

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

Abstract	i
List of Figures	vi
List of Tables	vii
List of Acronyms	viii
1 Introduction	1
2 Background	2
3 Systems Security Engineering	3
3.1 NIST Special Publication 800-160	3
3.2 Verification & Documentation	3
3.3 Principle of Complete Mediation	3
3.3.1 Formal Verification Using Computer-Aided Reasoning	3
4 Patrol Base Operations	1
4.1 Motivation	3
4.2 Ranger Handbook Description	3
4.3 Describing The Patrol Base Operations	3
4.4 Hierarchy of Secure State Machines	3
4.4.1 OMNI-Level	3
4.4.2 Escape	3
4.4.3 Top Level	3
4.4.4 Horizontal Slice	3
4.4.4.1 ssmPlanPB	3
4.4.4.2 ssmMoveToORP	3
4.4.4.3 ssmConductORP	3
4.4.4.4 ssmMoveToPB	3
4.4.4.5 ssmConductPB	3
4.4.5 Vertical Slice	3
4.4.5.1 ssmSecureHalt	3
4.4.5.2 ssmORPRecon	3
4.4.5.3 ssmMoveToORP4L	3
4.4.5.4 ssmFormRT	3

5	Secure State Machine Model	4
5.1	State Machines	5
5.1.1	Next-state Function	5
5.1.2	Next-output Function	5
5.1.3	Transition Commands	5
5.2	Secure State Machines	5
5.2.1	State Machine Versus Secure State Machine	5
5.2.2	Transition Types	5
5.2.3	Authentication	5
5.2.4	Authorization	5
5.3	Secure State Machines in HOL	5
5.3.1	Parameterizable Secure State Machine	5
5.3.2	Parameterization	5
5.3.3	Configurations: five parts	5
5.3.3.1	State Interpretation	5
5.3.3.2	Security context	5
5.3.3.3	Input stream	5
5.3.3.4	State	5
5.3.3.5	Output stream	5
5.3.4	Authentication	5
5.3.5	Configuration Interpretation	5
5.3.6	Transition Definitions	5
6	Patrol Base Operations as Secure State Machines	6
6.1	ssmPB: An Example from the Hierarchy	7
6.1.1	Principals	7
6.1.2	States	7
6.1.3	Commands	7
6.1.4	Next-State Function	7
6.1.5	Next-Output Function	7
6.1.6	Authentication	7
6.1.7	Authorization	7
6.1.8	Proved Theorems	7
6.1.8.1	Platoon Leader Is Trusted on plCommands	7
6.2	Other Variations	7
6.2.1	ssmPlanPB: Non-sequential Transitions	7
6.2.2	ssmConductORP: Principals Authorized for Subsets of Commands	7
7	Discussion	1
7.1	Recap	1
7.2	Mission Accomplished	1
7.3	Stop-Gaps, Lessons Learned, & Advice	1
7.4	Other Verifiable Theories	1
7.4.1	Platoon Theory, Soldier Theory, Squad Theory, etc.	1
7.4.2	Soldiers in Roles	1

8	Future Work & Implications	2
8.1	The Devil Is in The Details	2
8.2	Accountability Systems	6
8.3	Applicability	7
	Appendices	9
A	Access Control Logic Theories: Pretty-Printed Theories	10
B	Secure State Machine Theories: HOL Script Files	85
B.1	ssm	85
B.2	satList	94
C	Secure State Machine Theories Applied to Patrol Base Operations: HOL Script Files	97
C.1	OMNILEvel	97
C.2	TopLevel	99
C.2.1	PBTypeIntegrated Theory: Type Definitions	99
C.2.2	PBIntegratedDef Theory: Authentication & Authorization Definitions	101
C.2.3	ssmPlanPBIntegrated Theory: Theorems	103
C.3	Horizontal Slice	108
C.3.1	ssmPlanPB	108
C.3.1.1	PlanPBType Theory: Type Definitions	108
C.3.1.2	PlanPBDef Theory: Authentication & Authorization Definitions	108
C.3.1.3	ssmPlanPB Theory: Theorems	108
C.3.2	ssmMoveToORP	108
C.3.2.1	MoveToORPType Theory: Type Definitions	108
C.3.2.2	MoveToORPDef Theory: Authentication & Authorization Definitions	108
C.3.2.3	ssmMoveToORP Theory: Theorems	108
C.3.3	ssmConductORP	108
C.3.3.1	ConductORPType Theory: Type Definitions	108
C.3.3.2	ConductORPDef Theory: Authentication & Authorization Definitions	108
C.3.3.3	ssmConductORP Theory: Theorems	108
C.3.4	ssmMoveToPB	108
C.3.4.1	MoveToPBType Theory: Type Definitions	108
C.3.4.2	MoveToPBDef Theory: Authentication & Authorization Definitions	108
C.3.4.3	ssmMoveToPB Theory: Theorems	108
C.3.5	ssmConductPB	108
C.3.5.1	ConductPBType Theory: Type Definitions	108
C.3.5.2	ConductPBDef Theory: Authentication & Authorization Definitions	108
C.3.5.3	ssmConductPB Theory: Theorems	108
C.4	Vertical Slice	108
C.4.1	ssmSecureHalt	108

C.4.1.1	SecureHaltType Theory: Type Definitions	108
C.4.1.2	SecureHaltDef Theory: Authentication & Authorization Definitions	108
C.4.1.3	ssmSecureHalt Theory: Theorems	108
C.4.2	ssmORPRecon	108
C.4.2.1	ORPReconType Theory: Type Definitions	108
C.4.2.2	ORPReconDef Theory: Authentication & Authorization Definitions	108
C.4.2.3	ssmORPRecon Theory: Theorems	108
C.4.3	ssmMoveToORP4L	108
C.4.3.1	MoveToORP4LType Theory: Type Definitions	108
C.4.3.2	MoveToORP4LDef Theory: Authentication & Authoriza- tion Definitions	108
C.4.3.3	ssmMoveToORP4L Theory: Theorems	108
C.4.4	ssmFormRT	108
C.4.4.1	FormRTType Theory: Type Definitions	108
C.4.4.2	FormRTDef Theory: Authentication & Authorization Def- initions	108
C.4.4.3	ssmFormRT Theory: Theorems	108
D	Patrol Base Operations: Pretty-Printed Theories	109
E	Map of The File Folder Structure	110
	References	111

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 L^AT_EX. 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.

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 L^AT_EX. 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 way 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	OMNITYPE Theory	3
1.1	Datatypes	3
1.2	Theorems	3
2	ssm11 Theory	4
2.1	Datatypes	4
2.2	Definitions	4
2.3	Theorems	5
3	ssm Theory	11
3.1	Datatypes	11
3.2	Definitions	12
3.3	Theorems	13
4	satList Theory	21
4.1	Definitions	21
4.2	Theorems	21
5	ssmPB Theory	21
5.1	Definitions	21
5.2	Theorems	22
6	PBTypeIntegrated Theory	26
6.1	Datatypes	26
6.2	Theorems	27
7	PBIntegratedDef Theory	28
7.1	Definitions	28
7.2	Theorems	28
8	ssmConductORP Theory	33
8.1	Definitions	33
8.2	Theorems	33
9	ConductORPType Theory	38
9.1	Datatypes	38
9.2	Theorems	39
10	ssmConductPB Theory	39
10.1	Definitions	40
10.2	Theorems	40

11 ConductPBType Theory	45
11.1 Datatypes	45
11.2 Theorems	45
12 ssmMoveToORP Theory	46
12.1 Definitions	46
12.2 Theorems	46
13 MoveToORPType Theory	51
13.1 Datatypes	51
13.2 Theorems	51
14 ssmMoveToPB Theory	52
14.1 Definitions	52
14.2 Theorems	52
15 MoveToPBType Theory	56
15.1 Datatypes	56
15.2 Theorems	56
16 ssmPlanPB Theory	57
16.1 Theorems	57
17 PlanPBType Theory	67
17.1 Datatypes	67
17.2 Theorems	68

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]

$\vdash \forall a' a. \text{ESCc } a \neq \text{SLc } a'$

[command_one_one]

$\vdash (\forall a a'. (\text{ESCc } a = \text{ESCc } a') \iff (a = a')) \wedge$
 $\quad \forall a a'. (\text{SLc } a = \text{SLc } a') \iff (a = a')$

[escCommand_distinct_clauses]

$\vdash \text{returnToBase} \neq \text{changeMission} \wedge \text{returnToBase} \neq \text{resupply} \wedge$
 $\quad \text{returnToBase} \neq \text{reactToContact} \wedge \text{changeMission} \neq \text{resupply} \wedge$
 $\quad \text{changeMission} \neq \text{reactToContact} \wedge \text{resupply} \neq \text{reactToContact}$

[escOutput_distinct_clauses]

$\vdash \text{ReturnToBase} \neq \text{ChangeMission} \wedge \text{ReturnToBase} \neq \text{Resupply} \wedge$
 $\quad \text{ReturnToBase} \neq \text{ReactToContact} \wedge \text{ChangeMission} \neq \text{Resupply} \wedge$
 $\quad \text{ChangeMission} \neq \text{ReactToContact} \wedge \text{Resupply} \neq \text{ReactToContact}$

[escState_distinct_clauses]

$\vdash \text{RTB} \neq \text{CM} \wedge \text{RTB} \neq \text{RESUPPLY} \wedge \text{RTB} \neq \text{RTC} \wedge \text{CM} \neq \text{RESUPPLY} \wedge$
 $\quad \text{CM} \neq \text{RTC} \wedge \text{RESUPPLY} \neq \text{RTC}$

[output_distinct_clauses]

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

[output_one_one]

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

[principal_one_one]

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

[state_distinct_clauses]

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

[state_one_one]

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

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]

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

```

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) ⇔

```

$$(M, Oi, Os) \text{ satList } securityContext \wedge (M, Oi, Os) \text{ sat } input \wedge \\ (M, Oi, Os) \text{ sat } stateInterp \text{ state}$$

[CFGInterpret_ind]

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

[configuration_one_one]

$$\vdash \forall a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a'_0 \ a'_1 \ a'_2 \ a'_3 \ a'_4 \ a'_5. \\ (CFG \ a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 = CFG \ a'_0 \ a'_1 \ a'_2 \ a'_3 \ a'_4 \ a'_5) \iff \\ (a_0 = a'_0) \wedge (a_1 = a'_1) \wedge (a_2 = a'_2) \wedge (a_3 = a'_3) \wedge \\ (a_4 = a'_4) \wedge (a_5 = a'_5)$$

[order_distinct_clauses]

$$\vdash \forall a. \text{ SOME } a \neq \text{ NONE}$$

[order_one_one]

$$\vdash \forall a \ a'. (\text{SOME } a = \text{SOME } a') \iff (a = a')$$

[TR_cases]

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

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

[TR_discard_cmd_rule]

$$\begin{aligned}
& \vdash \text{TR } (M, Oi, Os) \text{ (discard cmd)} \\
& \quad (\text{CFG authenticationTest stateInterp securityContext} \\
& \quad \quad (x::ins) s \text{ outs}) \\
& \quad (\text{CFG authenticationTest stateInterp securityContext ins} \\
& \quad \quad \quad (\text{NS } s \text{ (discard cmd)}) (\text{Out } s \text{ (discard cmd)::outs})) \iff \\
& \neg \text{authenticationTest } x
\end{aligned}$$

[TR_EQ_rules_thm]

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

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

[TR_exec_cmd_rule]

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

[TR_ind]

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

$$\begin{aligned}
& \neg \text{authenticationTest } x \Rightarrow \\
& \text{TR}' (M, Oi, Os) (\text{discard } cmd) \\
& \quad (\text{CFG authenticationTest stateInterp securityContext} \\
& \quad \quad (x :: ins) s outs) \\
& \quad (\text{CFG authenticationTest stateInterp securityContext} \\
& \quad \quad ins (NS s (\text{discard } cmd))) \\
& \quad (\text{Out } s (\text{discard } cmd) :: outs))) \Rightarrow \\
& \forall a_0 a_1 a_2 a_3. \text{TR } a_0 a_1 a_2 a_3 \Rightarrow \text{TR}' a_0 a_1 a_2 a_3
\end{aligned}$$

[TR_rules]

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

[TR_strongind]

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

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

[TR_trap_cmd_rule]

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

[TRrule0]

$$\begin{aligned}
& \vdash \text{TR (M, Oi, Os) (exec cmd)} \\
& \quad (CFG \text{ authenticationTest stateInterp securityContext}
\end{aligned}$$

```

(P says prop (SOME cmd)::ins) s outs)
(CFG authenticationTest stateInterp securityContext ins
 (NS s (exec cmd)) (Out s (exec cmd)::outs))  $\iff$ 
authenticationTest (P says prop (SOME cmd))  $\wedge$ 
CFGInterpret (M, Oi, Os)
 (CFG authenticationTest stateInterp securityContext
  (P says prop (SOME cmd)::ins) s outs)

```

[TRrule1]

```

 $\vdash$  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))  $\iff$ 
authenticationTest (P says prop (SOME cmd))  $\wedge$ 
CFGInterpret (M, Oi, Os)
  (CFG authenticationTest stateInterp securityContext
   (P says prop (SOME cmd)::ins) s outs)

```

[trType_distinct_clauses]

```

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

```

[trType_one_one]

```

 $\vdash (\forall a a'. (\text{discard } a = \text{discard } a') \iff (a = a')) \wedge$ 
 $(\forall a a'. (\text{trap } a = \text{trap } a') \iff (a = a')) \wedge$ 
 $\forall a a'. (\text{exec } a = \text{exec } a') \iff (a = a')$ 

```

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]

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

[commandList_def]

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

[inputList_def]

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

[propCommandList_def]

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

[TR_def]

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

$$\begin{aligned}
& \text{outs})) \vee \\
& (\exists \text{elementTest } NS \ M \ Oi \ Os \ Out \ s \ context \ stateInterp \ x \\
& \quad \text{ins } outs. \\
& \quad (a_0 = (M, Oi, Os)) \wedge (a_1 = \text{discard } (\text{inputList } x)) \wedge \\
& \quad (a_2 = \\
& \quad \quad \text{CFG } \text{elementTest } \text{stateInterp } \text{context } (x::\text{ins}) \ s \\
& \quad \quad \text{outs}) \wedge \\
& \quad (a_3 = \\
& \quad \quad \text{CFG } \text{elementTest } \text{stateInterp } \text{context } \text{ins} \\
& \quad \quad (\text{NS } s \ (\text{discard } (\text{inputList } x))) \\
& \quad \quad (\text{Out } s \ (\text{discard } (\text{inputList } x))::\text{outs})) \wedge \\
& \quad \neg \text{authenticationTest } \text{elementTest } x) \Rightarrow \\
& TR' \ a_0 \ a_1 \ a_2 \ a_3) \Rightarrow \\
& TR' \ a_0 \ a_1 \ a_2 \ a_3)
\end{aligned}$$

3.3 Theorems

[CFGInterpret_def]

$$\begin{aligned}
& \vdash \text{CFGInterpret } (M, Oi, Os) \\
& \quad (\text{CFG } \text{elementTest } \text{stateInterp } \text{context } (x::\text{ins}) \ \text{state} \\
& \quad \quad \text{outStream}) \iff \\
& \quad (M, Oi, Os) \ \text{satList } \text{context } x \wedge (M, Oi, Os) \ \text{satList } x \wedge \\
& \quad (M, Oi, Os) \ \text{satList } \text{stateInterp } \text{state } x
\end{aligned}$$

[CFGInterpret_ind]

$$\begin{aligned}
& \vdash \forall P. \\
& \quad (\forall M \ Oi \ Os \ \text{elementTest } \text{stateInterp } \text{context } x \ \text{ins } \text{state} \\
& \quad \quad \text{outStream}. \\
& \quad \quad P \ (M, Oi, Os) \\
& \quad \quad (\text{CFG } \text{elementTest } \text{stateInterp } \text{context } (x::\text{ins}) \ \text{state} \\
& \quad \quad \quad \text{outStream})) \wedge \\
& \quad (\forall v_{15} \ v_{10} \ v_{11} \ v_{12} \ v_{13} \ v_{14}. \\
& \quad \quad P \ v_{15} \ (\text{CFG } v_{10} \ v_{11} \ v_{12} \ [] \ v_{13} \ v_{14})) \Rightarrow \\
& \quad \forall v \ v_1 \ v_2 \ v_3. \ P \ (v, v_1, v_2) \ v_3
\end{aligned}$$

[configuration_one_one]

$$\begin{aligned}
& \vdash \forall a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a'_0 \ a'_1 \ a'_2 \ a'_3 \ a'_4 \ a'_5. \\
& \quad (\text{CFG } a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 = \text{CFG } a'_0 \ a'_1 \ a'_2 \ a'_3 \ a'_4 \ a'_5) \iff \\
& \quad (a_0 = a'_0) \wedge (a_1 = a'_1) \wedge (a_2 = a'_2) \wedge (a_3 = a'_3) \wedge \\
& \quad (a_4 = a'_4) \wedge (a_5 = a'_5)
\end{aligned}$$

[extractCommand_def]

$$\vdash \text{extractCommand } (P \ \text{says prop } (\text{SOME } \text{cmd})) = \text{cmd}$$

[extractCommand_ind]

$$\begin{aligned}
& \vdash \forall P'. \\
& \quad (\forall P \ \text{cmd}. \ P' \ (P \ \text{says prop } (\text{SOME } \text{cmd}))) \wedge P' \ \text{TT} \wedge P' \ \text{FF} \wedge \\
& \quad (\forall v_1. \ P' \ (\text{prop } v_1)) \wedge (\forall v_3. \ P' \ (\text{notf } v_3)) \wedge
\end{aligned}$$

$$\begin{aligned}
& (\forall v_6 v_7. P' (v_6 \text{ andf } v_7)) \wedge (\forall v_{10} v_{11}. P' (v_{10} \text{ orf } v_{11})) \wedge \\
& (\forall v_{14} v_{15}. P' (v_{14} \text{ impf } v_{15})) \wedge \\
& (\forall v_{18} v_{19}. P' (v_{18} \text{ eqf } v_{19})) \wedge (\forall v_{129}. P' (v_{129} \text{ says TT})) \wedge \\
& (\forall v_{130}. P' (v_{130} \text{ says FF})) \wedge \\
& (\forall v_{132}. P' (v_{132} \text{ says prop NONE})) \wedge \\
& (\forall v_{133} v_{66}. P' (v_{133} \text{ says notf } v_{66})) \wedge \\
& (\forall v_{134} v_{69} v_{70}. P' (v_{134} \text{ says } (v_{69} \text{ andf } v_{70}))) \wedge \\
& (\forall v_{135} v_{73} v_{74}. P' (v_{135} \text{ says } (v_{73} \text{ orf } v_{74}))) \wedge \\
& (\forall v_{136} v_{77} v_{78}. P' (v_{136} \text{ says } (v_{77} \text{ impf } v_{78}))) \wedge \\
& (\forall v_{137} v_{81} v_{82}. P' (v_{137} \text{ says } (v_{81} \text{ eqf } v_{82}))) \wedge \\
& (\forall v_{138} v_{85} v_{86}. P' (v_{138} \text{ says } v_{85} \text{ says } v_{86})) \wedge \\
& (\forall v_{139} v_{89} v_{90}. P' (v_{139} \text{ says } v_{89} \text{ speaks_for } v_{90})) \wedge \\
& (\forall v_{140} v_{93} v_{94}. P' (v_{140} \text{ says } v_{93} \text{ controls } v_{94})) \wedge \\
& (\forall v_{141} v_{98} v_{99} v_{100}. P' (v_{141} \text{ says reps } v_{98} v_{99} v_{100})) \wedge \\
& (\forall v_{142} v_{103} v_{104}. P' (v_{142} \text{ says } v_{103} \text{ domi } v_{104})) \wedge \\
& (\forall v_{143} v_{107} v_{108}. P' (v_{143} \text{ says } v_{107} \text{ eqi } v_{108})) \wedge \\
& (\forall v_{144} v_{111} v_{112}. P' (v_{144} \text{ says } v_{111} \text{ doms } v_{112})) \wedge \\
& (\forall v_{145} v_{115} v_{116}. P' (v_{145} \text{ says } v_{115} \text{ eqs } v_{116})) \wedge \\
& (\forall v_{146} v_{119} v_{120}. P' (v_{146} \text{ says } v_{119} \text{ eqn } v_{120})) \wedge \\
& (\forall v_{147} v_{123} v_{124}. P' (v_{147} \text{ says } v_{123} \text{ lte } v_{124})) \wedge \\
& (\forall v_{148} v_{127} v_{128}. P' (v_{148} \text{ says } v_{127} \text{ lt } v_{128})) \wedge \\
& (\forall v_{24} v_{25}. P' (v_{24} \text{ speaks_for } v_{25})) \wedge \\
& (\forall v_{28} v_{29}. P' (v_{28} \text{ controls } v_{29})) \wedge \\
& (\forall v_{33} v_{34} v_{35}. P' (\text{reps } v_{33} v_{34} v_{35})) \wedge \\
& (\forall v_{38} v_{39}. P' (v_{38} \text{ domi } v_{39})) \wedge \\
& (\forall v_{42} v_{43}. P' (v_{42} \text{ eqi } v_{43})) \wedge \\
& (\forall v_{46} v_{47}. P' (v_{46} \text{ doms } v_{47})) \wedge \\
& (\forall v_{50} v_{51}. P' (v_{50} \text{ eqs } v_{51})) \wedge \\
& (\forall v_{54} v_{55}. P' (v_{54} \text{ eqn } v_{55})) \wedge \\
& (\forall v_{58} v_{59}. P' (v_{58} \text{ lte } v_{59})) \wedge \\
& (\forall v_{62} v_{63}. P' (v_{62} \text{ lt } v_{63})) \Rightarrow \\
& \forall v. P' v
\end{aligned}$$

