neq$ Report2 $\wedge$ ReceiveMission $\neq$ Complete $\wedge$
ReceiveMission $\neq$ unAuthenticated $\wedge$
ReceiveMission $\neq$ unAuthorized $\wedge$ Warno $\neq$ TentativePlan $\wedge$
Warno $\neq$ InitiateMovement $\wedge$ Warno $\neq$ Recon $\wedge$ Warno $\neq$ Report1 $\wedge$
Warno $\neq$ CompletePlan $\wedge$ Warno $\neq$ Opoid $\wedge$ Warno $\neq$ Supervise $\wedge$
Warno $\neq$ Report2 $\wedge$ Warno $\neq$ Complete $\wedge$
Warno $\neq$ unAuthenticated $\wedge$ Warno $\neq$ unAuthorized $\wedge$
TentativePlan $\neq$ InitiateMovement $\wedge$ TentativePlan $\neq$ Recon $\wedge$
TentativePlan $\neq$ Report1 $\wedge$ TentativePlan $\neq$ CompletePlan $\wedge$
TentativePlan $\neq$ Opoid $\wedge$ TentativePlan $\neq$ Supervise $\wedge$
TentativePlan $\neq$ Report2 $\wedge$ TentativePlan $\neq$ Complete $\wedge$
TentativePlan $\neq$ unAuthenticated $\wedge$
TentativePlan $\neq$ unAuthorized $\wedge$ InitiateMovement $\neq$ Recon $\wedge$
InitiateMovement $\neq$ Report1 $\wedge$
InitiateMovement $\neq$ CompletePlan $\wedge$ InitiateMovement $\neq$ Opoid $\wedge$
InitiateMovement $\neq$ Supervise $\wedge$ InitiateMovement $\neq$ Report2 $\wedge$
InitiateMovement $\neq$ Complete $\wedge$
InitiateMovement $\neq$ unAuthenticated $\wedge$
InitiateMovement $\neq$ unAuthorized $\wedge$ Recon $\neq$ Report1 $\wedge$
Recon $\neq$ CompletePlan $\wedge$ Recon $\neq$ Opoid $\wedge$ Recon $\neq$ Supervise $\wedge$
Recon $\neq$ Report2 $\wedge$ Recon $\neq$ Complete $\wedge$
Recon $\neq$ unAuthenticated $\wedge$ Recon $\neq$ unAuthorized $\wedge$
Report1 $\neq$ CompletePlan $\wedge$ Report1 $\neq$ Opoid $\wedge$
Report1 $\neq$ Supervise $\wedge$ Report1 $\neq$ Report2 $\wedge$
Report1 $\neq$ Complete $\wedge$ Report1 $\neq$ unAuthenticated $\wedge$
Report1 $\neq$ unAuthorized $\wedge$ CompletePlan $\neq$ Opoid $\wedge$
CompletePlan $\neq$ Supervise $\wedge$ CompletePlan $\neq$ Report2 $\wedge$
CompletePlan $\neq$ Complete $\wedge$ CompletePlan $\neq$ unAuthenticated $\wedge$
CompletePlan $\neq$ unAuthorized $\wedge$ Opoid $\neq$ Supervise $\wedge$
Opoid $\neq$ Report2 $\wedge$ Opoid $\neq$ Complete $\wedge$
Opoid $\neq$ unAuthenticated $\wedge$ Opoid $\neq$ unAuthorized $\wedge$
Supervise $\neq$ Report2 $\wedge$ Supervise $\neq$ Complete $\wedge$
Supervise $\neq$ unAuthenticated $\wedge$ Supervise $\neq$ unAuthorized $\wedge$
Report2 $\neq$ Complete $\wedge$ Report2 $\neq$ unAuthenticated $\wedge$

Report2 ≠ unAuthorized ∧ Complete ≠ unAuthenticated ∧
Complete ≠ unAuthorized ∧ unAuthenticated ≠ unAuthorized

[slRole_distinct_clauses]

⊢ PlatoonLeader ≠ PlatoonSergeant

[slState_distinct_clauses]