[extractInput_def]

$\vdash \text{extractInput } (P \text{ says prop } x) = x$

[extractInput_ind]

$\vdash \forall P'.$

$$\begin{aligned}
& (\forall P x. P' (P \text{ says prop } x)) \wedge P' \text{ TT} \wedge P' \text{ FF} \wedge \\
& (\forall v_1. P' (\text{prop } v_1)) \wedge (\forall v_3. P' (\text{notf } v_3)) \wedge \\
& (\forall v_6 v_7. P' (v_6 \text{ andf } v_7)) \wedge (\forall v_{10} v_{11}. P' (v_{10} \text{ orf } v_{11})) \wedge \\
& (\forall v_{14} v_{15}. P' (v_{14} \text{ impf } v_{15})) \wedge \\
& (\forall v_{18} v_{19}. P' (v_{18} \text{ eqf } v_{19})) \wedge (\forall v_{129}. P' (v_{129} \text{ says TT})) \wedge \\
& (\forall v_{130}. P' (v_{130} \text{ says FF})) \wedge \\
& (\forall v_{131} v_{66}. P' (v_{131} \text{ says notf } v_{66})) \wedge \\
& (\forall v_{132} v_{69} v_{70}. P' (v_{132} \text{ says } (v_{69} \text{ andf } v_{70}))) \wedge \\
& (\forall v_{133} v_{73} v_{74}. P' (v_{133} \text{ says } (v_{73} \text{ orf } v_{74}))) \wedge \\
& (\forall v_{134} v_{77} v_{78}. P' (v_{134} \text{ says } (v_{77} \text{ impf } v_{78}))) \wedge \\
& (\forall v_{135} v_{81} v_{82}. P' (v_{135} \text{ says } (v_{81} \text{ eqf } v_{82}))) \wedge
\end{aligned}$$

$$\begin{aligned}
& (\forall v136 \ v85 \ v86. \ P' \ (v136 \ \text{says} \ v85 \ \text{says} \ v86)) \wedge \\
& (\forall v137 \ v89 \ v90. \ P' \ (v137 \ \text{says} \ v89 \ \text{speaks_for} \ v90)) \wedge \\
& (\forall v138 \ v93 \ v94. \ P' \ (v138 \ \text{says} \ v93 \ \text{controls} \ v94)) \wedge \\
& (\forall v139 \ v98 \ v99 \ v100. \ P' \ (v139 \ \text{says} \ \text{reps} \ v98 \ v99 \ v100)) \wedge \\
& (\forall v140 \ v103 \ v104. \ P' \ (v140 \ \text{says} \ v103 \ \text{domi} \ v104)) \wedge \\
& (\forall v141 \ v107 \ v108. \ P' \ (v141 \ \text{says} \ v107 \ \text{eqi} \ v108)) \wedge \\
& (\forall v142 \ v111 \ v112. \ P' \ (v142 \ \text{says} \ v111 \ \text{doms} \ v112)) \wedge \\
& (\forall v143 \ v115 \ v116. \ P' \ (v143 \ \text{says} \ v115 \ \text{eqs} \ v116)) \wedge \\
& (\forall v144 \ v119 \ v120. \ P' \ (v144 \ \text{says} \ v119 \ \text{eqn} \ v120)) \wedge \\
& (\forall v145 \ v123 \ v124. \ P' \ (v145 \ \text{says} \ v123 \ \text{lte} \ v124)) \wedge \\
& (\forall v146 \ v127 \ v128. \ P' \ (v146 \ \text{says} \ v127 \ \text{lt} \ v128)) \wedge \\
& (\forall v24 \ v25. \ P' \ (v24 \ \text{speaks_for} \ v25)) \wedge \\
& (\forall v28 \ v29. \ P' \ (v28 \ \text{controls} \ v29)) \wedge \\
& (\forall v33 \ v34 \ v35. \ P' \ (\text{reps} \ v33 \ v34 \ v35)) \wedge \\
& (\forall v38 \ v39. \ P' \ (v38 \ \text{domi} \ v39)) \wedge \\
& (\forall v42 \ v43. \ P' \ (v42 \ \text{eqi} \ v43)) \wedge \\
& (\forall v46 \ v47. \ P' \ (v46 \ \text{doms} \ v47)) \wedge \\
& (\forall v50 \ v51. \ P' \ (v50 \ \text{eqs} \ v51)) \wedge \\
& (\forall v54 \ v55. \ P' \ (v54 \ \text{eqn} \ v55)) \wedge \\
& (\forall v58 \ v59. \ P' \ (v58 \ \text{lte} \ v59)) \wedge \\
& (\forall v62 \ v63. \ P' \ (v62 \ \text{lt} \ v63)) \Rightarrow \\
& \forall v. \ P' \ v
\end{aligned}$$

[extractPropCommand_def]

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

[extractPropCommand_ind]

$$\begin{aligned}
& \vdash \forall P'. \\
& \quad (\forall P \ \text{cmd}. \ P' \ (P \ \text{says} \ \text{prop} \ (\text{SOME} \ \text{cmd}))) \wedge P' \ \text{TT} \wedge P' \ \text{FF} \wedge \\
& \quad (\forall v_1. \ P' \ (\text{prop} \ v_1)) \wedge (\forall v_3. \ P' \ (\text{notf} \ v_3)) \wedge \\
& \quad (\forall v_6 \ v_7. \ P' \ (v_6 \ \text{andf} \ v_7)) \wedge (\forall v_{10} \ v_{11}. \ P' \ (v_{10} \ \text{orf} \ v_{11})) \wedge \\
& \quad (\forall v_{14} \ v_{15}. \ P' \ (v_{14} \ \text{impf} \ v_{15})) \wedge \\
& \quad (\forall v_{18} \ v_{19}. \ P' \ (v_{18} \ \text{eqf} \ v_{19})) \wedge (\forall v_{129}. \ P' \ (v_{129} \ \text{says} \ \text{TT})) \wedge \\
& \quad (\forall v_{130}. \ P' \ (v_{130} \ \text{says} \ \text{FF})) \wedge \\
& \quad (\forall v_{132}. \ P' \ (v_{132} \ \text{says} \ \text{prop} \ \text{NONE})) \wedge \\
& \quad (\forall v_{133} \ v_{66}. \ P' \ (v_{133} \ \text{says} \ \text{notf} \ v_{66})) \wedge \\
& \quad (\forall v_{134} \ v_{69} \ v_{70}. \ P' \ (v_{134} \ \text{says} \ (v_{69} \ \text{andf} \ v_{70}))) \wedge \\
& \quad (\forall v_{135} \ v_{73} \ v_{74}. \ P' \ (v_{135} \ \text{says} \ (v_{73} \ \text{orf} \ v_{74}))) \wedge \\
& \quad (\forall v_{136} \ v_{77} \ v_{78}. \ P' \ (v_{136} \ \text{says} \ (v_{77} \ \text{impf} \ v_{78}))) \wedge \\
& \quad (\forall v_{137} \ v_{81} \ v_{82}. \ P' \ (v_{137} \ \text{says} \ (v_{81} \ \text{eqf} \ v_{82}))) \wedge \\
& \quad (\forall v_{138} \ v_{85} \ v_{86}. \ P' \ (v_{138} \ \text{says} \ v_{85} \ \text{says} \ v_{86})) \wedge \\
& \quad (\forall v_{139} \ v_{89} \ v_{90}. \ P' \ (v_{139} \ \text{says} \ v_{89} \ \text{speaks_for} \ v_{90})) \wedge \\
& \quad (\forall v_{140} \ v_{93} \ v_{94}. \ P' \ (v_{140} \ \text{says} \ v_{93} \ \text{controls} \ v_{94})) \wedge \\
& \quad (\forall v_{141} \ v_{98} \ v_{99} \ v_{100}. \ P' \ (v_{141} \ \text{says} \ \text{reps} \ v_{98} \ v_{99} \ v_{100})) \wedge \\
& \quad (\forall v_{142} \ v_{103} \ v_{104}. \ P' \ (v_{142} \ \text{says} \ v_{103} \ \text{domi} \ v_{104})) \wedge \\
& \quad (\forall v_{143} \ v_{107} \ v_{108}. \ P' \ (v_{143} \ \text{says} \ v_{107} \ \text{eqi} \ v_{108})) \wedge \\
& \quad (\forall v_{144} \ v_{111} \ v_{112}. \ P' \ (v_{144} \ \text{says} \ v_{111} \ \text{doms} \ v_{112})) \wedge \\
& \quad (\forall v_{145} \ v_{115} \ v_{116}. \ P' \ (v_{145} \ \text{says} \ v_{115} \ \text{eqs} \ v_{116})) \wedge \\
& \quad (\forall v_{146} \ v_{119} \ v_{120}. \ P' \ (v_{146} \ \text{says} \ v_{119} \ \text{eqn} \ v_{120})) \wedge
\end{aligned}$$

$$\begin{aligned}
& (\forall v_{147} v_{123} v_{124}. P' (v_{147} \text{ says } v_{123} \text{ lte } v_{124})) \wedge \\
& (\forall v_{148} v_{127} v_{128}. P' (v_{148} \text{ says } v_{127} \text{ lt } v_{128})) \wedge \\
& (\forall v_{24} v_{25}. P' (v_{24} \text{ speaks_for } v_{25})) \wedge \\
& (\forall v_{28} v_{29}. P' (v_{28} \text{ controls } v_{29})) \wedge \\
& (\forall v_{33} v_{34} v_{35}. P' (\text{reps } v_{33} v_{34} v_{35})) \wedge \\
& (\forall v_{38} v_{39}. P' (v_{38} \text{ domi } v_{39})) \wedge \\
& (\forall v_{42} v_{43}. P' (v_{42} \text{ eqi } v_{43})) \wedge \\
& (\forall v_{46} v_{47}. P' (v_{46} \text{ doms } v_{47})) \wedge \\
& (\forall v_{50} v_{51}. P' (v_{50} \text{ eqs } v_{51})) \wedge \\
& (\forall v_{54} v_{55}. P' (v_{54} \text{ eqn } v_{55})) \wedge \\
& (\forall v_{58} v_{59}. P' (v_{58} \text{ lte } v_{59})) \wedge \\
& (\forall v_{62} v_{63}. P' (v_{62} \text{ lt } v_{63})) \Rightarrow \\
& \forall v. P' v
\end{aligned}$$

[TR_cases]

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

CFG elementTest stateInterp context ins
 (NS s (discard (inputList x)))
 (Out s (discard (inputList x))::outs)) \wedge
 \neg authenticationTest elementTest x

[TR_discard_cmd_rule]

\vdash 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)) \iff
 \neg authenticationTest elementTest x

[TR_EQ_rules_thm]

\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)) \wedge
 (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
 (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)) \iff
 \neg authenticationTest elementTest x)

[TR_exec_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) satList propCommandList x) \Rightarrow
 \forall 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

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

[TR_ind]

$\vdash \forall TR'.$

$$\begin{aligned}
& (\forall \text{elementTest} \ NS \ M \ Oi \ Os \ Out \ s \ \text{context} \ \text{stateInterp} \ x \ \text{ins} \\
& \quad \text{outs}. \\
& \text{authenticationTest} \ \text{elementTest} \ x \wedge \\
& \text{CFGInterpret} \ (M, Oi, Os) \\
& \quad (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \\
& \quad \text{outs}) \Rightarrow \\
& TR' \ (M, Oi, Os) \ (\text{exec} \ (\text{inputList} \ x)) \\
& \quad (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \\
& \quad (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ \text{ins} \\
& \quad \quad (NS \ s \ (\text{exec} \ (\text{inputList} \ x))) \\
& \quad \quad (\text{Out} \ s \ (\text{exec} \ (\text{inputList} \ x))::\text{outs}))) \wedge \\
& (\forall \text{elementTest} \ NS \ M \ Oi \ Os \ Out \ s \ \text{context} \ \text{stateInterp} \ x \ \text{ins} \\
& \quad \text{outs}. \\
& \text{authenticationTest} \ \text{elementTest} \ x \wedge \\
& \text{CFGInterpret} \ (M, Oi, Os) \\
& \quad (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \\
& \quad \text{outs}) \Rightarrow \\
& TR' \ (M, Oi, Os) \ (\text{trap} \ (\text{inputList} \ x)) \\
& \quad (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \\
& \quad (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ \text{ins} \\
& \quad \quad (NS \ s \ (\text{trap} \ (\text{inputList} \ x))) \\
& \quad \quad (\text{Out} \ s \ (\text{trap} \ (\text{inputList} \ x))::\text{outs}))) \wedge \\
& (\forall \text{elementTest} \ NS \ M \ Oi \ Os \ Out \ s \ \text{context} \ \text{stateInterp} \ x \ \text{ins} \\
& \quad \text{outs}. \\
& \neg \text{authenticationTest} \ \text{elementTest} \ x \Rightarrow \\
& TR' \ (M, Oi, Os) \ (\text{discard} \ (\text{inputList} \ x)) \\
& \quad (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ (x::\text{ins}) \ s \ \text{outs}) \\
& \quad (\text{CFG} \ \text{elementTest} \ \text{stateInterp} \ \text{context} \ \text{ins} \\
& \quad \quad (NS \ s \ (\text{discard} \ (\text{inputList} \ x))) \\
& \quad \quad (\text{Out} \ s \ (\text{discard} \ (\text{inputList} \ x))::\text{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
\end{aligned}$$

[TR_rules]

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

```

(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))

```


$$\begin{aligned}
& (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ s outs}) \\
& (\text{CFG elementTest stateInterp context ins} \\
& \quad (\text{NS s (discard (inputList x))}) \\
& \quad (\text{Out s (discard (inputList x))::outs})) \Rightarrow \\
& \forall a_0 \ a_1 \ a_2 \ a_3. \text{TR } a_0 \ a_1 \ a_2 \ a_3 \Rightarrow \text{TR}' a_0 \ a_1 \ a_2 \ a_3
\end{aligned}$$

[TR_trap_cmd_rule]

$$\begin{aligned}
& \vdash \forall \text{elementTest context stateInterp } x \text{ ins s outs.} \\
& \quad (\forall M \ Oi \ Os. \\
& \quad \quad \text{CFGInterpret } (M, Oi, Os) \\
& \quad \quad (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ s} \\
& \quad \quad \quad \text{outs}) \Rightarrow \\
& \quad \quad (M, Oi, Os) \text{ sat prop NONE}) \Rightarrow \\
& \quad \forall \text{NS Out } M \ Oi \ Os. \\
& \quad \text{TR } (M, Oi, Os) (\text{trap (inputList x)}) \\
& \quad (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ s outs}) \\
& \quad (\text{CFG elementTest stateInterp context ins} \\
& \quad \quad (\text{NS s (trap (inputList x))}) \\
& \quad \quad (\text{Out s (trap (inputList x))::outs})) \iff \\
& \quad \text{authenticationTest elementTest } x \wedge \\
& \quad \text{CFGInterpret } (M, Oi, Os) \\
& \quad \quad (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ s outs}) \wedge \\
& \quad \quad (M, Oi, Os) \text{ sat prop NONE}
\end{aligned}$$

[TRrule0]

$$\begin{aligned}
& \vdash \text{TR } (M, Oi, Os) (\text{exec (inputList x)}) \\
& \quad (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ s outs}) \\
& \quad (\text{CFG elementTest stateInterp context ins} \\
& \quad \quad (\text{NS s (exec (inputList x))}) \\
& \quad \quad (\text{Out s (exec (inputList x))::outs})) \iff \\
& \quad \text{authenticationTest elementTest } x \wedge \\
& \quad \text{CFGInterpret } (M, Oi, Os) \\
& \quad \quad (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ s outs})
\end{aligned}$$

[TRrule1]

$$\begin{aligned}
& \vdash \text{TR } (M, Oi, Os) (\text{trap (inputList x)}) \\
& \quad (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ s outs}) \\
& \quad (\text{CFG elementTest stateInterp context ins} \\
& \quad \quad (\text{NS s (trap (inputList x))}) \\
& \quad \quad (\text{Out s (trap (inputList x))::outs})) \iff \\
& \quad \text{authenticationTest elementTest } x \wedge \\
& \quad \text{CFGInterpret } (M, Oi, Os) \\
& \quad \quad (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ s outs})
\end{aligned}$$

[trType_distinct_clauses]

$$\begin{aligned}
& \vdash (\forall a' \ a. \text{discard } a \neq \text{trap } a') \wedge (\forall a' \ a. \text{discard } a \neq \text{exec } a') \wedge \\
& \quad \forall a' \ a. \text{trap } a \neq \text{exec } a'
\end{aligned}$$

[trType_one_one]

$$\vdash (\forall a \ a'. (\text{discard } a = \text{discard } a') \iff (a = a')) \wedge$$

$$(\forall a \ a'. (\text{trap } a = \text{trap } a') \iff (a = a')) \wedge$$

$$\forall a \ a'. (\text{exec } a = \text{exec } a') \iff (a = a')$$

4 satList Theory

Built: 13 May 2018

Parent Theories: aclRules

4.1 Definitions

[satList_def]

$$\vdash \forall M \ Oi \ Os \ formList.$$

$$(M, Oi, Os) \text{ satList } formList \iff$$

$$\text{FOLDR } (\lambda x \ y. x \wedge y) \ T \ (\text{MAP } (\lambda f. (M, Oi, Os) \text{ sat } f) \ formList)$$