⊢ PLAN_PB ≠ RECEIVE_MISSION ∧ PLAN_PB ≠ WARNO ∧
PLAN_PB ≠ TENTATIVE_PLAN ∧ PLAN_PB ≠ INITIATE_MOVEMENT ∧
PLAN_PB ≠ RECON ∧ PLAN_PB ≠ REPORT1 ∧
PLAN_PB ≠ COMPLETE_PLAN ∧ PLAN_PB ≠ OPOID ∧
PLAN_PB ≠ SUPERVISE ∧ PLAN_PB ≠ REPORT2 ∧
PLAN_PB ≠ COMPLETE ∧ RECEIVE_MISSION ≠ WARNO ∧
RECEIVE_MISSION ≠ TENTATIVE_PLAN ∧
RECEIVE_MISSION ≠ INITIATE_MOVEMENT ∧
RECEIVE_MISSION ≠ RECON ∧ RECEIVE_MISSION ≠ REPORT1 ∧
RECEIVE_MISSION ≠ COMPLETE_PLAN ∧ RECEIVE_MISSION ≠ OPOID ∧
RECEIVE_MISSION ≠ SUPERVISE ∧ RECEIVE_MISSION ≠ REPORT2 ∧
RECEIVE_MISSION ≠ COMPLETE ∧ WARNO ≠ TENTATIVE_PLAN ∧
WARNO ≠ INITIATE_MOVEMENT ∧ WARNO ≠ RECON ∧ WARNO ≠ REPORT1 ∧
WARNO ≠ COMPLETE_PLAN ∧ WARNO ≠ OPOID ∧ WARNO ≠ SUPERVISE ∧
WARNO ≠ REPORT2 ∧ WARNO ≠ COMPLETE ∧
TENTATIVE_PLAN ≠ INITIATE_MOVEMENT ∧ TENTATIVE_PLAN ≠ RECON ∧
TENTATIVE_PLAN ≠ REPORT1 ∧ TENTATIVE_PLAN ≠ COMPLETE_PLAN ∧
TENTATIVE_PLAN ≠ OPOID ∧ TENTATIVE_PLAN ≠ SUPERVISE ∧
TENTATIVE_PLAN ≠ REPORT2 ∧ TENTATIVE_PLAN ≠ COMPLETE ∧
INITIATE_MOVEMENT ≠ RECON ∧ INITIATE_MOVEMENT ≠ REPORT1 ∧
INITIATE_MOVEMENT ≠ COMPLETE_PLAN ∧
INITIATE_MOVEMENT ≠ OPOID ∧ INITIATE_MOVEMENT ≠ SUPERVISE ∧
INITIATE_MOVEMENT ≠ REPORT2 ∧ INITIATE_MOVEMENT ≠ COMPLETE ∧
RECON ≠ REPORT1 ∧ RECON ≠ COMPLETE_PLAN ∧ RECON ≠ OPOID ∧
RECON ≠ SUPERVISE ∧ RECON ≠ REPORT2 ∧ RECON ≠ COMPLETE ∧
REPORT1 ≠ COMPLETE_PLAN ∧ REPORT1 ≠ OPOID ∧
REPORT1 ≠ SUPERVISE ∧ REPORT1 ≠ REPORT2 ∧
REPORT1 ≠ COMPLETE ∧ COMPLETE_PLAN ≠ OPOID ∧
COMPLETE_PLAN ≠ SUPERVISE ∧ COMPLETE_PLAN ≠ REPORT2 ∧
COMPLETE_PLAN ≠ COMPLETE ∧ OPOID ≠ SUPERVISE ∧
OPOID ≠ REPORT2 ∧ OPOID ≠ COMPLETE ∧ SUPERVISE ≠ REPORT2 ∧
SUPERVISE ≠ COMPLETE ∧ REPORT2 ≠ COMPLETE

Index

Appendix C

Secure State Machine Theories: HOL Script Files

C.1 ssm

```
(*****************************************************************************
(* Secure State Machine Theory: authentication, authorization, and state
*)
(* interpretation.
*)
(* Author: Shiu-Kai Chin
*)
(* Date: 27 November 2015
*)
(*****************************************************************************


structure ssmScript = struct

(* ==== Interactive mode ====
app load ["TypeBase", "ssminfRules","listTheory","optionTheory","acl_infRu
          "satListTheory","ssmTheory"];
open TypeBase listTheory ssminfRules optionTheory acl_infRules satListTheor

app load ["TypeBase", "ssminfRules","listTheory","optionTheory","acl_infRu
          "satListTheory"];
open TypeBase listTheory ssminfRules optionTheory acl_infRules satListTheor
     ssmTheory
 ==== end interactive mode ==== *)

open HolKernel boolLib Parse bossLib
open TypeBase listTheory optionTheory ssminfRules acl_infRules satListTheor
```

```
(**********************)
(* create a new theory *)
(**********************)
val _ = new_theory "ssm";

(* ---------------------------------------------------------------------
(* Define the type of transition: discard, execute, or trap. We discard fro
*)
(* the input stream those inputs that are not of the form P says command. W
*)
(* execute commands that users and supervisors are authorized for. We trap
*)
(* commands that users are not authorized to execute.
*)
(* ---------------------------------------------------------------------

(* ---------------------------------------------------------------------
(* In keeping with virtual machine design principles as described by Popek
*)
(* and Goldberg, we add a TRAP instruction to the commands by users.
*)
(* In effect, we are LIFTING the commands available to users to include the
*)
(* TRAP instruction used by the state machine to handle authorization error
(* ---------------------------------------------------------------------

val _ =
Datatype
`trType =
  discard 'cmdlist | trap 'cmdlist | exec 'cmdlist`

val trType_distinct_clauses = distinct_of``:'cmdlist trType``
val _ = save_thm("trType_distinct_clauses",trType_distinct_clauses)

val trType_one_one = one_one_of``:'cmdlist trType``
val _ = save_thm("trType_one_one",trType_one_one)

(* ---------------------------------------------------------------------
(* Define configuration to include the security context within which the
*)
(* inputs are evaluated. The components are as follows: (1) the authentica
(* function, (2) the intepretation of the state, (3) the security context,
*)
(* (4) the input stream, (5) the state, and (6) the output stream.
*)
(* ---------------------------------------------------------------------
val _ =
Datatype
```

```
‘configuration =
 CFG
   (('command option,'principal,'d,'e)Form -> bool)
   (('state -> ('command option,'principal,'d,'e)Form list ->
    ('command option,'principal,'d,'e)Form list))
   ((('command option,'principal,'d,'e)Form list) ->
    (('command option,'principal,'d,'e)Form list))
   ((((('command option,'principal,'d,'e)Form) list) list)
   ('state)
   ('output list)‘

(* ------------------------------------------------------------------------
(* Prove one-to-one properties of configuration
*)
(* ------------------------------------------------------------------------
val configuration_one_one =
    one_one_of‘‘:('command option,'d,'e,'output,'principal,'state)configura

val _ = save_thm("configuration_one_one",configuration_one_one)

(* ------------------------------------------------------------------------
(* The interpretation of configuration is the conjunction of the formulas
*)
(* the context and the first element of a non-empty input stream.
*)
(* ------------------------------------------------------------------------
val CFGInterpret_def =
Define
‘CFGInterpret
 ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
 (CFG
   (elementTest:('command option,'principal,'d,'e)Form -> bool)
   (stateInterp:'state -> (('command option,'principal,'d,'e)Form list) ->
    (('command option,'principal,'d,'e)Form list))
   (context:(('command option,'principal,'d,'e)Form list) ->
    (('command option,'principal,'d,'e)Form list))
   ((x:('command option,'principal,'d,'e)Form list)::ins)
   (state:'state)
   (outStream:'output list))
 =
   ((M,Oi,Os) satList (context x)) /\
   ((M,Oi,Os) satList x) /\
   ((M,Oi,Os) satList (stateInterp state x))‘

(*************************************************************************
(* In the following definitions of authenticationTest, extractCommand, and
*)
(* commandList, we implicitly assume that the only authenticated inputs are
```

```
*)
(* of the form P says phi, i.e., we know who is making statement phi.
*)
(*****************************************************************

val authenticationTest_def =
Define
`authenticationTest
 (elementTest:('command option,'principal,'d,'e)Form -> bool)
 (x:('command option,'principal,'d,'e)Form list) =
 FOLDR (\p q.p /\ q) T (MAP elementTest x)`;

val extractCommand_def =
Define
`extractCommand (P says (prop (SOME cmd)):('command option,'principal,'d,'e
   cmd`;

val commandList_def =
Define
`commandList (x:('command option,'principal,'d,'e)Form list) =
 MAP extractCommand x`;

val extractPropCommand_def =
Define
`(extractPropCommand (P says (prop (SOME cmd)):('command option,'principal,
   ((prop (SOME cmd)):('command option,'principal,'d,'e)Form))`;

val propCommandList_def =
Define
`propCommandList (x:('command option,'principal,'d,'e)Form list) =
 MAP extractPropCommand x`;

val extractInput_def =
Define
`extractInput (P says (prop x):('command option,'principal,'d,'e)Form) = x

val inputList_def =
Define
`inputList (xs:('command option,'principal,'d,'e)Form list) =
 MAP extractInput xs`;

(* ----------------------------------------------------------------------
(* Define transition relation among configurations. This definition is
*)
(* parameterized in terms of next-state transition function and output
*)
(* function.
*)
```

```
(* ------------------------------------------------------------------------
val (TR_rules, TR_ind, TR_cases) =
Hol_reln
`(!(elementTest:('command option,'principal,'d,'e)Form -> bool)
    (NS: 'state -> ('command option list) trType -> 'state) M Oi Os Out (s:
    (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
    (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
     ('command option,'principal,'d,'e)Form list)
    (x:('command option,'principal,'d,'e)Form list)
    (ins:('command option,'principal,'d,'e)Form list list)
    (outs:'output list).
 (authenticationTest elementTest x) /\
 (CFGInterpret (M,Oi,Os)
  (CFG elementTest stateInterp context (x::ins) s outs)) ==>
 (TR
  ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
  (exec (inputList x))
  (CFG elementTest stateInterp context (x::ins) s outs)
  (CFG elementTest stateInterp context ins
       (NS s (exec (inputList x)))
       ((Out s (exec (inputList x)))::outs)))) /\
 (!(elementTest:('command option,'principal,'d,'e)Form -> bool)
    (NS: 'state -> ('command option list) trType -> 'state) M Oi Os Out (s:
    (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
    (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
     ('command option,'principal,'d,'e)Form list)
    (x:('command option,'principal,'d,'e)Form list)