4.2 Theorems

[satList_conj]

$$\vdash \forall l_1 \ l_2 \ M \ Oi \ Os.$$

$$(M, Oi, Os) \text{ satList } l_1 \wedge (M, Oi, Os) \text{ satList } l_2 \iff$$

$$(M, Oi, Os) \text{ satList } (l_1 ++ l_2)$$

[satList_CONS]

$$\vdash \forall h \ t \ M \ Oi \ Os.$$

$$(M, Oi, Os) \text{ satList } (h :: t) \iff$$

$$(M, Oi, Os) \text{ sat } h \wedge (M, Oi, Os) \text{ satList } t$$

[satList_nil]

$$\vdash (M, Oi, Os) \text{ satList } []$$

5 ssmPB Theory

Built: 13 May 2018

Parent Theories: PBType, ssm11, OMNIType

5.1 Definitions

[secContext_def]

$$\vdash \forall cmd.$$

$$\text{secContext } cmd =$$

$$[\text{Name PlatoonLeader controls prop (SOME (SLc } cmd))]$$

[ssmPBStateInterp_def]

$$\vdash \forall state. \text{ssmPBStateInterp } state = \text{TT}$$

5.2 Theorems

[authenticationTest_cmd_reject_lemma]

$\vdash \forall cmd. \neg \text{authenticationTest} (\text{prop} (\text{SOME } cmd))$

[authenticationTest_def]

$\vdash (\text{authenticationTest} (\text{Name PlatoonLeader says prop } cmd) \iff$
 $\quad T) \wedge (\text{authenticationTest TT} \iff F) \wedge$
 $(\text{authenticationTest FF} \iff F) \wedge$
 $(\text{authenticationTest} (\text{prop } v) \iff F) \wedge$
 $(\text{authenticationTest} (\text{notf } v_1) \iff F) \wedge$
 $(\text{authenticationTest} (v_2 \text{ andf } v_3) \iff F) \wedge$
 $(\text{authenticationTest} (v_4 \text{ orf } v_5) \iff F) \wedge$
 $(\text{authenticationTest} (v_6 \text{ impf } v_7) \iff F) \wedge$
 $(\text{authenticationTest} (v_8 \text{ eqf } v_9) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says TT}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says FF}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says notf } v_{67}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } (v_{68} \text{ andf } v_{69})) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } (v_{70} \text{ orf } v_{71})) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } (v_{72} \text{ impf } v_{73})) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75})) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{76} \text{ says } v_{77}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{80} \text{ controls } v_{81}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says reps } v_{82} \ v_{83} \ v_{84}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{85} \text{ domi } v_{86}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{87} \text{ eqi } v_{88}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{89} \text{ doms } v_{90}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{91} \text{ eqs } v_{92}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{93} \text{ eqn } v_{94}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{95} \text{ lte } v_{96}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{10} \text{ says } v_{97} \text{ lt } v_{98}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{12} \text{ speaks_for } v_{13}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{14} \text{ controls } v_{15}) \iff F) \wedge$
 $(\text{authenticationTest} (\text{reps } v_{16} \ v_{17} \ v_{18}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{19} \text{ domi } v_{20}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{21} \text{ eqi } v_{22}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{23} \text{ doms } v_{24}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{25} \text{ eqs } v_{26}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{27} \text{ eqn } v_{28}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{29} \text{ lte } v_{30}) \iff F) \wedge$
 $(\text{authenticationTest} (v_{31} \text{ lt } v_{32}) \iff F)$

[authenticationTest_ind]

$\vdash \forall P.$
 $(\forall cmd. P (\text{Name PlatoonLeader says prop } cmd)) \wedge P \text{ TT} \wedge$

$$\begin{aligned}
& P \text{ FF} \wedge (\forall v. P (\text{prop } v)) \wedge (\forall v_1. P (\text{notf } v_1)) \wedge \\
& (\forall v_2 v_3. P (v_2 \text{ andf } v_3)) \wedge (\forall v_4 v_5. P (v_4 \text{ orf } v_5)) \wedge \\
& (\forall v_6 v_7. P (v_6 \text{ impf } v_7)) \wedge (\forall v_8 v_9. P (v_8 \text{ eqf } v_9)) \wedge \\
& (\forall v_{10}. P (v_{10} \text{ says TT})) \wedge (\forall v_{10}. P (v_{10} \text{ says FF})) \wedge \\
& (\forall v_{133} v_{134} v_{66}. P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66})) \wedge \\
& (\forall v_{135} v_{136} v_{66}. P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66})) \wedge \\
& (\forall v_{10} v_{67}. P (v_{10} \text{ says notf } v_{67})) \wedge \\
& (\forall v_{10} v_{68} v_{69}. P (v_{10} \text{ says } (v_{68} \text{ andf } v_{69}))) \wedge \\
& (\forall v_{10} v_{70} v_{71}. P (v_{10} \text{ says } (v_{70} \text{ orf } v_{71}))) \wedge \\
& (\forall v_{10} v_{72} v_{73}. P (v_{10} \text{ says } (v_{72} \text{ impf } v_{73}))) \wedge \\
& (\forall v_{10} v_{74} v_{75}. P (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75}))) \wedge \\
& (\forall v_{10} v_{76} v_{77}. P (v_{10} \text{ says } v_{76} \text{ says } v_{77})) \wedge \\
& (\forall v_{10} v_{78} v_{79}. P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79})) \wedge \\
& (\forall v_{10} v_{80} v_{81}. P (v_{10} \text{ says } v_{80} \text{ controls } v_{81})) \wedge \\
& (\forall v_{10} v_{82} v_{83} v_{84}. P (v_{10} \text{ says reps } v_{82} v_{83} v_{84})) \wedge \\
& (\forall v_{10} v_{85} v_{86}. P (v_{10} \text{ says } v_{85} \text{ domi } v_{86})) \wedge \\
& (\forall v_{10} v_{87} v_{88}. P (v_{10} \text{ says } v_{87} \text{ eqi } v_{88})) \wedge \\
& (\forall v_{10} v_{89} v_{90}. P (v_{10} \text{ says } v_{89} \text{ doms } v_{90})) \wedge \\
& (\forall v_{10} v_{91} v_{92}. P (v_{10} \text{ says } v_{91} \text{ eqs } v_{92})) \wedge \\
& (\forall v_{10} v_{93} v_{94}. P (v_{10} \text{ says } v_{93} \text{ eqn } v_{94})) \wedge \\
& (\forall v_{10} v_{95} v_{96}. P (v_{10} \text{ says } v_{95} \text{ lte } v_{96})) \wedge \\
& (\forall v_{10} v_{97} v_{98}. P (v_{10} \text{ says } v_{97} \text{ lt } v_{98})) \wedge \\
& (\forall v_{12} v_{13}. P (v_{12} \text{ speaks_for } v_{13})) \wedge \\
& (\forall v_{14} v_{15}. P (v_{14} \text{ controls } v_{15})) \wedge \\
& (\forall v_{16} v_{17} v_{18}. P (\text{reps } v_{16} v_{17} v_{18})) \wedge \\
& (\forall v_{19} v_{20}. P (v_{19} \text{ domi } v_{20})) \wedge \\
& (\forall v_{21} v_{22}. P (v_{21} \text{ eqi } v_{22})) \wedge \\
& (\forall v_{23} v_{24}. P (v_{23} \text{ doms } v_{24})) \wedge \\
& (\forall v_{25} v_{26}. P (v_{25} \text{ eqs } v_{26})) \wedge (\forall v_{27} v_{28}. P (v_{27} \text{ eqn } v_{28})) \wedge \\
& (\forall v_{29} v_{30}. P (v_{29} \text{ lte } v_{30})) \wedge (\forall v_{31} v_{32}. P (v_{31} \text{ lt } v_{32})) \Rightarrow \\
& \forall v. P v
\end{aligned}$$

[PBNS_def]

$$\begin{aligned}
& \vdash (\text{PBNS PLAN_PB (exec (SLc crossLD))} = \text{MOVE_TO_ORP}) \wedge \\
& (\text{PBNS PLAN_PB (exec (SLc incomplete))} = \text{PLAN_PB}) \wedge \\
& (\text{PBNS MOVE_TO_ORP (exec (SLc conductorORP))} = \text{CONDUCT_ORP}) \wedge \\
& (\text{PBNS MOVE_TO_ORP (exec (SLc incomplete))} = \text{MOVE_TO_ORP}) \wedge \\
& (\text{PBNS CONDUCT_ORP (exec (SLc moveToPB))} = \text{MOVE_TO_PB}) \wedge \\
& (\text{PBNS CONDUCT_ORP (exec (SLc incomplete))} = \text{CONDUCT_ORP}) \wedge \\
& (\text{PBNS MOVE_TO_PB (exec (SLc conductPB))} = \text{CONDUCT_PB}) \wedge \\
& (\text{PBNS MOVE_TO_PB (exec (SLc incomplete))} = \text{MOVE_TO_PB}) \wedge \\
& (\text{PBNS CONDUCT_PB (exec (SLc completePB))} = \text{COMPLETE_PB}) \wedge \\
& (\text{PBNS CONDUCT_PB (exec (SLc incomplete))} = \text{CONDUCT_PB}) \wedge \\
& (\text{PBNS } s \text{ (trap (SLc cmd))} = s) \wedge \\
& (\text{PBNS } s \text{ (discard (SLc cmd))} = s)
\end{aligned}$$

[PBNS_ind]

$$\begin{aligned}
& \vdash \forall P. \\
& P \text{ PLAN_PB (exec (SLc crossLD))} \wedge
\end{aligned}$$

$$\begin{aligned}
& P \text{ PLAN_PB } (\text{exec } (\text{SLc incomplete})) \wedge \\
& P \text{ MOVE_TO_ORP } (\text{exec } (\text{SLc conductORP})) \wedge \\
& P \text{ MOVE_TO_ORP } (\text{exec } (\text{SLc incomplete})) \wedge \\
& P \text{ CONDUCT_ORP } (\text{exec } (\text{SLc moveToPB})) \wedge \\
& P \text{ CONDUCT_ORP } (\text{exec } (\text{SLc incomplete})) \wedge \\
& P \text{ MOVE_TO_PB } (\text{exec } (\text{SLc conductPB})) \wedge \\
& P \text{ MOVE_TO_PB } (\text{exec } (\text{SLc incomplete})) \wedge \\
& P \text{ CONDUCT_PB } (\text{exec } (\text{SLc completePB})) \wedge \\
& P \text{ CONDUCT_PB } (\text{exec } (\text{SLc incomplete})) \wedge \\
& (\forall s \text{ cmd. } P s (\text{trap } (\text{SLc cmd}))) \wedge \\
& (\forall s \text{ cmd. } P s (\text{discard } (\text{SLc cmd}))) \wedge \\
& (\forall s v_6. P s (\text{discard } (\text{ESCc } v_6))) \wedge \\
& (\forall s v_9. P s (\text{trap } (\text{ESCc } v_9))) \wedge \\
& (\forall v_{12}. P \text{ PLAN_PB } (\text{exec } (\text{ESCc } v_{12}))) \wedge \\
& P \text{ PLAN_PB } (\text{exec } (\text{SLc conductORP})) \wedge \\
& P \text{ PLAN_PB } (\text{exec } (\text{SLc moveToPB})) \wedge \\
& P \text{ PLAN_PB } (\text{exec } (\text{SLc conductPB})) \wedge \\
& P \text{ PLAN_PB } (\text{exec } (\text{SLc completePB})) \wedge \\
& (\forall v_{15}. P \text{ MOVE_TO_ORP } (\text{exec } (\text{ESCc } v_{15}))) \wedge \\
& P \text{ MOVE_TO_ORP } (\text{exec } (\text{SLc crossLD})) \wedge \\
& P \text{ MOVE_TO_ORP } (\text{exec } (\text{SLc moveToPB})) \wedge \\
& P \text{ MOVE_TO_ORP } (\text{exec } (\text{SLc conductPB})) \wedge \\
& P \text{ MOVE_TO_ORP } (\text{exec } (\text{SLc completePB})) \wedge \\
& (\forall v_{18}. P \text{ CONDUCT_ORP } (\text{exec } (\text{ESCc } v_{18}))) \wedge \\
& P \text{ CONDUCT_ORP } (\text{exec } (\text{SLc crossLD})) \wedge \\
& P \text{ CONDUCT_ORP } (\text{exec } (\text{SLc conductORP})) \wedge \\
& P \text{ CONDUCT_ORP } (\text{exec } (\text{SLc conductPB})) \wedge \\
& P \text{ CONDUCT_ORP } (\text{exec } (\text{SLc completePB})) \wedge \\
& (\forall v_{21}. P \text{ MOVE_TO_PB } (\text{exec } (\text{ESCc } v_{21}))) \wedge \\
& P \text{ MOVE_TO_PB } (\text{exec } (\text{SLc crossLD})) \wedge \\
& P \text{ MOVE_TO_PB } (\text{exec } (\text{SLc conductORP})) \wedge \\
& P \text{ MOVE_TO_PB } (\text{exec } (\text{SLc moveToPB})) \wedge \\
& P \text{ MOVE_TO_PB } (\text{exec } (\text{SLc completePB})) \wedge \\
& (\forall v_{24}. P \text{ CONDUCT_PB } (\text{exec } (\text{ESCc } v_{24}))) \wedge \\
& P \text{ CONDUCT_PB } (\text{exec } (\text{SLc crossLD})) \wedge \\
& P \text{ CONDUCT_PB } (\text{exec } (\text{SLc conductORP})) \wedge \\
& P \text{ CONDUCT_PB } (\text{exec } (\text{SLc moveToPB})) \wedge \\
& P \text{ CONDUCT_PB } (\text{exec } (\text{SLc conductPB})) \wedge \\
& (\forall v_{26}. P \text{ COMPLETE_PB } (\text{exec } v_{26})) \Rightarrow \\
& \forall v v_1. P v v_1
\end{aligned}$$

[PBOut_def]

$$\begin{aligned}
& \vdash (\text{PBOut PLAN_PB } (\text{exec } (\text{SLc crossLD})) = \text{MoveToORP}) \wedge \\
& (\text{PBOut PLAN_PB } (\text{exec } (\text{SLc incomplete})) = \text{PlanPB}) \wedge \\
& (\text{PBOut MOVE_TO_ORP } (\text{exec } (\text{SLc conductORP})) = \text{ConductORP}) \wedge \\
& (\text{PBOut MOVE_TO_ORP } (\text{exec } (\text{SLc incomplete})) = \text{MoveToORP}) \wedge \\
& (\text{PBOut CONDUCT_ORP } (\text{exec } (\text{SLc moveToPB})) = \text{MoveToPB}) \wedge \\
& (\text{PBOut CONDUCT_ORP } (\text{exec } (\text{SLc incomplete})) = \text{ConductORP}) \wedge \\
& (\text{PBOut MOVE_TO_PB } (\text{exec } (\text{SLc conductPB})) = \text{ConductPB}) \wedge
\end{aligned}$$

$(\text{PBOut MOVE_TO_PB (exec (SLc incomplete))} = \text{MoveToPB}) \wedge$
 $(\text{PBOut CONDUCT_PB (exec (SLc completePB))} = \text{CompletePB}) \wedge$
 $(\text{PBOut CONDUCT_PB (exec (SLc incomplete))} = \text{ConductPB}) \wedge$
 $(\text{PBOut } s \text{ (trap (SLc cmd))} = \text{unAuthorized}) \wedge$
 $(\text{PBOut } s \text{ (discard (SLc cmd))} = \text{unAuthenticated})$

[PBOut_ind]

$\vdash \forall P.$

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

[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
  
```

$$slState = \text{PLAN_PB} \mid \text{MOVE_TO_ORP} \mid \text{CONDUCT_ORP} \mid \text{MOVE_TO_PB} \\ \mid \text{CONDUCT_PB} \mid \text{COMPLETE_PB}$$

$$stateRole = \text{PlatoonLeader} \mid \text{Omni}$$

6.2 Theorems

[omniCommand_distinct_clauses]

$$\begin{aligned} \vdash & \text{ssmPlanPBComplete} \neq \text{ssmMoveToORPComplete} \wedge \\ & \text{ssmPlanPBComplete} \neq \text{ssmConductORPComplete} \wedge \\ & \text{ssmPlanPBComplete} \neq \text{ssmMoveToPBComplete} \wedge \\ & \text{ssmPlanPBComplete} \neq \text{ssmConductPBComplete} \wedge \\ & \text{ssmPlanPBComplete} \neq \text{invalidOmniCommand} \wedge \\ & \text{ssmMoveToORPComplete} \neq \text{ssmConductORPComplete} \wedge \\ & \text{ssmMoveToORPComplete} \neq \text{ssmMoveToPBComplete} \wedge \\ & \text{ssmMoveToORPComplete} \neq \text{ssmConductPBComplete} \wedge \\ & \text{ssmMoveToORPComplete} \neq \text{invalidOmniCommand} \wedge \\ & \text{ssmConductORPComplete} \neq \text{ssmMoveToPBComplete} \wedge \\ & \text{ssmConductORPComplete} \neq \text{ssmConductPBComplete} \wedge \\ & \text{ssmConductORPComplete} \neq \text{invalidOmniCommand} \wedge \\ & \text{ssmMoveToPBComplete} \neq \text{ssmConductPBComplete} \wedge \\ & \text{ssmMoveToPBComplete} \neq \text{invalidOmniCommand} \wedge \\ & \text{ssmConductPBComplete} \neq \text{invalidOmniCommand} \end{aligned}$$

[plCommand_distinct_clauses]

$$\begin{aligned} \vdash & \text{crossLD} \neq \text{conductORP} \wedge \text{crossLD} \neq \text{moveToPB} \wedge \\ & \text{crossLD} \neq \text{conductPB} \wedge \text{crossLD} \neq \text{completePB} \wedge \\ & \text{crossLD} \neq \text{incomplete} \wedge \text{conductORP} \neq \text{moveToPB} \wedge \\ & \text{conductORP} \neq \text{conductPB} \wedge \text{conductORP} \neq \text{completePB} \wedge \\ & \text{conductORP} \neq \text{incomplete} \wedge \text{moveToPB} \neq \text{conductPB} \wedge \\ & \text{moveToPB} \neq \text{completePB} \wedge \text{moveToPB} \neq \text{incomplete} \wedge \\ & \text{conductPB} \neq \text{completePB} \wedge \text{conductPB} \neq \text{incomplete} \wedge \\ & \text{completePB} \neq \text{incomplete} \end{aligned}$$

[slCommand_distinct_clauses]

$$\vdash \forall a' a. \text{PL } a \neq \text{OMNI } a'$$

[slCommand_one_one]

$$\begin{aligned} \vdash & (\forall a a'. (\text{PL } a = \text{PL } a') \iff (a = a')) \wedge \\ & \forall a a'. (\text{OMNI } a = \text{OMNI } a') \iff (a = a') \end{aligned}$$

[slOutput_distinct_clauses]

$$\begin{aligned} \vdash & \text{PlanPB} \neq \text{MoveToORP} \wedge \text{PlanPB} \neq \text{ConductORP} \wedge \\ & \text{PlanPB} \neq \text{MoveToPB} \wedge \text{PlanPB} \neq \text{ConductPB} \wedge \\ & \text{PlanPB} \neq \text{CompletePB} \wedge \text{PlanPB} \neq \text{unAuthenticated} \wedge \\ & \text{PlanPB} \neq \text{unAuthorized} \wedge \text{MoveToORP} \neq \text{ConductORP} \wedge \\ & \text{MoveToORP} \neq \text{MoveToPB} \wedge \text{MoveToORP} \neq \text{ConductPB} \wedge \\ & \text{MoveToORP} \neq \text{CompletePB} \wedge \text{MoveToORP} \neq \text{unAuthenticated} \wedge \end{aligned}$$