    (ins:('command option,'principal,'d,'e)Form list list)
    (outs:'output list).
 (authenticationTest elementTest x)  /\
 (CFGInterpret (M,Oi,Os)
  (CFG elementTest stateInterp context (x::ins) s outs)) ==>
 (TR
  ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
  (trap (inputList x))
 (CFG elementTest stateInterp context (x::ins) s outs)
 (CFG elementTest stateInterp context ins
      (NS s (trap (inputList x)))
      ((Out s (trap (inputList x)))::outs)))) /\
 (!(elementTest:('command option,'principal,'d,'e)Form -> bool)
    (NS: 'state -> ('command option list) trType -> 'state) M Oi Os Out (s:
    (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
    (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
     ('command option,'principal,'d,'e)Form list)
    (x:('command option,'principal,'d,'e)Form list)
```

```
    (ins:('command option,'principal,'d,'e)Form list list)
    (outs:'output list).
 ~(authenticationTest elementTest x) ==>
 (TR
  ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
  (discard (inputList x))
 (CFG elementTest stateInterp context (x::ins) s outs)
 (CFG elementTest stateInterp context ins
      (NS s (discard (inputList x)))
      ((Out s (discard (inputList x)))::outs))))`


(* ------------------------------------------------------------------------
(* Split up TR_rules into individual clauses
*)
(* ------------------------------------------------------------------------
val [rule0,rule1,rule2] = CONJUNCTS TR_rules

(*************************************************************************
(* Prove the converse of rule0, rule1, and rule2
*)
(*************************************************************************
val TR_lemma0 =
TAC_PROOF(([],flip_TR_rules rule0),
DISCH_TAC THEN
IMP_RES_TAC TR_cases THEN
PAT_ASSUM
 ``exec cmd = y``
 (fn th => ASSUME_TAC(REWRITE_RULE[trType_one_one,trType_distinct_clauses]t
PROVE_TAC[configuration_one_one,list_11,trType_distinct_clauses])


val TR_lemma1 =
TAC_PROOF(([],flip_TR_rules rule1),
DISCH_TAC THEN
IMP_RES_TAC TR_cases THEN
PAT_ASSUM
 ``trap cmd = y``
 (fn th => ASSUME_TAC(REWRITE_RULE[trType_one_one,trType_distinct_clauses]t
 PROVE_TAC[configuration_one_one,list_11,trType_distinct_clauses])

val TR_lemma2 =
TAC_PROOF(([],flip_TR_rules rule2),
DISCH_TAC THEN
IMP_RES_TAC TR_cases THEN
PAT_ASSUM
 ``discard (inputList x)= y``
 (fn th => ASSUME_TAC(REWRITE_RULE[trType_one_one,trType_distinct_clauses]t
```

```
PROVE_TAC[configuration_one_one, list_11, trType_distinct_clauses])

val TR_rules_converse =
TAC_PROOF(([], flip_TR_rules TR_rules),
REWRITE_TAC[TR_lemma0, TR_lemma1, TR_lemma2])

val TR_EQ_rules_thm = TR_EQ_rules TR_rules TR_rules_converse

val _ = save_thm("TR_EQ_rules_thm", TR_EQ_rules_thm)

val [TRrule0, TRrule1, TR_discard_cmd_rule] = CONJUNCTS TR_EQ_rules_thm

val _ = save_thm("TRrule0", TRrule0)
val _ = save_thm("TRrule1", TRrule1)
val _ = save_thm("TR_discard_cmd_rule", TR_discard_cmd_rule)

(* ----------------------------------------------------------------------
(* If (CFGInterpret
*)
(*      (M, Oi, Os)
*)
(*      (CFG elementTest stateInterpret certList
*)
(*           ((P says (prop (CMD cmd)))::ins) s outs) ==>
*)
(*     ((M, Oi, Os) sat (prop (CMD cmd))))
*)
(* is a valid inference rule, then executing cmd the exec(CMD cmd) transiti
(* occurs if and only if prop (CMD cmd), elementTest, and
*)
(* CFGInterpret (M, Oi, Os)
*)
(*  (CFG elementTest stateInterpret certList (P says prop (CMD cmd)::ins) s
*)
(* are true.
*)
(* ----------------------------------------------------------------------
val TR_exec_cmd_rule =
TAC_PROOF(([],
``!elementTest context stateInterp (x:('command option,'principal,'d,'e)For
   ins s outs.
 (!M Oi Os.
 (CFGInterpret
  ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po), (Os :
  (CFG elementTest
       (stateInterp:'state -> ('command option,'principal,'d,'e)Form list -
        ('command option,'principal,'d,'e)Form list) context
       (x::ins)
```

```
        (s:'state) (outs:'output list))) ==>
  (M,Oi,Os) satList (propCommandList (x:('command option, 'principal, 'd,
(!NS Out M Oi Os.
 TR
  ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po),
   (Os :'e po)) (exec (inputList x))
  (CFG (elementTest :('command option, 'principal, 'd, 'e) Form -> bool)
           (stateInterp:'state -> ('command option,'principal,'d,'e)Form li
            ('command option,'principal,'d,'e)Form list)
           (context :('command option, 'principal, 'd, 'e) Form list ->
            ('command option, 'principal, 'd, 'e) Form list)
           (x::ins)
           (s :'state) (outs :'output list))
   (CFG elementTest stateInterp context ins
           ((NS :'state -> 'command option list trType -> 'state) s (exec (
           (Out s (exec (inputList x))::outs)) <=>
  (authenticationTest elementTest x) /\
  (CFGInterpret (M,Oi,Os)
   (CFG elementTest stateInterp context (x::ins) s outs)) /\
   (M,Oi,Os) satList (propCommandList x))``),
REWRITE_TAC[TRrule0] THEN
REPEAT STRIP_TAC THEN
EQ_TAC THEN
REPEAT STRIP_TAC THEN
PROVE_TAC[])

val _ = save_thm("TR_exec_cmd_rule",TR_exec_cmd_rule)

(* ------------------------------------------------------------------------
(* If (CFGInterpret
*)
(*      (M,Oi,Os)
*)
(*      (CFG elementTest stateInterpret certList
*)
(*           ((P says (prop (CMD cmd)))::ins) s outs) ==>
*)
(*     ((M,Oi,Os) sat (prop TRAP)))
*)
(* is a valid inference rule, then executing cmd the trap(CMD cmd) transiti
(* occurs if and only if prop TRAP, elementTest, and
*)
(* CFGInterpret (M,Oi,Os)
*)
(*  (CFG elementTest stateInterpret certList (P says prop (CMD cmd)::ins)
*)
(*       s outs) are true.
*)
```

```
(* ------------------------------------------------------------------------
val TR_trap_cmd_rule =
TAC_PROOF(
([],``!elementTest context stateInterp (x:('command option,'principal,'d,'e
   ins s outs.
 (!M Oi Os.
 (CFGInterpret
  ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po), (Os :
  (CFG elementTest
       (stateInterp:'state -> ('command option,'principal,'d,'e)Form list -
        ('command option,'principal,'d,'e)Form list) context
       (x::ins)
       (s:'state) (outs:'output list))) ==>
  (M,Oi,Os) sat (prop NONE)) ==>
(!NS Out M Oi Os.
 TR
  ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po),
   (Os :'e po)) (trap (inputList x))
  (CFG (elementTest :('command option, 'principal, 'd, 'e) Form -> bool)
           (stateInterp:'state -> ('command option,'principal,'d,'e)Form li
            ('command option,'principal,'d,'e)Form list)
           (context :('command option, 'principal, 'd, 'e) Form list ->
            ('command option, 'principal, 'd, 'e) Form list)
           (x::ins)
           (s :'state) (outs :'output list))
   (CFG elementTest stateInterp context ins
           ((NS :'state -> 'command option list trType -> 'state) s (trap (
           (Out s (trap (inputList x))::outs)) <=>
  (authenticationTest elementTest x) /\
  (CFGInterpret (M,Oi,Os)
   (CFG elementTest stateInterp context (x::ins) s outs)) /\
   (M,Oi,Os) sat (prop NONE))``),
REWRITE_TAC[TRrule1] THEN
REPEAT STRIP_TAC THEN
EQ_TAC THEN
REPEAT STRIP_TAC THEN
PROVE_TAC[])

val _ = save_thm("TR_trap_cmd_rule",TR_trap_cmd_rule)

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

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

end (* structure *)
```