```

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) ∧

```

$(\forall xs \ v_{26} \ v_{25}.$
 $\quad \text{getOmniCommand } (v_{25} \text{ doms } v_{26}::xs) = \text{getOmniCommand } xs) \wedge$
 $(\forall xs \ v_{28} \ v_{27}.$
 $\quad \text{getOmniCommand } (v_{27} \text{ eqs } v_{28}::xs) = \text{getOmniCommand } xs) \wedge$
 $(\forall xs \ v_{30} \ v_{29}.$
 $\quad \text{getOmniCommand } (v_{29} \text{ eqn } v_{30}::xs) = \text{getOmniCommand } xs) \wedge$
 $(\forall xs \ v_{32} \ v_{31}.$
 $\quad \text{getOmniCommand } (v_{31} \text{ lte } v_{32}::xs) = \text{getOmniCommand } xs) \wedge$
 $\forall xs \ v_{34} \ v_{33}.$
 $\quad \text{getOmniCommand } (v_{33} \text{ lt } v_{34}::xs) = \text{getOmniCommand } xs$

[getOmniCommand_ind]

$\vdash \forall P.$
 $\quad P \ \square \ \wedge$
 $\quad (\forall cmd \ xs.$
 $\quad \quad P$
 $\quad \quad (\text{Name Omni controls prop (SOME (SLc (OMNI } cmd)))::}$
 $\quad \quad \quad xs)) \wedge (\forall xs. P \ xs \Rightarrow P \ (\text{TT}::xs)) \wedge$
 $\quad (\forall xs. P \ xs \Rightarrow P \ (\text{FF}::xs)) \wedge$
 $\quad (\forall v_2 \ xs. P \ xs \Rightarrow P \ (\text{prop } v_2::xs)) \wedge$
 $\quad (\forall v_3 \ xs. P \ xs \Rightarrow P \ (\text{notf } v_3::xs)) \wedge$
 $\quad (\forall v_4 \ v_5 \ xs. P \ xs \Rightarrow P \ (v_4 \ \text{andf } v_5::xs)) \wedge$
 $\quad (\forall v_6 \ v_7 \ xs. P \ xs \Rightarrow P \ (v_6 \ \text{orf } v_7::xs)) \wedge$
 $\quad (\forall v_8 \ v_9 \ xs. P \ xs \Rightarrow P \ (v_8 \ \text{impf } v_9::xs)) \wedge$
 $\quad (\forall v_{10} \ v_{11} \ xs. P \ xs \Rightarrow P \ (v_{10} \ \text{eqf } v_{11}::xs)) \wedge$
 $\quad (\forall v_{12} \ v_{13} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says } v_{13}::xs)) \wedge$
 $\quad (\forall v_{14} \ v_{15} \ xs. P \ xs \Rightarrow P \ (v_{14} \ \text{speaks_for } v_{15}::xs)) \wedge$
 $\quad (\forall v_{16} \ xs. P \ xs \Rightarrow P \ (v_{16} \ \text{controls TT}::xs)) \wedge$
 $\quad (\forall v_{16} \ xs. P \ xs \Rightarrow P \ (v_{16} \ \text{controls FF}::xs)) \wedge$
 $\quad (\forall v_{134} \ xs. P \ xs \Rightarrow P \ (\text{Name } v_{134} \ \text{controls prop NONE}::xs)) \wedge$
 $\quad (\forall v_{144} \ xs.$
 $\quad \quad P \ xs \Rightarrow$
 $\quad \quad P \ (\text{Name PlatoonLeader controls prop (SOME } v_{144})::xs)) \wedge$
 $\quad (\forall v_{146} \ xs.$
 $\quad \quad P \ xs \Rightarrow$
 $\quad \quad P \ (\text{Name Omni controls prop (SOME (ESCc } v_{146}))::xs)) \wedge$
 $\quad (\forall v_{150} \ xs.$
 $\quad \quad P \ xs \Rightarrow$
 $\quad \quad P$
 $\quad \quad (\text{Name Omni controls prop (SOME (SLc (PL } v_{150})))::}$
 $\quad \quad \quad xs)) \wedge$
 $\quad (\forall v_{135} \ v_{136} \ v_{68} \ xs.$
 $\quad \quad P \ xs \Rightarrow P \ (v_{135} \ \text{meet } v_{136} \ \text{controls prop } v_{68}::xs)) \wedge$
 $\quad (\forall v_{137} \ v_{138} \ v_{68} \ xs.$
 $\quad \quad P \ xs \Rightarrow P \ (v_{137} \ \text{quoting } v_{138} \ \text{controls prop } v_{68}::xs)) \wedge$
 $\quad (\forall v_{16} \ v_{69} \ xs. P \ xs \Rightarrow P \ (v_{16} \ \text{controls notf } v_{69}::xs)) \wedge$
 $\quad (\forall v_{16} \ v_{70} \ v_{71} \ xs.$
 $\quad \quad P \ xs \Rightarrow P \ (v_{16} \ \text{controls } (v_{70} \ \text{andf } v_{71})::xs)) \wedge$
 $\quad (\forall v_{16} \ v_{72} \ v_{73} \ xs.$

$$\begin{aligned}
& P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } (v_{72} \text{ orf } v_{73})::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{74} \text{ } v_{75} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } (v_{74} \text{ impf } v_{75})::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{76} \text{ } v_{77} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } (v_{76} \text{ eqf } v_{77})::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{78} \text{ } v_{79} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{78} \text{ says } v_{79}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{80} \text{ } v_{81} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{80} \text{ speaks_for } v_{81}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{82} \text{ } v_{83} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{82} \text{ controls } v_{83}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{84} \text{ } v_{85} \text{ } v_{86} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls reps } v_{84} \text{ } v_{85} \text{ } v_{86}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{87} \text{ } v_{88} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{87} \text{ domi } v_{88}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{89} \text{ } v_{90} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{89} \text{ eqi } v_{90}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{91} \text{ } v_{92} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{91} \text{ doms } v_{92}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{93} \text{ } v_{94} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{93} \text{ eqs } v_{94}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{95} \text{ } v_{96} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{95} \text{ eqn } v_{96}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{97} \text{ } v_{98} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{97} \text{ lte } v_{98}::xs)) \wedge \\
& (\forall v_{16} \text{ } v_{99} \text{ } v_{100} \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v_{16} \text{ controls } v_{99} \text{ lt } v_{100}::xs)) \wedge \\
& (\forall v_{18} \text{ } v_{19} \text{ } v_{20} \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (\text{reps } v_{18} \text{ } v_{19} \text{ } v_{20}::xs)) \wedge \\
& (\forall v_{21} \text{ } v_{22} \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v_{21} \text{ domi } v_{22}::xs)) \wedge \\
& (\forall v_{23} \text{ } v_{24} \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v_{23} \text{ eqi } v_{24}::xs)) \wedge \\
& (\forall v_{25} \text{ } v_{26} \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v_{25} \text{ doms } v_{26}::xs)) \wedge \\
& (\forall v_{27} \text{ } v_{28} \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v_{27} \text{ eqs } v_{28}::xs)) \wedge \\
& (\forall v_{29} \text{ } v_{30} \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v_{29} \text{ eqn } v_{30}::xs)) \wedge \\
& (\forall v_{31} \text{ } v_{32} \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v_{31} \text{ lte } v_{32}::xs)) \wedge \\
& (\forall v_{33} \text{ } v_{34} \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v_{33} \text{ lt } v_{34}::xs)) \Rightarrow \\
& \forall v. P \text{ } v
\end{aligned}$$

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

```

[secContext_ind]

```

 $\vdash \forall P.$ 
  ( $\forall x\ xs. P\ \text{PLAN\_PB}\ (x::xs)$ )  $\wedge$ 
  ( $\forall x\ xs. P\ \text{MOVE\_TO\_ORP}\ (x::xs)$ )  $\wedge$ 
  ( $\forall x\ xs. P\ \text{CONDUCT\_ORP}\ (x::xs)$ )  $\wedge$ 
  ( $\forall x\ xs. P\ \text{MOVE\_TO\_PB}\ (x::xs)$ )  $\wedge$ 
  ( $\forall x\ xs. P\ \text{CONDUCT\_PB}\ (x::xs)$ )  $\wedge$  ( $\forall v_4. P\ v_4\ []$ )  $\wedge$ 
  ( $\forall v_5\ v_6. P\ \text{COMPLETE\_PB}\ (v_5::v_6)$ )  $\Rightarrow$ 
   $\forall v\ v_1. P\ v\ v_1$ 

```

8 ssmConductORP Theory

Built: 13 May 2018

Parent Theories: ConductORPType, ssm11, OMNITYPE

8.1 Definitions

[secContextConductORP_def]

```

 $\vdash \forall plcmd\ psgcmd\ incomplete.$ 
  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]

```

 $\vdash \forall slState. \text{ssmConductORPStateInterp}\ slState = \text{TT}$ 

```

8.2 Theorems

[authTestConductORP_cmd_reject_lemma]

```

 $\vdash \forall cmd. \neg \text{authTestConductORP}\ (\text{prop}\ (\text{SOME}\ cmd))$ 

```

[authTestConductORP_def]

$$\begin{aligned} \vdash & (\text{authTestConductORP } (\text{Name PlatoonLeader says prop } cmd) \iff \\ & T) \wedge \\ & (\text{authTestConductORP } (\text{Name PlatoonSergeant says prop } cmd) \iff \\ & T) \wedge (\text{authTestConductORP } TT \iff F) \wedge \\ & (\text{authTestConductORP } FF \iff F) \wedge \\ & (\text{authTestConductORP } (\text{prop } v) \iff F) \wedge \\ & (\text{authTestConductORP } (\text{notf } v_1) \iff F) \wedge \\ & (\text{authTestConductORP } (v_2 \text{ andf } v_3) \iff F) \wedge \\ & (\text{authTestConductORP } (v_4 \text{ orf } v_5) \iff F) \wedge \\ & (\text{authTestConductORP } (v_6 \text{ impf } v_7) \iff F) \wedge \\ & (\text{authTestConductORP } (v_8 \text{ eqf } v_9) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } TT) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } FF) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says notf } v_{67}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } (v_{68} \text{ andf } v_{69})) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } (v_{70} \text{ orf } v_{71})) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } (v_{72} \text{ impf } v_{73})) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75})) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{76} \text{ says } v_{77}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{80} \text{ controls } v_{81}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says reps } v_{82} \ v_{83} \ v_{84}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{85} \text{ domi } v_{86}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{87} \text{ eqi } v_{88}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{89} \text{ doms } v_{90}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{91} \text{ eqs } v_{92}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{93} \text{ eqn } v_{94}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{95} \text{ lte } v_{96}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{10} \text{ says } v_{97} \text{ lt } v_{98}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{12} \text{ speaks_for } v_{13}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{14} \text{ controls } v_{15}) \iff F) \wedge \\ & (\text{authTestConductORP } (\text{reps } v_{16} \ v_{17} \ v_{18}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{19} \text{ domi } v_{20}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{21} \text{ eqi } v_{22}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{23} \text{ doms } v_{24}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{25} \text{ eqs } v_{26}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{27} \text{ eqn } v_{28}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{29} \text{ lte } v_{30}) \iff F) \wedge \\ & (\text{authTestConductORP } (v_{31} \text{ lt } v_{32}) \iff F) \end{aligned}$$

[authTestConductORP_ind]

$$\begin{aligned} \vdash & \forall P. \\ & (\forall cmd. P (\text{Name PlatoonLeader says prop } cmd)) \wedge \\ & (\forall cmd. P (\text{Name PlatoonSergeant says prop } cmd)) \wedge P \ TT \wedge \\ & P \ FF \wedge (\forall v. P (\text{prop } v)) \wedge (\forall v_1. P (\text{notf } v_1)) \wedge \\ & (\forall v_2 \ v_3. P (v_2 \text{ andf } v_3)) \wedge (\forall v_4 \ v_5. P (v_4 \text{ orf } v_5)) \wedge \end{aligned}$$

$$\begin{aligned}
& (\forall v_6 v_7. P (v_6 \text{ impf } v_7)) \wedge (\forall v_8 v_9. P (v_8 \text{ eqf } v_9)) \wedge \\
& (\forall v_{10}. P (v_{10} \text{ says TT})) \wedge (\forall v_{10}. P (v_{10} \text{ says FF})) \wedge \\
& (\forall v_{133} v_{134} v_{66}. P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66})) \wedge \\
& (\forall v_{135} v_{136} v_{66}. P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66})) \wedge \\
& (\forall v_{10} v_{67}. P (v_{10} \text{ says notf } v_{67})) \wedge \\
& (\forall v_{10} v_{68} v_{69}. P (v_{10} \text{ says } (v_{68} \text{ andf } v_{69}))) \wedge \\
& (\forall v_{10} v_{70} v_{71}. P (v_{10} \text{ says } (v_{70} \text{ orf } v_{71}))) \wedge \\
& (\forall v_{10} v_{72} v_{73}. P (v_{10} \text{ says } (v_{72} \text{ impf } v_{73}))) \wedge \\
& (\forall v_{10} v_{74} v_{75}. P (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75}))) \wedge \\
& (\forall v_{10} v_{76} v_{77}. P (v_{10} \text{ says } v_{76} \text{ says } v_{77})) \wedge \\
& (\forall v_{10} v_{78} v_{79}. P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79})) \wedge \\
& (\forall v_{10} v_{80} v_{81}. P (v_{10} \text{ says } v_{80} \text{ controls } v_{81})) \wedge \\
& (\forall v_{10} v_{82} v_{83} v_{84}. P (v_{10} \text{ says reps } v_{82} v_{83} v_{84})) \wedge \\
& (\forall v_{10} v_{85} v_{86}. P (v_{10} \text{ says } v_{85} \text{ domi } v_{86})) \wedge \\
& (\forall v_{10} v_{87} v_{88}. P (v_{10} \text{ says } v_{87} \text{ eqi } v_{88})) \wedge \\
& (\forall v_{10} v_{89} v_{90}. P (v_{10} \text{ says } v_{89} \text{ doms } v_{90})) \wedge \\
& (\forall v_{10} v_{91} v_{92}. P (v_{10} \text{ says } v_{91} \text{ eqs } v_{92})) \wedge \\
& (\forall v_{10} v_{93} v_{94}. P (v_{10} \text{ says } v_{93} \text{ eqn } v_{94})) \wedge \\
& (\forall v_{10} v_{95} v_{96}. P (v_{10} \text{ says } v_{95} \text{ lte } v_{96})) \wedge \\
& (\forall v_{10} v_{97} v_{98}. P (v_{10} \text{ says } v_{97} \text{ lt } v_{98})) \wedge \\
& (\forall v_{12} v_{13}. P (v_{12} \text{ speaks_for } v_{13})) \wedge \\
& (\forall v_{14} v_{15}. P (v_{14} \text{ controls } v_{15})) \wedge \\
& (\forall v_{16} v_{17} v_{18}. P (\text{reps } v_{16} v_{17} v_{18})) \wedge \\
& (\forall v_{19} v_{20}. P (v_{19} \text{ domi } v_{20})) \wedge \\
& (\forall v_{21} v_{22}. P (v_{21} \text{ eqi } v_{22})) \wedge \\
& (\forall v_{23} v_{24}. P (v_{23} \text{ doms } v_{24})) \wedge \\
& (\forall v_{25} v_{26}. P (v_{25} \text{ eqs } v_{26})) \wedge (\forall v_{27} v_{28}. P (v_{27} \text{ eqn } v_{28})) \wedge \\
& (\forall v_{29} v_{30}. P (v_{29} \text{ lte } v_{30})) \wedge (\forall v_{31} v_{32}. P (v_{31} \text{ lt } v_{32})) \Rightarrow \\
& \forall v. P v
\end{aligned}$$

[conductORPNS_def]

$$\begin{aligned}
& \vdash (\text{conductORPNS CONDUCT_ORP (exec (PL secure))} = \text{SECURE}) \wedge \\
& (\text{conductORPNS CONDUCT_ORP (exec (PL plIncomplete))} = \\
& \quad \text{CONDUCT_ORP}) \wedge \\
& (\text{conductORPNS SECURE (exec (PSG actionsIn))} = \text{ACTIONS_IN}) \wedge \\
& (\text{conductORPNS SECURE (exec (PSG psgIncomplete))} = \text{SECURE}) \wedge \\
& (\text{conductORPNS ACTIONS_IN (exec (PL withdraw))} = \text{WITHDRAW}) \wedge \\
& (\text{conductORPNS ACTIONS_IN (exec (PL plIncomplete))} = \\
& \quad \text{ACTIONS_IN}) \wedge \\
& (\text{conductORPNS WITHDRAW (exec (PL complete))} = \text{COMPLETE}) \wedge \\
& (\text{conductORPNS WITHDRAW (exec (PL plIncomplete))} = \text{WITHDRAW}) \wedge \\
& (\text{conductORPNS } s \text{ (trap (PL cmd'))} = s) \wedge \\
& (\text{conductORPNS } s \text{ (trap (PSG cmd))} = s) \wedge \\
& (\text{conductORPNS } s \text{ (discard (PL cmd'))} = s) \wedge \\
& (\text{conductORPNS } s \text{ (discard (PSG cmd))} = s)
\end{aligned}$$

[conductORPNS_ind]

$$\begin{aligned}
& \vdash \forall P. \\
& \quad P \text{ CONDUCT_ORP (exec (PL secure))} \wedge
\end{aligned}$$

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

[conductORPOut_def]

\vdash (conductORPOut CONDUCT_ORP (exec (PL secure)) = Secure) \wedge
 (conductORPOut CONDUCT_ORP (exec (PL plIncomplete)) =
 ConductORP) \wedge
 (conductORPOut SECURE (exec (PSG actionsIn)) = ActionsIn) \wedge
 (conductORPOut SECURE (exec (PSG psgIncomplete)) = Secure) \wedge
 (conductORPOut ACTIONS_IN (exec (PL withdraw)) = Withdraw) \wedge
 (conductORPOut ACTIONS_IN (exec (PL plIncomplete)) =
 ActionsIn) \wedge
 (conductORPOut WITHDRAW (exec (PL complete)) = Complete) \wedge
 (conductORPOut WITHDRAW (exec (PL plIncomplete)) =
 Withdraw) \wedge
 (conductORPOut s (trap (PL cmd')) = unauthorized) \wedge
 (conductORPOut s (trap (PSG cmd)) = unauthorized) \wedge
 (conductORPOut s (discard (PL cmd')) = unAuthenticated) \wedge
 (conductORPOut s (discard (PSG cmd)) = unAuthenticated)

[conductORPOut_ind]

$\vdash \forall P.$
 P CONDUCT_ORP (exec (PL secure)) \wedge
 P CONDUCT_ORP (exec (PL plIncomplete)) \wedge
 P SECURE (exec (PSG actionsIn)) \wedge
 P SECURE (exec (PSG psgIncomplete)) \wedge
 P ACTIONS_IN (exec (PL withdraw)) \wedge
 P ACTIONS_IN (exec (PL plIncomplete)) \wedge

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

[PlatoonLeader_exec_plCommand_justified_thm]

$$\begin{aligned}
& \vdash \forall NS \text{ Out } M \ Oi \ Os. \\
& \text{TR } (M, Oi, Os) \ (\text{exec } (\text{SLc } (\text{PL } plCommand))) \\
& \quad (\text{CFG authTestConductORP ssmConductORPStateInterp} \\
& \quad \quad (\text{secContextConductORP } plCommand \ psgCommand \ incomplete) \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL } plCommand)))::ins) \ s \ outs) \\
& \quad (\text{CFG authTestConductORP ssmConductORPStateInterp} \\
& \quad \quad (\text{secContextConductORP } plCommand \ psgCommand \ incomplete) \\
& \quad \quad \quad ins \ (NS \ s \ (\text{exec } (\text{SLc } (\text{PL } plCommand)))) \\
& \quad \quad \quad (\text{Out } s \ (\text{exec } (\text{SLc } (\text{PL } plCommand)))::outs)) \iff \\
& \text{authTestConductORP} \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL } plCommand)))) \wedge \\
& \text{CFGInterpret } (M, Oi, Os) \\
& \quad (\text{CFG authTestConductORP ssmConductORPStateInterp} \\
& \quad \quad (\text{secContextConductORP } plCommand \ psgCommand \ incomplete) \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL } plCommand)))::ins) \ s \ outs) \wedge \\
& \quad (M, Oi, Os) \text{ sat } \text{prop } (\text{SOME } (\text{SLc } (\text{PL } plCommand)))
\end{aligned}$$

[PlatoonLeader_plCommand_lemma]

$$\begin{aligned}
& \vdash \text{CFGInterpret } (M, Oi, Os) \\
& \quad (\text{CFG authTestConductORP ssmConductORPStateInterp} \\
& \quad \quad (\text{secContextConductORP } plCommand \ psgCommand \ incomplete) \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL } plCommand)))::ins) \ s \ outs) \Rightarrow \\
& \quad (M, Oi, Os) \text{ sat } \text{prop } (\text{SOME } (\text{SLc } (\text{PL } plCommand)))
\end{aligned}$$

[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]

$$\vdash \text{secure} \neq \text{withdraw} \wedge \text{secure} \neq \text{complete} \wedge \\ \text{secure} \neq \text{plIncomplete} \wedge \text{withdraw} \neq \text{complete} \wedge \\ \text{withdraw} \neq \text{plIncomplete} \wedge \text{complete} \neq \text{plIncomplete}$$

[psgCommand_distinct_clauses]

$$\vdash \text{actionsIn} \neq \text{psgIncomplete}$$

[slCommand_distinct_clauses]

$$\vdash \forall a' a. \text{PL } a \neq \text{PSG } a'$$

[slCommand_one_one]

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

[slOutput_distinct_clauses]

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

[slRole_distinct_clauses]

$$\vdash \text{PlatoonLeader} \neq \text{PlatoonSergeant}$$

[slState_distinct_clauses]

$$\vdash \text{CONDUCT_ORP} \neq \text{SECURE} \wedge \text{CONDUCT_ORP} \neq \text{ACTIONS_IN} \wedge \\ \text{CONDUCT_ORP} \neq \text{WITHDRAW} \wedge \text{CONDUCT_ORP} \neq \text{COMPLETE} \wedge \\ \text{SECURE} \neq \text{ACTIONS_IN} \wedge \text{SECURE} \neq \text{WITHDRAW} \wedge \text{SECURE} \neq \text{COMPLETE} \wedge \\ \text{ACTIONS_IN} \neq \text{WITHDRAW} \wedge \text{ACTIONS_IN} \neq \text{COMPLETE} \wedge \\ \text{WITHDRAW} \neq \text{COMPLETE}$$

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 v1) ⇔ F) ∧
  (authTestConductPB (v2 andf v3) ⇔ F) ∧
  (authTestConductPB (v4 orf v5) ⇔ F) ∧
  (authTestConductPB (v6 impf v7) ⇔ F) ∧
  (authTestConductPB (v8 eqf v9) ⇔ F) ∧
  (authTestConductPB (v10 says TT) ⇔ F) ∧
  (authTestConductPB (v10 says FF) ⇔ F) ∧
  (authTestConductPB (v133 meet v134 says prop v66) ⇔ F) ∧
  (authTestConductPB (v135 quoting v136 says prop v66) ⇔ F) ∧
  (authTestConductPB (v10 says notf v67) ⇔ F) ∧
  (authTestConductPB (v10 says (v68 andf v69)) ⇔ F) ∧
  (authTestConductPB (v10 says (v70 orf v71)) ⇔ F) ∧
  (authTestConductPB (v10 says (v72 impf v73)) ⇔ F) ∧
  (authTestConductPB (v10 says (v74 eqf v75)) ⇔ F) ∧
  (authTestConductPB (v10 says v76 says v77) ⇔ F) ∧
  (authTestConductPB (v10 says v78 speaks_for v79) ⇔ F) ∧
  (authTestConductPB (v10 says v80 controls v81) ⇔ F) ∧
  (authTestConductPB (v10 says reps v82 v83 v84) ⇔ F) ∧
  (authTestConductPB (v10 says v85 domi v86) ⇔ F) ∧
  (authTestConductPB (v10 says v87 eqi v88) ⇔ F) ∧
  (authTestConductPB (v10 says v89 doms v90) ⇔ F) ∧