C.2 satList

```
(* ------------------------------------------------------------------------
(* Definition of satList for conjunctions of ACL formulas
*)
(* Author: Shiu-Kai Chin
*)
(* Date: 24 July 2014
*)
(* ------------------------------------------------------------------------
structure satListScript = struct

(*  interactive mode
 app load
  ["TypeBase","listTheory","acl_infRules"];
*)
open HolKernel boolLib Parse bossLib
open TypeBase acl_infRules listTheory

(***********
* create a new theory
***********)
val _ = new_theory "satList";

(*************************************************************)
(* Configurations and policies are represented by lists      *)
(* of formulas in the access-control logic.                  *)
(* Previously, for a formula f in the access-control logic,  *)
(* we ultimately interpreted it within the context of a      *)
(* Kripke structure M and partial orders Oi:'Int po and      *)
(* Os:'Sec po. This is represented as (M,Oi,Os) sat f.       *)
(* The natural extension is to interpret a list of formulas  *)
(* [f0;..;fn] as a conjunction:                              *)
(* (M,Oi,Os) sat f0 /\ ... /\ (M,Oi,Os) sat fn               *)
(*************************************************************)

val _ = set_fixity "satList" (Infixr 540);


val satList_def =
Define
`((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po))
 satList
 formList =
 FOLDR
 (\x y. x /\ y) T
 (MAP
  (\ (f:('prop,'pName,'Int,'Sec)Form).
```

```
   ((M:('prop,'world,'pName,'Int,'Sec)Kripke),
     Oi:'Int po,Os:'Sec po) sat f)formList)``;

(************************)
(* Properties of satList *)
(************************)
val satList_nil =
TAC_PROOF(
([],
``((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po)) sat
REWRITE_TAC[satList_def,FOLDR,MAP])

val _ = save_thm("satList_nil",satList_nil)

val satList_conj =
TAC_PROOF(
([],
``!l1 l2 M Oi Os.(((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(
    satList l1) /\
   (((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po))
    satList l2) =
   (((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po))
    satList (l1 ++ l2))``),
Induct THEN
REWRITE_TAC[APPEND,satList_nil] THEN
REWRITE_TAC[satList_def,MAP] THEN
CONV_TAC(DEPTH_CONV BETA_CONV) THEN
REWRITE_TAC[FOLDR] THEN
CONV_TAC(DEPTH_CONV BETA_CONV) THEN
REWRITE_TAC[GSYM satList_def] THEN
PROVE_TAC[])

val _ = save_thm("satList_conj",satList_conj)

val satList_CONS =
TAC_PROOF(([],
``!h t M Oi Os.(((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os
    satList (h::t)) =
   (((M,Oi,Os) sat h) /\
   (((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po))
    satList t))``),
REPEAT STRIP_TAC THEN
REWRITE_TAC[satList_def,MAP] THEN
CONV_TAC(DEPTH_CONV BETA_CONV) THEN
REWRITE_TAC[FOLDR] THEN
CONV_TAC(DEPTH_CONV BETA_CONV) THEN
REWRITE_TAC[])
```

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

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

end (* structure *)
```

Appendix D

Secure State Machine Theories Applied to Patrol Base Operations: HOL Script Files

D.1 OMNILevel

```
(*************************************************************************
(* OMNIScript
*)
(* Author: Lori Pickering
*)
(* Date: 10 May 2018
*)
(* This file is intended to allow for integration among the ssms.
The idea    *)
(* is to provide an OMNI-level integrating theory, in the sense of a super-
*)
(* conscious that knows when each ssm is complete and provides that info to
*)
(* higher-level state machines.
*)
(*************************************************************************

structure OMNIScript = struct

(* ==== Interactive Mode =====
app load ["TypeBase","listTheory", "optionTheory",
          "OMNITypeTheory",
```

```
        "acl_infRules","aclDrulesTheory","aclrulesTheory"];
open TypeBase listTheory optionTheory
     OMNITypeTheory
     acl_infRules aclDrulesTheory aclrulesTheory
 ====  End Interactive Mode ====  *)

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open OMNITypeTheory
open acl_infRules aclDrulesTheory aclrulesTheory

val _ = new_theory "OMNI";
(*****************************************************************************
(* Define slCommands for OMNI.
*)
(*****************************************************************************
(* ==== Area 52 ==========

val _=
Datatype `stateRole = Omni`

val _=
Datatype `omniCommand = ssmPlanPBComplete
                      | ssmMoveToORPComplete
                      | ssmConductORPComplete
                      | ssmMoveToPBComplete
                      | ssmConductPBComplete`

val omniCommand_distinct_clauses = distinct_of ``:omniCommand``
val _= save_thm("omniCommand_distinct_clauses",
                omniCommand_distinct_clauses)

val _=
Datatype `slCommand = OMNI omniCommand`


val omniAuthentication_def =
Define
`(omniAuthentication
        (Name Omni says prop (cmd:((slCommand command) option))
        :((slCommand command) option, stateRole, 'd,'e)Form) = T) /\
 (omniAuthentication _ = F)`

val omniAuthorization_def =
Define
`(omniAuthorization
        (Name Omni controls prop (cmd:((slCommand command) option))
        :((slCommand command) option, stateRole, 'd,'e)Form) = T) /\
```

```
 (omniAuthorization _ = F)‘


This may not be necessary...But, it is interesting.  Save for a later time.
(*************************************************************************
(* Prove that
*)
(*   Omni says omniCommand ==> omniCommand
*)
(*************************************************************************

set_goal([],
     ‘‘(Name Omni says prop (cmd:((slCommand command) option))
         :((slCommand command) option, stateRole, 'd, 'e)Form) ==>
                      prop (cmd:((slCommand command) option))‘‘)



val th1 = ASSUME‘‘(Name Omni says prop (cmd:((slCommand command) option))
         :((slCommand command) option, stateRole, 'd, 'e)Form) = TT‘‘
val th2 = REWRITE_RULE[omniAuthentication_def]th1

 ========= End Area 52 ==== *)

val _= export_theory();
end
```

D.2 TopLevel

D.2.1 PBTypeIntegrated Theory: Type Definitions

```
(*************************************************************************
(* PBTypeIntegrated
*)
(* Author: Lori Pickering
*)
(* Date 12 May 2018
*)
(* This theory contains the type definitions for ssmPBIntegrated
*)
(*************************************************************************
structure PBTypeIntegratedScript = struct
```

```
(* ===== Interactive Mode ====
app load ["TypeBase"]
open TypeBase
 ==== end Interactive Mode ==== *)


open HolKernel Parse boolLib bossLib;
open TypeBase OMNITypeTheory

val _= new_theory "PBTypeIntegrated";

(*****************************************************************************
(* Define types
*)
(*****************************************************************************
val _=
Datatype `plCommand = crossLD   (* Move to MOVE_TO_ORP state *)
                    | conductORP
                    | moveToPB
                    | conductPB
                    | completePB
                    | incomplete`

val plCommand_distinct_clauses = distinct_of ``:plCommand``
val _= save_thm("plCommand_distinct_clauses",
                plCommand_distinct_clauses)

val _=
Datatype `omniCommand = ssmPlanPBComplete
                    | ssmMoveToORPComplete
                    | ssmConductORPComplete
                    | ssmMoveToPBComplete
                    | ssmConductPBComplete
                    | invalidOmniCommand`

val omniCommand_distinct_clauses = distinct_of ``:omniCommand``
val _= save_thm("omniCommand_distinct_clauses",
                omniCommand_distinct_clauses)


val _=
Datatype `slCommand = PL plCommand
                    | OMNI omniCommand`


val slCommand_distinct_clauses = distinct_of ``:slCommand``
val _= save_thm("slCommand_distinct_clauses",
                slCommand_distinct_clauses)
```

```
val slCommand_one_one = one_one_of ``:slCommand``
val _= save_thm("slCommand_one_one", slCommand_one_one)



val _=
Datatype `stateRole = PlatoonLeader | Omni`

val stateRole_distinct_clauses = distinct_of ``:stateRole``
val _= save_thm("stateRole_distinct_clauses",
                stateRole_distinct_clauses)


val _=
Datatype `slState = PLAN_PB
                  | MOVE_TO_ORP
                  | CONDUCT_ORP
                  | MOVE_TO_PB
                  | CONDUCT_PB
                  | COMPLETE_PB`

val slState_distinct_clauses = distinct_of ``:slState``
val _ = save_thm("slState_distinct_clauses",slState_distinct_clauses)

val _=
Datatype `slOutput = PlanPB
                   | MoveToORP
                   | ConductORP
                   | MoveToPB
                   | ConductPB
                   | CompletePB
                   | unAuthenticated
                   | unAuthorized`

val slOutput_distinct_clauses = distinct_of ``:slOutput``
val _ = save_thm("slOutput_distinct_clauses",slOutput_distinct_clauses)


val _= export_theory();
end
```

D.2.2 PBIntegratedDef Theory: Authentication & Authorization Definitions