```

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

$[\text{authTestConductPB_ind}]$

$\vdash \forall P.$

$(\forall \text{cmd}. P (\text{Name PlatoonLeader says prop cmd})) \wedge$
 $(\forall \text{cmd}. P (\text{Name PlatoonSergeant says prop cmd})) \wedge P \text{ TT} \wedge$
 $P \text{ FF} \wedge (\forall v. P (\text{prop } v)) \wedge (\forall v_1. P (\text{notf } v_1)) \wedge$
 $(\forall v_2 \ v_3. P (v_2 \text{ andf } v_3)) \wedge (\forall v_4 \ v_5. P (v_4 \text{ orf } v_5)) \wedge$
 $(\forall v_6 \ v_7. P (v_6 \text{ impf } v_7)) \wedge (\forall v_8 \ v_9. P (v_8 \text{ eqf } v_9)) \wedge$
 $(\forall v_{10}. P (v_{10} \text{ says TT})) \wedge (\forall v_{10}. P (v_{10} \text{ says FF})) \wedge$
 $(\forall v_{133} \ v_{134} \ v_{66}. P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66})) \wedge$
 $(\forall v_{135} \ v_{136} \ v_{66}. P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66})) \wedge$
 $(\forall v_{10} \ v_{67}. P (v_{10} \text{ says notf } v_{67})) \wedge$
 $(\forall v_{10} \ v_{68} \ v_{69}. P (v_{10} \text{ says } (v_{68} \text{ andf } v_{69}))) \wedge$
 $(\forall v_{10} \ v_{70} \ v_{71}. P (v_{10} \text{ says } (v_{70} \text{ orf } v_{71}))) \wedge$
 $(\forall v_{10} \ v_{72} \ v_{73}. P (v_{10} \text{ says } (v_{72} \text{ impf } v_{73}))) \wedge$
 $(\forall v_{10} \ v_{74} \ v_{75}. P (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75}))) \wedge$
 $(\forall v_{10} \ v_{76} \ v_{77}. P (v_{10} \text{ says } v_{76} \text{ says } v_{77})) \wedge$
 $(\forall v_{10} \ v_{78} \ v_{79}. P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79})) \wedge$
 $(\forall v_{10} \ v_{80} \ v_{81}. P (v_{10} \text{ says } v_{80} \text{ controls } v_{81})) \wedge$
 $(\forall v_{10} \ v_{82} \ v_{83} \ v_{84}. P (v_{10} \text{ says reps } v_{82} \ v_{83} \ v_{84})) \wedge$
 $(\forall v_{10} \ v_{85} \ v_{86}. P (v_{10} \text{ says } v_{85} \text{ domi } v_{86})) \wedge$
 $(\forall v_{10} \ v_{87} \ v_{88}. P (v_{10} \text{ says } v_{87} \text{ eqi } v_{88})) \wedge$
 $(\forall v_{10} \ v_{89} \ v_{90}. P (v_{10} \text{ says } v_{89} \text{ doms } v_{90})) \wedge$
 $(\forall v_{10} \ v_{91} \ v_{92}. P (v_{10} \text{ says } v_{91} \text{ eqs } v_{92})) \wedge$
 $(\forall v_{10} \ v_{93} \ v_{94}. P (v_{10} \text{ says } v_{93} \text{ eqn } v_{94})) \wedge$
 $(\forall v_{10} \ v_{95} \ v_{96}. P (v_{10} \text{ says } v_{95} \text{ lte } v_{96})) \wedge$
 $(\forall v_{10} \ v_{97} \ v_{98}. P (v_{10} \text{ says } v_{97} \text{ lt } v_{98})) \wedge$
 $(\forall v_{12} \ v_{13}. P (v_{12} \text{ speaks_for } v_{13})) \wedge$
 $(\forall v_{14} \ v_{15}. P (v_{14} \text{ controls } v_{15})) \wedge$
 $(\forall v_{16} \ v_{17} \ v_{18}. P (\text{reps } v_{16} \ v_{17} \ v_{18})) \wedge$
 $(\forall v_{19} \ v_{20}. P (v_{19} \text{ domi } v_{20})) \wedge$
 $(\forall v_{21} \ v_{22}. P (v_{21} \text{ eqi } v_{22})) \wedge$
 $(\forall v_{23} \ v_{24}. P (v_{23} \text{ doms } v_{24})) \wedge$
 $(\forall v_{25} \ v_{26}. P (v_{25} \text{ eqs } v_{26})) \wedge (\forall v_{27} \ v_{28}. P (v_{27} \text{ eqn } v_{28})) \wedge$
 $(\forall v_{29} \ v_{30}. P (v_{29} \text{ lte } v_{30})) \wedge (\forall v_{31} \ v_{32}. P (v_{31} \text{ lt } v_{32})) \Rightarrow$

$$\forall v. P \ v$$

[conductPBNS_def]

$$\begin{aligned} \vdash & (\text{conductPBNS CONDUCT_PB (exec (PL securePB))} = \text{SECURE_PB}) \wedge \\ & (\text{conductPBNS CONDUCT_PB (exec (PL plIncompletePB))} = \\ & \quad \text{CONDUCT_PB}) \wedge \\ & (\text{conductPBNS SECURE_PB (exec (PSG actionsInPB))} = \\ & \quad \text{ACTIONS_IN_PB}) \wedge \\ & (\text{conductPBNS SECURE_PB (exec (PSG psgIncompletePB))} = \\ & \quad \text{SECURE_PB}) \wedge \\ & (\text{conductPBNS ACTIONS_IN_PB (exec (PL withdrawPB))} = \\ & \quad \text{WITHDRAW_PB}) \wedge \\ & (\text{conductPBNS ACTIONS_IN_PB (exec (PL plIncompletePB))} = \\ & \quad \text{ACTIONS_IN_PB}) \wedge \\ & (\text{conductPBNS WITHDRAW_PB (exec (PL completePB))} = \\ & \quad \text{COMPLETE_PB}) \wedge \\ & (\text{conductPBNS WITHDRAW_PB (exec (PL plIncompletePB))} = \\ & \quad \text{WITHDRAW_PB}) \wedge (\text{conductPBNS } s \text{ (trap (PL cmd'))} = s) \wedge \\ & (\text{conductPBNS } s \text{ (trap (PSG cmd))} = s) \wedge \\ & (\text{conductPBNS } s \text{ (discard (PL cmd'))} = s) \wedge \\ & (\text{conductPBNS } s \text{ (discard (PSG cmd))} = s) \end{aligned}$$

[conductPBNS_ind]

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

[conductPBOut_def]

$$\begin{aligned}
&\vdash (\text{conductPBOut CONDUCT_PB (exec (PL securePB))} = \text{ConductPB}) \wedge \\
&\quad (\text{conductPBOut CONDUCT_PB (exec (PL plIncompletePB))} = \\
&\quad \quad \text{ConductPB}) \wedge \\
&\quad (\text{conductPBOut SECURE_PB (exec (PSG actionsInPB))} = \\
&\quad \quad \text{SecurePB}) \wedge \\
&\quad (\text{conductPBOut SECURE_PB (exec (PSG psgIncompletePB))} = \\
&\quad \quad \text{SecurePB}) \wedge \\
&\quad (\text{conductPBOut ACTIONS_IN_PB (exec (PL withdrawPB))} = \\
&\quad \quad \text{ActionsInPB}) \wedge \\
&\quad (\text{conductPBOut ACTIONS_IN_PB (exec (PL plIncompletePB))} = \\
&\quad \quad \text{ActionsInPB}) \wedge \\
&\quad (\text{conductPBOut WITHDRAW_PB (exec (PL completePB))} = \\
&\quad \quad \text{WithdrawPB}) \wedge \\
&\quad (\text{conductPBOut WITHDRAW_PB (exec (PL plIncompletePB))} = \\
&\quad \quad \text{WithdrawPB}) \wedge \\
&\quad (\text{conductPBOut } s \text{ (trap (PL cmd'))} = \text{unAuthorized}) \wedge \\
&\quad (\text{conductPBOut } s \text{ (trap (PSG cmd))} = \text{unAuthorized}) \wedge \\
&\quad (\text{conductPBOut } s \text{ (discard (PL cmd'))} = \text{unAuthenticated}) \wedge \\
&\quad (\text{conductPBOut } s \text{ (discard (PSG cmd))} = \text{unAuthenticated})
\end{aligned}$$
[conductPBOut_ind]

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

[PlatoonLeader_exec_plCommandPB_justified_thm]

$\vdash \forall NS \text{ Out } M \text{ } Oi \text{ } Os.$
 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)) \iff
 authTestConductPB
 (Name PlatoonLeader says
 prop (SOME (SLc (PL $plCommand$)))) \wedge
 CFGInterpret (M, Oi, Os)
 (CFG authTestConductPB ssmConductPBStateInterp
 (secContextConductPB $plCommand$ $psgCommand$ incomplete)
 (Name PlatoonLeader says
 prop (SOME (SLc (PL $plCommand$)))::ins) s outs) \wedge
 (M, Oi, Os) sat prop (SOME (SLc (PL $plCommand$)))

[PlatoonLeader_plCommandPB_lemma]

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

[PlatoonSergeant_exec_psgCommandPB_justified_thm]

$\vdash \forall NS \text{ Out } M \text{ } Oi \text{ } Os.$
 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)) \iff
 authTestConductPB
 (Name PlatoonSergeant says
 prop (SOME (SLc (PSG $psgCommand$)))) \wedge
 CFGInterpret (M, Oi, Os)
 (CFG authTestConductPB ssmConductPBStateInterp
 (secContextConductPB $plCommand$ $psgCommand$ incomplete)
 (Name PlatoonSergeant says
 prop (SOME (SLc (PSG $psgCommand$)))::ins) s outs) \wedge
 (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]

$$\begin{aligned}
&\vdash \text{ConductPB} \neq \text{SecurePB} \wedge \text{ConductPB} \neq \text{ActionsInPB} \wedge \\
&\quad \text{ConductPB} \neq \text{WithdrawPB} \wedge \text{ConductPB} \neq \text{CompletePB} \wedge \\
&\quad \text{ConductPB} \neq \text{unAuthenticated} \wedge \text{ConductPB} \neq \text{unAuthorized} \wedge \\
&\quad \text{SecurePB} \neq \text{ActionsInPB} \wedge \text{SecurePB} \neq \text{WithdrawPB} \wedge \\
&\quad \text{SecurePB} \neq \text{CompletePB} \wedge \text{SecurePB} \neq \text{unAuthenticated} \wedge \\
&\quad \text{SecurePB} \neq \text{unAuthorized} \wedge \text{ActionsInPB} \neq \text{WithdrawPB} \wedge \\
&\quad \text{ActionsInPB} \neq \text{CompletePB} \wedge \text{ActionsInPB} \neq \text{unAuthenticated} \wedge \\
&\quad \text{ActionsInPB} \neq \text{unAuthorized} \wedge \text{WithdrawPB} \neq \text{CompletePB} \wedge \\
&\quad \text{WithdrawPB} \neq \text{unAuthenticated} \wedge \text{WithdrawPB} \neq \text{unAuthorized} \wedge \\
&\quad \text{CompletePB} \neq \text{unAuthenticated} \wedge \text{CompletePB} \neq \text{unAuthorized} \wedge \\
&\quad \text{unAuthenticated} \neq \text{unAuthorized}
\end{aligned}$$

[slRole_distinct_clauses]

$$\vdash \text{PlatoonLeader} \neq \text{PlatoonSergeant}$$

[slState_distinct_clauses]

$$\begin{aligned}
&\vdash \text{CONDUCT_PB} \neq \text{SECURE_PB} \wedge \text{CONDUCT_PB} \neq \text{ACTIONS_IN_PB} \wedge \\
&\quad \text{CONDUCT_PB} \neq \text{WITHDRAW_PB} \wedge \text{CONDUCT_PB} \neq \text{COMPLETE_PB} \wedge \\
&\quad \text{SECURE_PB} \neq \text{ACTIONS_IN_PB} \wedge \text{SECURE_PB} \neq \text{WITHDRAW_PB} \wedge \\
&\quad \text{SECURE_PB} \neq \text{COMPLETE_PB} \wedge \text{ACTIONS_IN_PB} \neq \text{WITHDRAW_PB} \wedge \\
&\quad \text{ACTIONS_IN_PB} \neq \text{COMPLETE_PB} \wedge \text{WITHDRAW_PB} \neq \text{COMPLETE_PB}
\end{aligned}$$

12 ssmMoveToORP Theory

Built: 13 May 2018

Parent Theories: MoveToORPType, ssm11, OMNIType

12.1 Definitions

[secContextMoveToORP_def]

$$\begin{aligned}
&\vdash \forall cmd. \\
&\quad \text{secContextMoveToORP } cmd = \\
&\quad [\text{Name PlatoonLeader controls prop (SOME (SLc cmd))}]
\end{aligned}$$

[ssmMoveToORPStateInterp_def]

$$\vdash \forall state. \text{ssmMoveToORPStateInterp } state = \text{TT}$$

12.2 Theorems

[authTestMoveToORP_cmd_reject_lemma]

$$\vdash \forall cmd. \neg \text{authTestMoveToORP (prop (SOME cmd))}$$

[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 v₁) \iff F) \wedge
 (authTestMoveToORP (v₂ andf v₃) \iff F) \wedge
 (authTestMoveToORP (v₄ orf v₅) \iff F) \wedge
 (authTestMoveToORP (v₆ impf v₇) \iff F) \wedge
 (authTestMoveToORP (v₈ eqf v₉) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says TT) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says FF) \iff F) \wedge
 (authTestMoveToORP (v133 meet v134 says prop v₆₆) \iff F) \wedge
 (authTestMoveToORP (v135 quoting v136 says prop v₆₆) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says notf v₆₇) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says (v₆₈ andf v₆₉)) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says (v₇₀ orf v₇₁)) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says (v₇₂ impf v₇₃)) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says (v₇₄ eqf v₇₅)) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₇₆ says v₇₇) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₇₈ speaks_for v₇₉) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₈₀ controls v₈₁) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says reps v₈₂ v₈₃ v₈₄) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₈₅ domi v₈₆) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₈₇ eqi v₈₈) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₈₉ doms v₉₀) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₉₁ eqs v₉₂) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₉₃ eqn v₉₄) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₉₅ lte v₉₆) \iff F) \wedge
 (authTestMoveToORP (v₁₀ says v₉₇ lt v₉₈) \iff F) \wedge
 (authTestMoveToORP (v₁₂ speaks_for v₁₃) \iff F) \wedge
 (authTestMoveToORP (v₁₄ controls v₁₅) \iff F) \wedge
 (authTestMoveToORP (reps v₁₆ v₁₇ v₁₈) \iff F) \wedge
 (authTestMoveToORP (v₁₉ domi v₂₀) \iff F) \wedge
 (authTestMoveToORP (v₂₁ eqi v₂₂) \iff F) \wedge
 (authTestMoveToORP (v₂₃ doms v₂₄) \iff F) \wedge
 (authTestMoveToORP (v₂₅ eqs v₂₆) \iff F) \wedge
 (authTestMoveToORP (v₂₇ eqn v₂₈) \iff F) \wedge
 (authTestMoveToORP (v₂₉ lte v₃₀) \iff F) \wedge
 (authTestMoveToORP (v₃₁ lt v₃₂) \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 v₁)) \wedge
 ($\forall v_2 v_3.$ P (v₂ andf v₃)) \wedge ($\forall v_4 v_5.$ P (v₄ orf v₅)) \wedge
 ($\forall v_6 v_7.$ P (v₆ impf v₇)) \wedge ($\forall v_8 v_9.$ P (v₈ eqf v₉)) \wedge
 ($\forall v_{10}.$ P (v₁₀ says TT)) \wedge ($\forall v_{10}.$ P (v₁₀ says FF)) \wedge
 ($\forall v_{133} v_{134} v_{66}.$ P (v133 meet v134 says prop v₆₆)) \wedge
 ($\forall v_{135} v_{136} v_{66}.$ P (v135 quoting v136 says prop v₆₆)) \wedge