```
(*****************************************************************************
```

```
(* PBIntegratedDefTheory
*)
(* Author: Lori Pickering
*)
(* Date: 7 May 2018
*)
(* Definitions for ssmPBIntegratedTheory.
*)
(*****************************************************************************
structure PBIntegratedDefScript = struct

(* ===== Interactive Mode ====
app load  ["TypeBase", "listTheory","optionTheory",
           "uavUtilities",
          "OMNITypeTheory",
          "PBIntegratedDefTheory","PBTypeIntegratedTheory"];

open TypeBase listTheory optionTheory
     aclsemanticsTheory aclfoundationTheory
     uavUtilities
     OMNITypeTheory
     PBIntegratedDefTheory PBTypeIntegratedTheory
 ==== end Interactive Mode ==== *)

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open  uavUtilities
open OMNITypeTheory  PBTypeIntegratedTheory

val _ = new_theory "PBIntegratedDef";
(* ---------------------------------------------------------------------
(* state Interpretation function
*)
(* ---------------------------------------------------------------------
(* This function doesn't do anything but is necessary to specialize other
*)
(* theorems.
*)
(* ---------------------------------------------------------------------
val secContext_def = Define `
    secContext (x:((slCommand command)option, stateRole, 'd,'e)Form list) =
        [(TT:((slCommand command)option, stateRole, 'd,'e)Form)]`

val secHelper =
Define`
  (secHelper (cmd:omniCommand) =
     [(Name Omni) controls prop (SOME (SLc (OMNI (cmd:omniCommand))))])`
```

```
val getOmniCommand_def =
Define `
  (getOmniCommand ([]:((slCommand command)option, stateRole, 'd,'e)Form lis
                        = invalidOmniCommand:omniCommand) /\
  (getOmniCommand (((Name Omni) controls prop (SOME (SLc (OMNI cmd))))::xs)
                        = (cmd:omniCommand)) /\
  (getOmniCommand ((x:((slCommand command)option, stateRole, 'd,'e)Form)::x
                        =  (getOmniCommand xs))`

val secAuthorization_def =
Define `
  (secAuthorization (xs:((slCommand command)option, stateRole,'d,'e)Form li
                    = secHelper (getOmniCommand xs)) `


val secContext_def =
Define `
 (secContext (PLAN_PB) ((x:((slCommand command)option, stateRole, 'd,'e)For
        [(prop (SOME (SLc (OMNI (ssmPlanPBComplete))))
         :((slCommand command)option, stateRole, 'd,'e)Form) impf
         (Name PlatoonLeader) controls prop (SOME (SLc (PL crossLD)))
          :((slCommand command)option, stateRole, 'd,'e)Form])
/\
 (secContext (MOVE_TO_ORP) ((x:((slCommand command)option, stateRole, 'd,'e
        [prop (SOME (SLc  (OMNI (ssmMoveToORPComplete)))) impf
         (Name PlatoonLeader) controls prop (SOME (SLc (PL conductORP)))])
 (secContext (CONDUCT_ORP) ((x:((slCommand command)option, stateRole, 'd,'e
        [prop (SOME (SLc (OMNI (ssmConductORPComplete)))) impf
         (Name PlatoonLeader) controls prop (SOME (SLc (PL moveToPB)))]) /\
 (secContext (MOVE_TO_PB) ((x:((slCommand command)option, stateRole, 'd,'e)
        [prop (SOME (SLc (OMNI (ssmMoveToPBComplete)))) impf
         (Name PlatoonLeader) controls prop (SOME (SLc (PL conductPB)))]) /
 (secContext (CONDUCT_PB) ((x:((slCommand command)option, stateRole, 'd,'e)
        [prop (SOME (SLc (OMNI (ssmConductPBComplete)))) impf
         (Name PlatoonLeader) controls prop (SOME (SLc (PL completePB)))])`

(* ===== Area 52 ====

 ==== End Area 52 ==== *)

val _= export_theory();
end
```

D.2.3 ssmPlanPBIntegrated Theory: Theorems

```
(*****************************************************************************
(* ssmPBIntegratedTheory
```

```
*)
(* Author: Lori Pickering
*)
(* Date: 7 May 2018
*)
(* This theory aims to integrate the topLevel ssm with the sublevel ssms.
It *)
(* does this by adding a condition to the security context.  In particular,
*)
(* it requires that the "COMPLETE" state in the subLevel ssm must preceede
*)
(* transition to the next state at the topLeve.  I.e.,
*)
(*   planPBComplete ==>
*)
(*   PlatoonLeader controls crossLD.
*)
(* In the ssmPlanPB ssm, the last state is COMPLETE.  This is reached when
(* the appropriate authority says complete and the transition is made.
*)
(* Note that following the ACL, if P says x and P controls x, then x.
*)
(* Therefore, it is not necessary for anyone to say x at the topLevel, bec
(* it is already proved at the lower level.
*)
(* However, indicating that at the topLevel remains something to workout.
*)
(*************************************************************************