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

[moveToORPNS_def]

$$\begin{aligned}
& \vdash (\text{moveToORPNS MOVE_TO_ORP (exec (SLc pltForm))} = \text{PLT_FORM}) \wedge \\
& (\text{moveToORPNS MOVE_TO_ORP (exec (SLc incomplete))} = \\
& \quad \text{MOVE_TO_ORP}) \wedge \\
& (\text{moveToORPNS PLT_FORM (exec (SLc pltMove))} = \text{PLT_MOVE}) \wedge \\
& (\text{moveToORPNS PLT_FORM (exec (SLc incomplete))} = \text{PLT_FORM}) \wedge \\
& (\text{moveToORPNS PLT_MOVE (exec (SLc pltSecureHalt))} = \\
& \quad \text{PLT_SECURE_HALT}) \wedge \\
& (\text{moveToORPNS PLT_MOVE (exec (SLc incomplete))} = \text{PLT_MOVE}) \wedge \\
& (\text{moveToORPNS PLT_SECURE_HALT (exec (SLc complete))} = \\
& \quad \text{COMPLETE}) \wedge \\
& (\text{moveToORPNS PLT_SECURE_HALT (exec (SLc incomplete))} = \\
& \quad \text{PLT_SECURE_HALT}) \wedge (\text{moveToORPNS } s \text{ (trap (SLc cmd))} = s) \wedge \\
& (\text{moveToORPNS } s \text{ (discard (SLc cmd))} = s)
\end{aligned}$$

[moveToORPNS_ind]

$$\begin{aligned}
& \vdash \forall P. \\
& \quad P \text{ MOVE_TO_ORP (exec (SLc pltForm))} \wedge \\
& \quad P \text{ MOVE_TO_ORP (exec (SLc incomplete))} \wedge \\
& \quad P \text{ PLT_FORM (exec (SLc pltMove))} \wedge \\
& \quad P \text{ PLT_FORM (exec (SLc incomplete))} \wedge \\
& \quad P \text{ PLT_MOVE (exec (SLc pltSecureHalt))} \wedge \\
& \quad P \text{ PLT_MOVE (exec (SLc incomplete))} \wedge
\end{aligned}$$

$$\begin{aligned}
& P \text{ PLT_SECURE_HALT (exec (SLc complete))} \wedge \\
& P \text{ PLT_SECURE_HALT (exec (SLc incomplete))} \wedge \\
& (\forall s \text{ cmd. } P \text{ s (trap (SLc cmd))}) \wedge \\
& (\forall s \text{ cmd. } P \text{ s (discard (SLc cmd))}) \wedge \\
& (\forall s \text{ v}_6. P \text{ s (discard (ESCc v}_6\text{))}) \wedge \\
& (\forall s \text{ v}_9. P \text{ s (trap (ESCc v}_9\text{))}) \wedge \\
& (\forall v_{12}. P \text{ MOVE_TO_ORP (exec (ESCc v}_{12}\text{))}) \wedge \\
& P \text{ MOVE_TO_ORP (exec (SLc pltMove))} \wedge \\
& P \text{ MOVE_TO_ORP (exec (SLc pltSecureHalt))} \wedge \\
& P \text{ MOVE_TO_ORP (exec (SLc complete))} \wedge \\
& (\forall v_{15}. P \text{ PLT_FORM (exec (ESCc v}_{15}\text{))}) \wedge \\
& P \text{ PLT_FORM (exec (SLc pltForm))} \wedge \\
& P \text{ PLT_FORM (exec (SLc pltSecureHalt))} \wedge \\
& P \text{ PLT_FORM (exec (SLc complete))} \wedge \\
& (\forall v_{18}. P \text{ PLT_MOVE (exec (ESCc v}_{18}\text{))}) \wedge \\
& P \text{ PLT_MOVE (exec (SLc pltForm))} \wedge \\
& P \text{ PLT_MOVE (exec (SLc pltMove))} \wedge \\
& P \text{ PLT_MOVE (exec (SLc complete))} \wedge \\
& (\forall v_{21}. P \text{ PLT_SECURE_HALT (exec (ESCc v}_{21}\text{))}) \wedge \\
& P \text{ PLT_SECURE_HALT (exec (SLc pltForm))} \wedge \\
& P \text{ PLT_SECURE_HALT (exec (SLc pltMove))} \wedge \\
& P \text{ PLT_SECURE_HALT (exec (SLc pltSecureHalt))} \wedge \\
& (\forall v_{23}. P \text{ COMPLETE (exec v}_{23}\text{)}) \Rightarrow \\
& \forall v \text{ v}_1. P \text{ v v}_1
\end{aligned}$$

[moveToORPOut_def]

$$\begin{aligned}
& \vdash (\text{moveToORPOut MOVE_TO_ORP (exec (SLc pltForm))} = \text{PLTForm}) \wedge \\
& (\text{moveToORPOut MOVE_TO_ORP (exec (SLc incomplete))} = \\
& \quad \text{MoveToORP}) \wedge \\
& (\text{moveToORPOut PLT_FORM (exec (SLc pltMove))} = \text{PLTMove}) \wedge \\
& (\text{moveToORPOut PLT_FORM (exec (SLc incomplete))} = \text{PLTForm}) \wedge \\
& (\text{moveToORPOut PLT_MOVE (exec (SLc pltSecureHalt))} = \\
& \quad \text{PLTSecureHalt}) \wedge \\
& (\text{moveToORPOut PLT_MOVE (exec (SLc incomplete))} = \text{PLTMove}) \wedge \\
& (\text{moveToORPOut PLT_SECURE_HALT (exec (SLc complete))} = \\
& \quad \text{Complete}) \wedge \\
& (\text{moveToORPOut PLT_SECURE_HALT (exec (SLc incomplete))} = \\
& \quad \text{PLTSecureHalt}) \wedge \\
& (\text{moveToORPOut s (trap (SLc cmd))} = \text{unAuthorized}) \wedge \\
& (\text{moveToORPOut s (discard (SLc cmd))} = \text{unAuthenticated})
\end{aligned}$$

[moveToORPOut_ind]

$$\begin{aligned}
& \vdash \forall P. \\
& \quad P \text{ MOVE_TO_ORP (exec (SLc pltForm))} \wedge \\
& \quad P \text{ MOVE_TO_ORP (exec (SLc incomplete))} \wedge \\
& \quad P \text{ PLT_FORM (exec (SLc pltMove))} \wedge \\
& \quad P \text{ PLT_FORM (exec (SLc incomplete))} \wedge \\
& \quad P \text{ PLT_MOVE (exec (SLc pltSecureHalt))} \wedge \\
& \quad P \text{ PLT_MOVE (exec (SLc incomplete))} \wedge
\end{aligned}$$

$P \text{ PLT_SECURE_HALT (exec (SLc complete))} \wedge$
 $P \text{ PLT_SECURE_HALT (exec (SLc incomplete))} \wedge$
 $(\forall s \text{ cmd. } P \text{ s (trap (SLc cmd))}) \wedge$
 $(\forall s \text{ cmd. } P \text{ s (discard (SLc cmd))}) \wedge$
 $(\forall s \text{ v}_6. P \text{ s (discard (ESCc v}_6\text{))}) \wedge$
 $(\forall s \text{ v}_9. P \text{ s (trap (ESCc v}_9\text{))}) \wedge$
 $(\forall v_{12}. P \text{ MOVE_TO_ORP (exec (ESCc v}_{12}\text{))}) \wedge$
 $P \text{ MOVE_TO_ORP (exec (SLc pltMove))} \wedge$
 $P \text{ MOVE_TO_ORP (exec (SLc pltSecureHalt))} \wedge$
 $P \text{ MOVE_TO_ORP (exec (SLc complete))} \wedge$
 $(\forall v_{15}. P \text{ PLT_FORM (exec (ESCc v}_{15}\text{))}) \wedge$
 $P \text{ PLT_FORM (exec (SLc pltForm))} \wedge$
 $P \text{ PLT_FORM (exec (SLc pltSecureHalt))} \wedge$
 $P \text{ PLT_FORM (exec (SLc complete))} \wedge$
 $(\forall v_{18}. P \text{ PLT_MOVE (exec (ESCc v}_{18}\text{))}) \wedge$
 $P \text{ PLT_MOVE (exec (SLc pltForm))} \wedge$
 $P \text{ PLT_MOVE (exec (SLc pltMove))} \wedge$
 $P \text{ PLT_MOVE (exec (SLc complete))} \wedge$
 $(\forall v_{21}. P \text{ PLT_SECURE_HALT (exec (ESCc v}_{21}\text{))}) \wedge$
 $P \text{ PLT_SECURE_HALT (exec (SLc pltForm))} \wedge$
 $P \text{ PLT_SECURE_HALT (exec (SLc pltMove))} \wedge$
 $P \text{ PLT_SECURE_HALT (exec (SLc pltSecureHalt))} \wedge$
 $(\forall v_{23}. P \text{ COMPLETE (exec v}_{23}\text{)}) \Rightarrow$
 $\forall v \text{ v}_1. P \text{ v v}_1$

[PlatoonLeader_exec_slCommand_justified_thm]

$\vdash \forall NS \text{ Out } M \text{ Oi } Os.$
 $\text{TR } (M, Oi, Os) \text{ (exec (SLc slCommand))}$
 $(\text{CFG authTestMoveToORP ssmMoveToORPStateInterp}$
 $\quad (\text{secContextMoveToORP slCommand})$
 $\quad (\text{Name PlatoonLeader says prop (SOME (SLc slCommand))} ::$
 $\quad \quad \text{ins) s outs})$
 $(\text{CFG authTestMoveToORP ssmMoveToORPStateInterp}$
 $\quad (\text{secContextMoveToORP slCommand}) \text{ ins}$
 $\quad (NS \text{ s (exec (SLc slCommand))})$
 $\quad (\text{Out s (exec (SLc slCommand))} :: \text{outs})) \iff$
 authTestMoveToORP
 $\quad (\text{Name PlatoonLeader says prop (SOME (SLc slCommand))}) \wedge$
 $\text{CFGInterpret } (M, Oi, Os)$
 $\quad (\text{CFG authTestMoveToORP ssmMoveToORPStateInterp}$
 $\quad \quad (\text{secContextMoveToORP slCommand})$
 $\quad \quad (\text{Name PlatoonLeader says prop (SOME (SLc slCommand))} ::$
 $\quad \quad \quad \text{ins) s outs}) \wedge$
 $(M, Oi, Os) \text{ sat prop (SOME (SLc slCommand))})$

[PlatoonLeader_slCommand_lemma]

$\vdash \text{CFGInterpret } (M, Oi, Os)$
 $(\text{CFG authTestMoveToORP ssmMoveToORPStateInterp}$
 $\quad (\text{secContextMoveToORP slCommand})$

$(\text{Name PlatoonLeader says prop (SOME (SLc slCommand)))::$
 $\text{ins) } s \text{ outs) } \Rightarrow$
 $(M, Oi, Os) \text{ sat prop (SOME (SLc slCommand))}$

13 MoveToORPType Theory

Built: 13 May 2018

Parent Theories: indexedLists, patternMatches

13.1 Datatypes

$\text{slCommand} = \text{pltForm} \mid \text{pltMove} \mid \text{pltSecureHalt} \mid \text{complete}$
 $\quad \mid \text{incomplete}$
 $\text{slOutput} = \text{MoveToORP} \mid \text{PLTForm} \mid \text{PLTMove} \mid \text{PLTSecureHalt}$
 $\quad \mid \text{Complete} \mid \text{unAuthorized} \mid \text{unAuthenticated}$
 $\text{slState} = \text{MOVE_TO_ORP} \mid \text{PLT_FORM} \mid \text{PLT_MOVE} \mid \text{PLT_SECURE_HALT}$
 $\quad \mid \text{COMPLETE}$
 $\text{stateRole} = \text{PlatoonLeader}$

13.2 Theorems

[slCommand_distinct_clauses]

$\vdash \text{pltForm} \neq \text{pltMove} \wedge \text{pltForm} \neq \text{pltSecureHalt} \wedge$
 $\text{pltForm} \neq \text{complete} \wedge \text{pltForm} \neq \text{incomplete} \wedge$
 $\text{pltMove} \neq \text{pltSecureHalt} \wedge \text{pltMove} \neq \text{complete} \wedge$
 $\text{pltMove} \neq \text{incomplete} \wedge \text{pltSecureHalt} \neq \text{complete} \wedge$
 $\text{pltSecureHalt} \neq \text{incomplete} \wedge \text{complete} \neq \text{incomplete}$

[slOutput_distinct_clauses]

$\vdash \text{MoveToORP} \neq \text{PLTForm} \wedge \text{MoveToORP} \neq \text{PLTMove} \wedge$
 $\text{MoveToORP} \neq \text{PLTSecureHalt} \wedge \text{MoveToORP} \neq \text{Complete} \wedge$
 $\text{MoveToORP} \neq \text{unAuthorized} \wedge \text{MoveToORP} \neq \text{unAuthenticated} \wedge$
 $\text{PLTForm} \neq \text{PLTMove} \wedge \text{PLTForm} \neq \text{PLTSecureHalt} \wedge$
 $\text{PLTForm} \neq \text{Complete} \wedge \text{PLTForm} \neq \text{unAuthorized} \wedge$
 $\text{PLTForm} \neq \text{unAuthenticated} \wedge \text{PLTMove} \neq \text{PLTSecureHalt} \wedge$
 $\text{PLTMove} \neq \text{Complete} \wedge \text{PLTMove} \neq \text{unAuthorized} \wedge$
 $\text{PLTMove} \neq \text{unAuthenticated} \wedge \text{PLTSecureHalt} \neq \text{Complete} \wedge$
 $\text{PLTSecureHalt} \neq \text{unAuthorized} \wedge$
 $\text{PLTSecureHalt} \neq \text{unAuthenticated} \wedge \text{Complete} \neq \text{unAuthorized} \wedge$
 $\text{Complete} \neq \text{unAuthenticated} \wedge \text{unAuthorized} \neq \text{unAuthenticated}$

[slState_distinct_clauses]

$\vdash \text{MOVE_TO_ORP} \neq \text{PLT_FORM} \wedge \text{MOVE_TO_ORP} \neq \text{PLT_MOVE} \wedge$
 $\text{MOVE_TO_ORP} \neq \text{PLT_SECURE_HALT} \wedge \text{MOVE_TO_ORP} \neq \text{COMPLETE} \wedge$
 $\text{PLT_FORM} \neq \text{PLT_MOVE} \wedge \text{PLT_FORM} \neq \text{PLT_SECURE_HALT} \wedge$
 $\text{PLT_FORM} \neq \text{COMPLETE} \wedge \text{PLT_MOVE} \neq \text{PLT_SECURE_HALT} \wedge$
 $\text{PLT_MOVE} \neq \text{COMPLETE} \wedge \text{PLT_SECURE_HALT} \neq \text{COMPLETE}$

14 ssmMoveToPB Theory

Built: 13 May 2018

Parent Theories: MoveToPBType, ssm11, OMNIType

14.1 Definitions

[[secContextMoveToPB_def](#)]

```
⊢ ∀ cmd.
  secContextMoveToPB cmd =
    [Name PlatoonLeader controls prop (SOME (SLc cmd))]
```

[[ssmMoveToPBStateInterp_def](#)]

```
⊢ ∀ state. ssmMoveToPBStateInterp state = TT
```

14.2 Theorems

[[authTestMoveToPB_cmd_reject_lemma](#)]

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

[[authTestMoveToPB_def](#)]

```
⊢ (authTestMoveToPB (Name PlatoonLeader says prop cmd) ⇔ T) ∧
  (authTestMoveToPB TT ⇔ F) ∧ (authTestMoveToPB FF ⇔ F) ∧
  (authTestMoveToPB (prop v) ⇔ F) ∧
  (authTestMoveToPB (notf v1) ⇔ F) ∧
  (authTestMoveToPB (v2 andf v3) ⇔ F) ∧
  (authTestMoveToPB (v4 orf v5) ⇔ F) ∧
  (authTestMoveToPB (v6 impf v7) ⇔ F) ∧
  (authTestMoveToPB (v8 eqf v9) ⇔ F) ∧
  (authTestMoveToPB (v10 says TT) ⇔ F) ∧
  (authTestMoveToPB (v10 says FF) ⇔ F) ∧
  (authTestMoveToPB (v133 meet v134 says prop v66) ⇔ F) ∧
  (authTestMoveToPB (v135 quoting v136 says prop v66) ⇔ F) ∧
  (authTestMoveToPB (v10 says notf v67) ⇔ F) ∧
  (authTestMoveToPB (v10 says (v68 andf v69)) ⇔ F) ∧
  (authTestMoveToPB (v10 says (v70 orf v71)) ⇔ F) ∧
  (authTestMoveToPB (v10 says (v72 impf v73)) ⇔ F) ∧
  (authTestMoveToPB (v10 says (v74 eqf v75)) ⇔ F) ∧
  (authTestMoveToPB (v10 says v76 says v77) ⇔ F) ∧
  (authTestMoveToPB (v10 says v78 speaks_for v79) ⇔ F) ∧
  (authTestMoveToPB (v10 says v80 controls v81) ⇔ F) ∧
  (authTestMoveToPB (v10 says reps v82 v83 v84) ⇔ F) ∧
  (authTestMoveToPB (v10 says v85 domi v86) ⇔ F) ∧
  (authTestMoveToPB (v10 says v87 eqi v88) ⇔ F) ∧
  (authTestMoveToPB (v10 says v89 doms v90) ⇔ F) ∧
  (authTestMoveToPB (v10 says v91 eqs v92) ⇔ F) ∧
  (authTestMoveToPB (v10 says v93 eqn v94) ⇔ F) ∧
```

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

[authTestMoveToPB_ind]

$\vdash \forall P.$

$(\forall \text{cmd}. P (\text{Name PlatoonLeader says prop cmd})) \wedge P \text{ TT} \wedge$
 $P \text{ FF} \wedge (\forall v. P (\text{prop } v)) \wedge (\forall v_1. P (\text{notf } v_1)) \wedge$
 $(\forall v_2 \ v_3. P (v_2 \text{ andf } v_3)) \wedge (\forall v_4 \ v_5. P (v_4 \text{ orf } v_5)) \wedge$
 $(\forall v_6 \ v_7. P (v_6 \text{ impf } v_7)) \wedge (\forall v_8 \ v_9. P (v_8 \text{ eqf } v_9)) \wedge$
 $(\forall v_{10}. P (v_{10} \text{ says TT})) \wedge (\forall v_{10}. P (v_{10} \text{ says FF})) \wedge$
 $(\forall v_{133} \ v_{134} \ v_{66}. P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66})) \wedge$
 $(\forall v_{135} \ v_{136} \ v_{66}. P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66})) \wedge$
 $(\forall v_{10} \ v_{67}. P (v_{10} \text{ says notf } v_{67})) \wedge$
 $(\forall v_{10} \ v_{68} \ v_{69}. P (v_{10} \text{ says } (v_{68} \text{ andf } v_{69}))) \wedge$
 $(\forall v_{10} \ v_{70} \ v_{71}. P (v_{10} \text{ says } (v_{70} \text{ orf } v_{71}))) \wedge$
 $(\forall v_{10} \ v_{72} \ v_{73}. P (v_{10} \text{ says } (v_{72} \text{ impf } v_{73}))) \wedge$
 $(\forall v_{10} \ v_{74} \ v_{75}. P (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75}))) \wedge$
 $(\forall v_{10} \ v_{76} \ v_{77}. P (v_{10} \text{ says } v_{76} \text{ says } v_{77})) \wedge$
 $(\forall v_{10} \ v_{78} \ v_{79}. P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79})) \wedge$
 $(\forall v_{10} \ v_{80} \ v_{81}. P (v_{10} \text{ says } v_{80} \text{ controls } v_{81})) \wedge$
 $(\forall v_{10} \ v_{82} \ v_{83} \ v_{84}. P (v_{10} \text{ says reps } v_{82} \ v_{83} \ v_{84})) \wedge$
 $(\forall v_{10} \ v_{85} \ v_{86}. P (v_{10} \text{ says } v_{85} \text{ domi } v_{86})) \wedge$
 $(\forall v_{10} \ v_{87} \ v_{88}. P (v_{10} \text{ says } v_{87} \text{ eqi } v_{88})) \wedge$
 $(\forall v_{10} \ v_{89} \ v_{90}. P (v_{10} \text{ says } v_{89} \text{ doms } v_{90})) \wedge$
 $(\forall v_{10} \ v_{91} \ v_{92}. P (v_{10} \text{ says } v_{91} \text{ eqs } v_{92})) \wedge$
 $(\forall v_{10} \ v_{93} \ v_{94}. P (v_{10} \text{ says } v_{93} \text{ eqn } v_{94})) \wedge$
 $(\forall v_{10} \ v_{95} \ v_{96}. P (v_{10} \text{ says } v_{95} \text{ lte } v_{96})) \wedge$
 $(\forall v_{10} \ v_{97} \ v_{98}. P (v_{10} \text{ says } v_{97} \text{ lt } v_{98})) \wedge$
 $(\forall v_{12} \ v_{13}. P (v_{12} \text{ speaks_for } v_{13})) \wedge$
 $(\forall v_{14} \ v_{15}. P (v_{14} \text{ controls } v_{15})) \wedge$
 $(\forall v_{16} \ v_{17} \ v_{18}. P (\text{reps } v_{16} \ v_{17} \ v_{18})) \wedge$
 $(\forall v_{19} \ v_{20}. P (v_{19} \text{ domi } v_{20})) \wedge$
 $(\forall v_{21} \ v_{22}. P (v_{21} \text{ eqi } v_{22})) \wedge$
 $(\forall v_{23} \ v_{24}. P (v_{23} \text{ doms } v_{24})) \wedge$
 $(\forall v_{25} \ v_{26}. P (v_{25} \text{ eqs } v_{26})) \wedge (\forall v_{27} \ v_{28}. P (v_{27} \text{ eqn } v_{28})) \wedge$
 $(\forall v_{29} \ v_{30}. P (v_{29} \text{ lte } v_{30})) \wedge (\forall v_{31} \ v_{32}. P (v_{31} \text{ lt } v_{32})) \Rightarrow$
 $\forall v. P \ v$