structure ssmPBIntegratedScript = struct

(* ===== Interactive Mode ====
app load  ["TypeBase", "listTheory","optionTheory","listSyntax",
          "acl_infRules","aclDrulesTheory","aclrulesTheory",
          "aclsemanticsTheory", "aclfoundationTheory",
          "satListTheory","ssmTheory","ssminfRules","uavUtilities",
          "OMNITypeTheory", "PBTypeIntegratedTheory","PBIntegratedDefTheory
          "ssmPBIntegratedTheory"];

open TypeBase listTheory optionTheory listSyntax
     acl_infRules aclDrulesTheory aclrulesTheory
     aclsemanticsTheory aclfoundationTheory
     satListTheory ssmTheory ssminfRules uavUtilities
     OMNITypeTheory PBTypeIntegratedTheory PBIntegratedDefTheory
     ssmPBIntegratedTheory
 ==== end Interactive Mode ==== *)
```

```
open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open acl_infRules aclDrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfRules uavUtilities
open OMNITypeTheory PBTypeIntegratedTheory PBIntegratedDefTheory


val _ = new_theory "ssmPBIntegrated";


(*****************************************************************************
(* Define next-state and next-output functions
*)
(*****************************************************************************
val PBNS_def =
Define `
(PBNS PLAN_PB      (exec [SOME (SLc (PL crossLD))])     = MOVE_TO_ORP) /\
(PBNS MOVE_TO_ORP (exec [SOME (SLc (PL conductORP))]) = CONDUCT_ORP) /\
(PBNS CONDUCT_ORP (exec [SOME (SLc (PL moveToPB))])    = MOVE_TO_PB)
/\
(PBNS MOVE_TO_PB  (exec [SOME (SLc (PL conductPB))]) =  CONDUCT_PB)
/\
(PBNS CONDUCT_PB  (exec [SOME (SLc (PL completePB))]) = COMPLETE_PB) /\
(PBNS (s:slState) (trap _)     = s) /\
(PBNS (s:slState) (discard _) = s)`

val PBOut_def =
Define `
(PBOut PLAN_PB      (exec [SOME (SLc (PL crossLD))])     = MoveToORP) /\
(PBOut MOVE_TO_ORP (exec [SOME (SLc (PL conductORP))]) = ConductORP) /\
(PBOut CONDUCT_ORP (exec [SOME (SLc (PL moveToPB))])    = MoveToPB)
/\
(PBOut MOVE_TO_PB  (exec [SOME (SLc (PL conductPB))]) =  ConductPB)
/\
(PBOut CONDUCT_PB  (exec [SOME (SLc (PL completePB))]) = CompletePB) /\
(PBOut (s:slState) (trap _)     = unAuthorized) /\
(PBOut (s:slState) (discard _) = unAuthenticated)`

(*****************************************************************************
(* Define authentication function
*)
(*****************************************************************************
val inputOK_def =
Define `
(inputOK (((Name PlatoonLeader) says prop (cmd:((slCommand command)option))
          :((slCommand command)option, stateRole,'d,'e)Form) = T) /\
(inputOK (((Name Omni)          says prop (cmd:((slCommand command)option))
          :((slCommand command)option, stateRole,'d,'e)Form) = T) /\
```

```
(inputOK _ = F)‘


(*************************************************************************
(* Prove that commands are rejected unless that are requested by a properly
*)
(* authenticated principal.
*)
(*************************************************************************

val inputOK_cmd_reject_lemma =
Q.prove(‘!cmd. ~(inputOK
                  ((prop (SOME cmd))))‘,
                                (PROVE_TAC[inputOK_def]))



(* ===== Just playing around with this ====
val inputOK_not_reject_lemma =
Q.prove(‘!cmd.
          ~(
            (inputOK (((Name PlatoonLeader) says prop (cmd:((slCommand comma
             :((slCommand command)option, stateRole,'d,'e)Form)) \/
            (inputOK (((Name Omni)            says prop (cmd:((slCommand comma
             :((slCommand command)option, stateRole,'d,'e)Form)))

 ==== OK, done fooling around ==== *)


val _ = export_theory();

end
```

D.3 Horizontal Slice

D.3.1 ssmPlanPB

D.3.1.1 PlanPBType Theory: Type Definitions

D.3.1.2 PlanPBDef Theory: Authentication & Authorization Definitions

D.3.1.3 ssmPlanPB Theory: Theorems

D.3.2 ssmMoveToORP

D.3.2.1 MoveToORPType Theory: Type Definitions

D.3.2.2 MoveToORPDef Theory: Authentication & Authorization Definitions

D.3.2.3 ssmMoveToORP Theory: Theorems

D.3.3 ssmConductORP

D.3.3.1 ConductORPType Theory: Type Definitions

D.3.3.2 ConductORPDef Theory: Authentication & Authorization Definitions

D.3.3.3 ssmConductORP Theory: Theorems

D.3.4 ssmMoveToPB

D.3.4.1 MoveToPBType Theory: Type Definitions

D.3.4.2 MoveToPBDef Theory: Authentication & Authorization Definitions

D.3.4.3 ssmMoveToPB Theory: Theorems

D.3.5 ssmConductPB

Appendix E

Map of The File Folder Structure

References

[1] Shiu-Kai Chin and Susan Beth Older. *Access Control, Security, and Trust: A Logical Approach*. Chapman & Hall: CRC Cryptography and Network Security Series. Chapman and Hall/CRC, July 2010.