[moveToPBNS_def]

$$\begin{aligned}
&\vdash (\text{moveToPBNS MOVE_TO_PB (exec (SLc pltForm))} = \text{PLT_FORM}) \wedge \\
&\quad (\text{moveToPBNS MOVE_TO_PB (exec (SLc incomplete))} = \\
&\quad \quad \text{MOVE_TO_PB}) \wedge \\
&\quad (\text{moveToPBNS PLT_FORM (exec (SLc pltMove))} = \text{PLT_MOVE}) \wedge \\
&\quad (\text{moveToPBNS PLT_FORM (exec (SLc incomplete))} = \text{PLT_FORM}) \wedge \\
&\quad (\text{moveToPBNS PLT_MOVE (exec (SLc pltHalt))} = \text{PLT_HALT}) \wedge \\
&\quad (\text{moveToPBNS PLT_MOVE (exec (SLc incomplete))} = \text{PLT_MOVE}) \wedge \\
&\quad (\text{moveToPBNS PLT_HALT (exec (SLc complete))} = \text{COMPLETE}) \wedge \\
&\quad (\text{moveToPBNS PLT_HALT (exec (SLc incomplete))} = \text{PLT_HALT}) \wedge \\
&\quad (\text{moveToPBNS } s \text{ (trap (SLc cmd))} = s) \wedge \\
&\quad (\text{moveToPBNS } s \text{ (discard (SLc cmd))} = s)
\end{aligned}$$

[moveToPBNS_ind]

$$\begin{aligned}
&\vdash \forall P. \\
&\quad P \text{ MOVE_TO_PB (exec (SLc pltForm))} \wedge \\
&\quad P \text{ MOVE_TO_PB (exec (SLc incomplete))} \wedge \\
&\quad P \text{ PLT_FORM (exec (SLc pltMove))} \wedge \\
&\quad P \text{ PLT_FORM (exec (SLc incomplete))} \wedge \\
&\quad P \text{ PLT_MOVE (exec (SLc pltHalt))} \wedge \\
&\quad P \text{ PLT_MOVE (exec (SLc incomplete))} \wedge \\
&\quad P \text{ PLT_HALT (exec (SLc complete))} \wedge \\
&\quad P \text{ PLT_HALT (exec (SLc incomplete))} \wedge \\
&\quad (\forall s \text{ cmd. } P \text{ } s \text{ (trap (SLc cmd))}) \wedge \\
&\quad (\forall s \text{ cmd. } P \text{ } s \text{ (discard (SLc cmd))}) \wedge \\
&\quad (\forall s \text{ } v_6. P \text{ } s \text{ (discard (ESCc } v_6))) \wedge \\
&\quad (\forall s \text{ } v_9. P \text{ } s \text{ (trap (ESCc } v_9))) \wedge \\
&\quad (\forall v_{12}. P \text{ MOVE_TO_PB (exec (ESCc } v_{12}))) \wedge \\
&\quad P \text{ MOVE_TO_PB (exec (SLc pltMove))} \wedge \\
&\quad P \text{ MOVE_TO_PB (exec (SLc pltHalt))} \wedge \\
&\quad P \text{ MOVE_TO_PB (exec (SLc complete))} \wedge \\
&\quad (\forall v_{15}. P \text{ PLT_FORM (exec (ESCc } v_{15}))) \wedge \\
&\quad P \text{ PLT_FORM (exec (SLc pltForm))} \wedge \\
&\quad P \text{ PLT_FORM (exec (SLc pltHalt))} \wedge \\
&\quad P \text{ PLT_FORM (exec (SLc complete))} \wedge \\
&\quad (\forall v_{18}. P \text{ PLT_MOVE (exec (ESCc } v_{18}))) \wedge \\
&\quad P \text{ PLT_MOVE (exec (SLc pltForm))} \wedge \\
&\quad P \text{ PLT_MOVE (exec (SLc pltMove))} \wedge \\
&\quad P \text{ PLT_MOVE (exec (SLc complete))} \wedge \\
&\quad (\forall v_{21}. P \text{ PLT_HALT (exec (ESCc } v_{21}))) \wedge \\
&\quad P \text{ PLT_HALT (exec (SLc pltForm))} \wedge \\
&\quad P \text{ PLT_HALT (exec (SLc pltMove))} \wedge \\
&\quad P \text{ PLT_HALT (exec (SLc pltHalt))} \wedge \\
&\quad (\forall v_{23}. P \text{ COMPLETE (exec } v_{23})) \Rightarrow \\
&\quad \forall v \text{ } v_1. P \text{ } v \text{ } v_1
\end{aligned}$$

[moveToPBOut_def]

$$\begin{aligned}
&\vdash (\text{moveToPBOut MOVE_TO_PB (exec (SLc pltForm))} = \text{PLTForm}) \wedge \\
&\quad (\text{moveToPBOut MOVE_TO_PB (exec (SLc incomplete))} = \text{MoveToPB}) \wedge \\
&\quad (\text{moveToPBOut PLT_FORM (exec (SLc pltMove))} = \text{PLTMove}) \wedge
\end{aligned}$$

$(\text{moveToPBOut PLT_FORM (exec (SLc incomplete))} = \text{PLTForm}) \wedge$
 $(\text{moveToPBOut PLT_MOVE (exec (SLc pltHalt))} = \text{PLTHalt}) \wedge$
 $(\text{moveToPBOut PLT_MOVE (exec (SLc incomplete))} = \text{PLTMove}) \wedge$
 $(\text{moveToPBOut PLT_HALT (exec (SLc complete))} = \text{Complete}) \wedge$
 $(\text{moveToPBOut PLT_HALT (exec (SLc incomplete))} = \text{PLTHalt}) \wedge$
 $(\text{moveToPBOut } s \text{ (trap (SLc cmd))} = \text{unAuthorized}) \wedge$
 $(\text{moveToPBOut } s \text{ (discard (SLc cmd))} = \text{unAuthenticated})$

[moveToPBOut_ind]

$\vdash \forall P.$
 $P \text{ MOVE_TO_PB (exec (SLc pltForm))} \wedge$
 $P \text{ MOVE_TO_PB (exec (SLc incomplete))} \wedge$
 $P \text{ PLT_FORM (exec (SLc pltMove))} \wedge$
 $P \text{ PLT_FORM (exec (SLc incomplete))} \wedge$
 $P \text{ PLT_MOVE (exec (SLc pltHalt))} \wedge$
 $P \text{ PLT_MOVE (exec (SLc incomplete))} \wedge$
 $P \text{ PLT_HALT (exec (SLc complete))} \wedge$
 $P \text{ PLT_HALT (exec (SLc incomplete))} \wedge$
 $(\forall s \text{ cmd. } P \text{ } s \text{ (trap (SLc cmd))}) \wedge$
 $(\forall s \text{ cmd. } P \text{ } s \text{ (discard (SLc cmd))}) \wedge$
 $(\forall s \text{ } v_6. P \text{ } s \text{ (discard (ESCc } v_6))) \wedge$
 $(\forall s \text{ } v_9. P \text{ } s \text{ (trap (ESCc } v_9))) \wedge$
 $(\forall v_{12}. P \text{ MOVE_TO_PB (exec (ESCc } v_{12}))) \wedge$
 $P \text{ MOVE_TO_PB (exec (SLc pltMove))} \wedge$
 $P \text{ MOVE_TO_PB (exec (SLc pltHalt))} \wedge$
 $P \text{ MOVE_TO_PB (exec (SLc complete))} \wedge$
 $(\forall v_{15}. P \text{ PLT_FORM (exec (ESCc } v_{15}))) \wedge$
 $P \text{ PLT_FORM (exec (SLc pltForm))} \wedge$
 $P \text{ PLT_FORM (exec (SLc pltHalt))} \wedge$
 $P \text{ PLT_FORM (exec (SLc complete))} \wedge$
 $(\forall v_{18}. P \text{ PLT_MOVE (exec (ESCc } v_{18}))) \wedge$
 $P \text{ PLT_MOVE (exec (SLc pltForm))} \wedge$
 $P \text{ PLT_MOVE (exec (SLc pltMove))} \wedge$
 $P \text{ PLT_MOVE (exec (SLc complete))} \wedge$
 $(\forall v_{21}. P \text{ PLT_HALT (exec (ESCc } v_{21}))) \wedge$
 $P \text{ PLT_HALT (exec (SLc pltForm))} \wedge$
 $P \text{ PLT_HALT (exec (SLc pltMove))} \wedge$
 $P \text{ PLT_HALT (exec (SLc pltHalt))} \wedge$
 $(\forall v_{23}. P \text{ COMPLETE (exec } v_{23})) \Rightarrow$
 $\forall v \text{ } v_1. P \text{ } v \text{ } v_1$

[PlatoonLeader_exec_slCommand_justified_thm]

$\vdash \forall NS \text{ Out } M \text{ } O_i \text{ } O_s.$
 $\text{TR } (M, O_i, O_s) \text{ (exec (SLc slCommand))}$
 $(\text{CFG authTestMoveToPB ssmMoveToPBStateInterp}$
 $\quad (\text{secContextMoveToPB slCommand})$
 $\quad (\text{Name PlatoonLeader says prop (SOME (SLc slCommand))} ::$
 $\quad \quad \text{ins) } s \text{ outs})$
 $\quad (\text{CFG authTestMoveToPB ssmMoveToPBStateInterp}$

```

        (secContextMoveToPB slCommand) ins
        (NS s (exec (SLc slCommand)))
        (Out s (exec (SLc slCommand))::outs))  $\iff$ 
authTestMoveToPB
  (Name PlatoonLeader says prop (SOME (SLc slCommand)))  $\wedge$ 
CFGInterpret (M, Oi, Os)
  (CFG authTestMoveToPB ssmMoveToPBStateInterp
    (secContextMoveToPB slCommand)
    (Name PlatoonLeader says prop (SOME (SLc slCommand))::
      ins) s outs)  $\wedge$ 
  (M, Oi, Os) sat prop (SOME (SLc slCommand))

```

[PlatoonLeader_slCommand_lemma]

```

 $\vdash$  CFGInterpret (M, Oi, Os)
  (CFG authTestMoveToPB ssmMoveToPBStateInterp
    (secContextMoveToPB slCommand)
    (Name PlatoonLeader says prop (SOME (SLc slCommand))::
      ins) s outs)  $\Rightarrow$ 
  (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]

```

 $\vdash$  pltForm  $\neq$  pltMove  $\wedge$  pltForm  $\neq$  pltHalt  $\wedge$  pltForm  $\neq$  complete  $\wedge$ 
  pltForm  $\neq$  incomplete  $\wedge$  pltMove  $\neq$  pltHalt  $\wedge$ 
  pltMove  $\neq$  complete  $\wedge$  pltMove  $\neq$  incomplete  $\wedge$ 
  pltHalt  $\neq$  complete  $\wedge$  pltHalt  $\neq$  incomplete  $\wedge$ 
  complete  $\neq$  incomplete

```

[slOutput_distinct_clauses]

$\vdash \text{MoveToPB} \neq \text{PLTForm} \wedge \text{MoveToPB} \neq \text{PLTMove} \wedge$
 $\text{MoveToPB} \neq \text{PLTHalt} \wedge \text{MoveToPB} \neq \text{Complete} \wedge$
 $\text{MoveToPB} \neq \text{unAuthorized} \wedge \text{MoveToPB} \neq \text{unAuthenticated} \wedge$
 $\text{PLTForm} \neq \text{PLTMove} \wedge \text{PLTForm} \neq \text{PLTHalt} \wedge \text{PLTForm} \neq \text{Complete} \wedge$
 $\text{PLTForm} \neq \text{unAuthorized} \wedge \text{PLTForm} \neq \text{unAuthenticated} \wedge$
 $\text{PLTMove} \neq \text{PLTHalt} \wedge \text{PLTMove} \neq \text{Complete} \wedge$
 $\text{PLTMove} \neq \text{unAuthorized} \wedge \text{PLTMove} \neq \text{unAuthenticated} \wedge$
 $\text{PLTHalt} \neq \text{Complete} \wedge \text{PLTHalt} \neq \text{unAuthorized} \wedge$
 $\text{PLTHalt} \neq \text{unAuthenticated} \wedge \text{Complete} \neq \text{unAuthorized} \wedge$
 $\text{Complete} \neq \text{unAuthenticated} \wedge \text{unAuthorized} \neq \text{unAuthenticated}$

[slState_distinct_clauses]

$\vdash \text{MOVE_TO_PB} \neq \text{PLT_FORM} \wedge \text{MOVE_TO_PB} \neq \text{PLT_MOVE} \wedge$
 $\text{MOVE_TO_PB} \neq \text{PLT_HALT} \wedge \text{MOVE_TO_PB} \neq \text{COMPLETE} \wedge$
 $\text{PLT_FORM} \neq \text{PLT_MOVE} \wedge \text{PLT_FORM} \neq \text{PLT_HALT} \wedge$
 $\text{PLT_FORM} \neq \text{COMPLETE} \wedge \text{PLT_MOVE} \neq \text{PLT_HALT} \wedge$
 $\text{PLT_MOVE} \neq \text{COMPLETE} \wedge \text{PLT_HALT} \neq \text{COMPLETE}$

16 ssmPlanPB Theory

Built: 13 May 2018

Parent Theories: PlanPBDef, ssm

16.1 Theorems

[inputOK_def]

$\vdash (\text{inputOK} (\text{Name PlatoonLeader says prop } cmd) \iff T) \wedge$
 $(\text{inputOK} (\text{Name PlatoonSergeant says prop } cmd) \iff T) \wedge$
 $(\text{inputOK } TT \iff F) \wedge (\text{inputOK } FF \iff F) \wedge$
 $(\text{inputOK} (\text{prop } v) \iff F) \wedge (\text{inputOK} (\text{notf } v_1) \iff F) \wedge$
 $(\text{inputOK} (v_2 \text{ andf } v_3) \iff F) \wedge (\text{inputOK} (v_4 \text{ orf } v_5) \iff F) \wedge$
 $(\text{inputOK} (v_6 \text{ impf } v_7) \iff F) \wedge (\text{inputOK} (v_8 \text{ eqf } v_9) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } TT) \iff F) \wedge (\text{inputOK} (v_{10} \text{ says } FF) \iff F) \wedge$
 $(\text{inputOK} (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66}) \iff F) \wedge$
 $(\text{inputOK} (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66}) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says notf } v_{67}) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } (v_{68} \text{ andf } v_{69})) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } (v_{70} \text{ orf } v_{71})) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } (v_{72} \text{ impf } v_{73})) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75})) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } v_{76} \text{ says } v_{77}) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79}) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } v_{80} \text{ controls } v_{81}) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says reps } v_{82} \ v_{83} \ v_{84}) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } v_{85} \text{ domi } v_{86}) \iff F) \wedge$
 $(\text{inputOK} (v_{10} \text{ says } v_{87} \text{ eqi } v_{88}) \iff F) \wedge$

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

[inputOK_ind]

$\vdash \forall P.$

$(\forall \text{cmd}. P (\text{Name PlatoonLeader says prop cmd})) \wedge$
 $(\forall \text{cmd}. P (\text{Name PlatoonSergeant says prop cmd})) \wedge P \text{ TT} \wedge$
 $P \text{ FF} \wedge (\forall v. P (\text{prop } v)) \wedge (\forall v_1. P (\text{notf } v_1)) \wedge$
 $(\forall v_2 \ v_3. P (v_2 \text{ andf } v_3)) \wedge (\forall v_4 \ v_5. P (v_4 \text{ orf } v_5)) \wedge$
 $(\forall v_6 \ v_7. P (v_6 \text{ impf } v_7)) \wedge (\forall v_8 \ v_9. P (v_8 \text{ eqf } v_9)) \wedge$
 $(\forall v_{10}. P (v_{10} \text{ says TT})) \wedge (\forall v_{10}. P (v_{10} \text{ says FF})) \wedge$
 $(\forall v_{133} \ v_{134} \ v_{66}. P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66})) \wedge$
 $(\forall v_{135} \ v_{136} \ v_{66}. P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66})) \wedge$
 $(\forall v_{10} \ v_{67}. P (v_{10} \text{ says notf } v_{67})) \wedge$
 $(\forall v_{10} \ v_{68} \ v_{69}. P (v_{10} \text{ says } (v_{68} \text{ andf } v_{69}))) \wedge$
 $(\forall v_{10} \ v_{70} \ v_{71}. P (v_{10} \text{ says } (v_{70} \text{ orf } v_{71}))) \wedge$
 $(\forall v_{10} \ v_{72} \ v_{73}. P (v_{10} \text{ says } (v_{72} \text{ impf } v_{73}))) \wedge$
 $(\forall v_{10} \ v_{74} \ v_{75}. P (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75}))) \wedge$
 $(\forall v_{10} \ v_{76} \ v_{77}. P (v_{10} \text{ says } v_{76} \text{ says } v_{77})) \wedge$
 $(\forall v_{10} \ v_{78} \ v_{79}. P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79})) \wedge$
 $(\forall v_{10} \ v_{80} \ v_{81}. P (v_{10} \text{ says } v_{80} \text{ controls } v_{81})) \wedge$
 $(\forall v_{10} \ v_{82} \ v_{83} \ v_{84}. P (v_{10} \text{ says reps } v_{82} \ v_{83} \ v_{84})) \wedge$
 $(\forall v_{10} \ v_{85} \ v_{86}. P (v_{10} \text{ says } v_{85} \text{ domi } v_{86})) \wedge$
 $(\forall v_{10} \ v_{87} \ v_{88}. P (v_{10} \text{ says } v_{87} \text{ eqi } v_{88})) \wedge$
 $(\forall v_{10} \ v_{89} \ v_{90}. P (v_{10} \text{ says } v_{89} \text{ doms } v_{90})) \wedge$
 $(\forall v_{10} \ v_{91} \ v_{92}. P (v_{10} \text{ says } v_{91} \text{ eqs } v_{92})) \wedge$
 $(\forall v_{10} \ v_{93} \ v_{94}. P (v_{10} \text{ says } v_{93} \text{ eqn } v_{94})) \wedge$
 $(\forall v_{10} \ v_{95} \ v_{96}. P (v_{10} \text{ says } v_{95} \text{ lte } v_{96})) \wedge$
 $(\forall v_{10} \ v_{97} \ v_{98}. P (v_{10} \text{ says } v_{97} \text{ lt } v_{98})) \wedge$
 $(\forall v_{12} \ v_{13}. P (v_{12} \text{ speaks_for } v_{13})) \wedge$
 $(\forall v_{14} \ v_{15}. P (v_{14} \text{ controls } v_{15})) \wedge$
 $(\forall v_{16} \ v_{17} \ v_{18}. P (\text{reps } v_{16} \ v_{17} \ v_{18})) \wedge$
 $(\forall v_{19} \ v_{20}. P (v_{19} \text{ domi } v_{20})) \wedge$
 $(\forall v_{21} \ v_{22}. P (v_{21} \text{ eqi } v_{22})) \wedge$
 $(\forall v_{23} \ v_{24}. P (v_{23} \text{ doms } v_{24})) \wedge$
 $(\forall v_{25} \ v_{26}. P (v_{25} \text{ eqs } v_{26})) \wedge (\forall v_{27} \ v_{28}. P (v_{27} \text{ eqn } v_{28})) \wedge$
 $(\forall v_{29} \ v_{30}. P (v_{29} \text{ lte } v_{30})) \wedge (\forall v_{31} \ v_{32}. P (v_{31} \text{ lt } v_{32})) \Rightarrow$
 $\forall v. P \ v$

[planPBNS_def]

```

⊢ (planPBNS WARN0 (exec x) =
  if
    (getRecon x = [SOME (SLc (PL recon))]) ∧
    (getTentativePlan x = [SOME (SLc (PL tentativePlan))]) ∧
    (getReport x = [SOME (SLc (PL report1))]) ∧
    (getInitMove x = [SOME (SLc (PSG initiateMovement))])
  then
    REPORT1
  else WARN0) ∧
(planPBNS PLAN_PB (exec x) =
  if getPlCom x = receiveMission then RECEIVE_MISSION
  else PLAN_PB) ∧
(planPBNS RECEIVE_MISSION (exec x) =
  if getPlCom x = warn0 then WARN0 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 WARN0 (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 WARN0 (exec x) =
  if
    (getRecon x = [SOME (SLc (PL recon))]) ∧
    (getTentativePlan x = [SOME (SLc (PL tentativePlan))]) ∧
    (getReport x = [SOME (SLc (PL report1))]) ∧
    (getInitMove x = [SOME (SLc (PSG initiateMovement))])
  then
    REPORT1
  else WARN0)

```



```

then
  Report1
  else unauthorized)  $\wedge$ 
(planPBOut PLAN_PB (exec  $x$ ) =
  if getPlCom  $x$  = receiveMission then ReceiveMission
  else unauthorized)  $\wedge$ 
(planPBOut RECEIVE_MISSION (exec  $x$ ) =
  if getPlCom  $x$  = warno then Warno else unauthorized)  $\wedge$ 
(planPBOut REPORT1 (exec  $x$ ) =
  if getPlCom  $x$  = completePlan then CompletePlan
  else unauthorized)  $\wedge$ 
(planPBOut COMPLETE_PLAN (exec  $x$ ) =
  if getPlCom  $x$  = opoid then Opoid else unauthorized)  $\wedge$ 
(planPBOut OPOID (exec  $x$ ) =
  if getPlCom  $x$  = supervise then Supervise
  else unauthorized)  $\wedge$ 
(planPBOut SUPERVISE (exec  $x$ ) =
  if getPlCom  $x$  = report2 then Report2 else unauthorized)  $\wedge$ 
(planPBOut REPORT2 (exec  $x$ ) =
  if getPlCom  $x$  = complete then Complete else unauthorized)  $\wedge$ 
(planPBOut  $s$  (trap  $v_0$ ) = unauthorized)  $\wedge$ 
(planPBOut  $s$  (discard  $v_1$ ) = unAuthenticated)

```

[planPBOut_ind]

```

 $\vdash \forall P.$ 
  ( $\forall x. P \text{ WARNO (exec } x)$ )  $\wedge$  ( $\forall x. P \text{ PLAN\_PB (exec } x)$ )  $\wedge$ 
  ( $\forall x. P \text{ RECEIVE\_MISSION (exec } x)$ )  $\wedge$ 
  ( $\forall x. P \text{ REPORT1 (exec } x)$ )  $\wedge$  ( $\forall x. P \text{ COMPLETE\_PLAN (exec } x)$ )  $\wedge$ 
  ( $\forall x. P \text{ OPOID (exec } x)$ )  $\wedge$  ( $\forall x. P \text{ SUPERVISE (exec } x)$ )  $\wedge$ 
  ( $\forall x. P \text{ REPORT2 (exec } x)$ )  $\wedge$  ( $\forall s \ v_0. P \ s \ (\text{trap } v_0)$ )  $\wedge$ 
  ( $\forall s \ v_1. P \ s \ (\text{discard } v_1)$ )  $\wedge$ 
  ( $\forall v_6. P \text{ TENTATIVE\_PLAN (exec } v_6)$ )  $\wedge$ 
  ( $\forall v_7. P \text{ INITIATE\_MOVEMENT (exec } v_7)$ )  $\wedge$ 
  ( $\forall v_8. P \text{ RECON (exec } v_8)$ )  $\wedge$  ( $\forall v_9. P \text{ COMPLETE (exec } v_9)$ )  $\Rightarrow$ 
   $\forall v \ v_1. P \ v \ v_1$ 

```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma]

```

 $\vdash s \neq \text{WARNO} \Rightarrow$ 
   $plCommand \neq \text{invalidPlCommand} \Rightarrow$ 
   $plCommand \neq \text{report1} \Rightarrow$ 
   $\forall NS \ Out \ M \ Oi \ Os.$ 
    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))  $\iff$ 
authenticationTest inputOK
  [Name PlatoonLeader says
    prop (SOME (SLc (PL plCommand)))]  $\wedge$ 
CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secContextNull
    ([Name PlatoonLeader says
      prop (SOME (SLc (PL plCommand)))]::ins) s outs)  $\wedge$ 
  (M, Oi, Os) satList
  propCommandList
  [Name PlatoonLeader says
    prop (SOME (SLc (PL plCommand)))]

```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]

```

 $\vdash s \neq \text{WARNO} \Rightarrow$ 
  plCommand  $\neq$  invalidPlCommand  $\Rightarrow$ 
  plCommand  $\neq$  report1  $\Rightarrow$ 
 $\forall NS \text{ Out } M \text{ Oi } Os.$ 
  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))  $\iff$ 
authenticationTest inputOK
  [Name PlatoonLeader says
    prop (SOME (SLc (PL plCommand)))]  $\wedge$ 
CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secContextNull
    ([Name PlatoonLeader says
      prop (SOME (SLc (PL plCommand)))]::ins) s outs)  $\wedge$ 
  (M, Oi, Os) satList [prop (SOME (SLc (PL plCommand)))]

```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma]

```

 $\vdash s \neq \text{WARNO} \Rightarrow$ 
  plCommand  $\neq$  invalidPlCommand  $\Rightarrow$ 
  plCommand  $\neq$  report1  $\Rightarrow$ 

```

$\forall M \ O_i \ O_s.$
 CFGInterpret (M, O_i, O_s)
 (CFG inputOK secContext secContextNull
 ([Name PlatoonLeader says
 prop (SOME (SLc (PL $plCommand$))))] $::ins$) s $outs$) \Rightarrow
 (M, O_i, O_s) satList
 propCommandList
 [Name PlatoonLeader says
 prop (SOME (SLc (PL $plCommand$)))]

[PlatoonLeader_psgCommand_notDiscard_thm]

$\vdash \forall NS \ Out \ M \ O_i \ O_s.$
 $\neg TR \ (M, O_i, O_s)$
 (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]

$\vdash \forall NS \ Out \ M \ O_i \ O_s.$
 $TR \ (M, O_i, O_s)$
 (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))  $\iff$ 
authenticationTest inputOK
  [Name PlatoonLeader says
    prop (SOME (SLc (PSG psgCommand)))]  $\wedge$ 
CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secContextNull
    ([Name PlatoonLeader says
      prop (SOME (SLc (PSG psgCommand)))]::ins) s outs)  $\wedge$ 
(M, Oi, Os) sat prop NONE

```

[PlatoonLeader_trap_psgCommand_lemma]

```

 $\vdash \forall M \text{ } Oi \text{ } Os.$ 
CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secContextNull
    ([Name PlatoonLeader says
      prop (SOME (SLc (PSG psgCommand)))]::ins) s outs)  $\Rightarrow$ 
(M, Oi, Os) sat prop NONE

```

[PlatoonLeader_WARNO_exec_report1_justified_lemma]

```

 $\vdash \forall NS \text{ } Out \text{ } M \text{ } Oi \text{ } Os.$ 
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))  $\iff$ 
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)))]  $\wedge$ 
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)  $\wedge$ 
(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]

$\vdash \forall NS \text{ Out } M \text{ Oi } Os.$
 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) WARNNO outs)
(NS WARNNO
  (exec
    [SOME (SLc (PL recon));
     SOME (SLc (PL tentativePlan));
     SOME (SLc (PSG initiateMovement));
     SOME (SLc (PL report1))])
  (Out WARNNO
    (exec
      [SOME (SLc (PL recon));
       SOME (SLc (PL tentativePlan));
       SOME (SLc (PSG initiateMovement));
       SOME (SLc (PL report1)))]::outs))  $\iff$ 
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)))]  $\wedge$ 
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) WARNNO outs)  $\wedge$ 
  (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]

$\vdash \forall M \ O_i \ O_s.$
 CFGInterpret (M, O_i, O_s)
 (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) \Rightarrow
 (M, O_i, O_s) 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]

$\vdash \forall NS \ Out \ M \ O_i \ O_s.$
 TR (M, O_i, O_s)
 (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)) \iff
 authenticationTest inputOK
 [Name PlatoonSergeant says
 prop (SOME (SLc (PL plCommand)))] \wedge
 CFGInterpret (M, O_i, O_s)
 (CFG inputOK secContext secContextNull
 ([Name PlatoonSergeant says

prop (SOME (SLc (PL *plCommand*))))]::ins) *s outs*) \wedge
 (*M, Oi, Os*) sat prop NONE

[PlatoonSergeant_trap_plCommand_justified_thm]

$\vdash \forall NS \text{ Out } M \text{ Oi } Os.$
 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)) \iff
 authenticationTest inputOK
 [Name PlatoonSergeant says
 prop (SOME (SLc (PL *plCommand*)))] \wedge
 CFGInterpret (*M, Oi, Os*)
 (CFG inputOK secContext secContextNull
 ([Name PlatoonSergeant says
 prop (SOME (SLc (PL *plCommand*))))]::ins) *s outs*) \wedge
 (*M, Oi, Os*) sat prop NONE

[PlatoonSergeant_trap_plCommand_lemma]

$\vdash \forall M \text{ Oi } Os.$
 CFGInterpret (*M, Oi, Os*)
 (CFG inputOK secContext secContextNull
 ([Name PlatoonSergeant says
 prop (SOME (SLc (PL *plCommand*))))]::ins) *s outs*) \Rightarrow
 (*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

$slState = \text{PLAN_PB} \mid \text{RECEIVE_MISSION} \mid \text{WARNO} \mid \text{TENTATIVE_PLAN}$
 $\mid \text{INITIATE_MOVEMENT} \mid \text{RECON} \mid \text{REPORT1} \mid \text{COMPLETE_PLAN}$
 $\mid \text{OPOID} \mid \text{SUPERVISE} \mid \text{REPORT2} \mid \text{COMPLETE}$

$stateRole = \text{PlatoonLeader} \mid \text{PlatoonSergeant}$

17.2 Theorems

[plCommand_distinct_clauses]

$\vdash \text{receiveMission} \neq \text{warno} \wedge \text{receiveMission} \neq \text{tentativePlan} \wedge$
 $\text{receiveMission} \neq \text{recon} \wedge \text{receiveMission} \neq \text{report1} \wedge$
 $\text{receiveMission} \neq \text{completePlan} \wedge \text{receiveMission} \neq \text{opoid} \wedge$
 $\text{receiveMission} \neq \text{supervise} \wedge \text{receiveMission} \neq \text{report2} \wedge$
 $\text{receiveMission} \neq \text{complete} \wedge \text{receiveMission} \neq \text{plIncomplete} \wedge$
 $\text{receiveMission} \neq \text{invalidPlCommand} \wedge \text{warno} \neq \text{tentativePlan} \wedge$
 $\text{warno} \neq \text{recon} \wedge \text{warno} \neq \text{report1} \wedge \text{warno} \neq \text{completePlan} \wedge$
 $\text{warno} \neq \text{opoid} \wedge \text{warno} \neq \text{supervise} \wedge \text{warno} \neq \text{report2} \wedge$
 $\text{warno} \neq \text{complete} \wedge \text{warno} \neq \text{plIncomplete} \wedge$
 $\text{warno} \neq \text{invalidPlCommand} \wedge \text{tentativePlan} \neq \text{recon} \wedge$
 $\text{tentativePlan} \neq \text{report1} \wedge \text{tentativePlan} \neq \text{completePlan} \wedge$
 $\text{tentativePlan} \neq \text{opoid} \wedge \text{tentativePlan} \neq \text{supervise} \wedge$
 $\text{tentativePlan} \neq \text{report2} \wedge \text{tentativePlan} \neq \text{complete} \wedge$
 $\text{tentativePlan} \neq \text{plIncomplete} \wedge$
 $\text{tentativePlan} \neq \text{invalidPlCommand} \wedge \text{recon} \neq \text{report1} \wedge$
 $\text{recon} \neq \text{completePlan} \wedge \text{recon} \neq \text{opoid} \wedge \text{recon} \neq \text{supervise} \wedge$
 $\text{recon} \neq \text{report2} \wedge \text{recon} \neq \text{complete} \wedge \text{recon} \neq \text{plIncomplete} \wedge$
 $\text{recon} \neq \text{invalidPlCommand} \wedge \text{report1} \neq \text{completePlan} \wedge$
 $\text{report1} \neq \text{opoid} \wedge \text{report1} \neq \text{supervise} \wedge \text{report1} \neq \text{report2} \wedge$
 $\text{report1} \neq \text{complete} \wedge \text{report1} \neq \text{plIncomplete} \wedge$
 $\text{report1} \neq \text{invalidPlCommand} \wedge \text{completePlan} \neq \text{opoid} \wedge$
 $\text{completePlan} \neq \text{supervise} \wedge \text{completePlan} \neq \text{report2} \wedge$
 $\text{completePlan} \neq \text{complete} \wedge \text{completePlan} \neq \text{plIncomplete} \wedge$
 $\text{completePlan} \neq \text{invalidPlCommand} \wedge \text{opoid} \neq \text{supervise} \wedge$
 $\text{opoid} \neq \text{report2} \wedge \text{opoid} \neq \text{complete} \wedge \text{opoid} \neq \text{plIncomplete} \wedge$
 $\text{opoid} \neq \text{invalidPlCommand} \wedge \text{supervise} \neq \text{report2} \wedge$
 $\text{supervise} \neq \text{complete} \wedge \text{supervise} \neq \text{plIncomplete} \wedge$
 $\text{supervise} \neq \text{invalidPlCommand} \wedge \text{report2} \neq \text{complete} \wedge$
 $\text{report2} \neq \text{plIncomplete} \wedge \text{report2} \neq \text{invalidPlCommand} \wedge$
 $\text{complete} \neq \text{plIncomplete} \wedge \text{complete} \neq \text{invalidPlCommand} \wedge$
 $\text{plIncomplete} \neq \text{invalidPlCommand}$

[psgCommand_distinct_clauses]

$\vdash \text{initiateMovement} \neq \text{psgIncomplete} \wedge$
 $\text{initiateMovement} \neq \text{invalidPsgCommand} \wedge$
 $\text{psgIncomplete} \neq \text{invalidPsgCommand}$

[slCommand_distinct_clauses]

$\vdash \forall a' a. \text{PL } a \neq \text{PSG } a'$

[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 \text{PlanPB} \neq \text{ReceiveMission} \wedge \text{PlanPB} \neq \text{Warno} \wedge \\ \text{PlanPB} \neq \text{TentativePlan} \wedge \text{PlanPB} \neq \text{InitiateMovement} \wedge \\ \text{PlanPB} \neq \text{Recon} \wedge \text{PlanPB} \neq \text{Report1} \wedge \text{PlanPB} \neq \text{CompletePlan} \wedge \\ \text{PlanPB} \neq \text{Opoid} \wedge \text{PlanPB} \neq \text{Supervise} \wedge \text{PlanPB} \neq \text{Report2} \wedge \\ \text{PlanPB} \neq \text{Complete} \wedge \text{PlanPB} \neq \text{unAuthenticated} \wedge \\ \text{PlanPB} \neq \text{unAuthorized} \wedge \text{ReceiveMission} \neq \text{Warno} \wedge \\ \text{ReceiveMission} \neq \text{TentativePlan} \wedge \\ \text{ReceiveMission} \neq \text{InitiateMovement} \wedge \text{ReceiveMission} \neq \text{Recon} \wedge \\ \text{ReceiveMission} \neq \text{Report1} \wedge \text{ReceiveMission} \neq \text{CompletePlan} \wedge \\ \text{ReceiveMission} \neq \text{Opoid} \wedge \text{ReceiveMission} \neq \text{Supervise} \wedge \\ \text{ReceiveMission} \neq \text{Report2} \wedge \text{ReceiveMission} \neq \text{Complete} \wedge \\ \text{ReceiveMission} \neq \text{unAuthenticated} \wedge \\ \text{ReceiveMission} \neq \text{unAuthorized} \wedge \text{Warno} \neq \text{TentativePlan} \wedge \\ \text{Warno} \neq \text{InitiateMovement} \wedge \text{Warno} \neq \text{Recon} \wedge \text{Warno} \neq \text{Report1} \wedge \\ \text{Warno} \neq \text{CompletePlan} \wedge \text{Warno} \neq \text{Opoid} \wedge \text{Warno} \neq \text{Supervise} \wedge \\ \text{Warno} \neq \text{Report2} \wedge \text{Warno} \neq \text{Complete} \wedge \\ \text{Warno} \neq \text{unAuthenticated} \wedge \text{Warno} \neq \text{unAuthorized} \wedge \\ \text{TentativePlan} \neq \text{InitiateMovement} \wedge \text{TentativePlan} \neq \text{Recon} \wedge \\ \text{TentativePlan} \neq \text{Report1} \wedge \text{TentativePlan} \neq \text{CompletePlan} \wedge \\ \text{TentativePlan} \neq \text{Opoid} \wedge \text{TentativePlan} \neq \text{Supervise} \wedge \\ \text{TentativePlan} \neq \text{Report2} \wedge \text{TentativePlan} \neq \text{Complete} \wedge \\ \text{TentativePlan} \neq \text{unAuthenticated} \wedge \\ \text{TentativePlan} \neq \text{unAuthorized} \wedge \text{InitiateMovement} \neq \text{Recon} \wedge \\ \text{InitiateMovement} \neq \text{Report1} \wedge \\ \text{InitiateMovement} \neq \text{CompletePlan} \wedge \text{InitiateMovement} \neq \text{Opoid} \wedge \\ \text{InitiateMovement} \neq \text{Supervise} \wedge \text{InitiateMovement} \neq \text{Report2} \wedge \\ \text{InitiateMovement} \neq \text{Complete} \wedge \\ \text{InitiateMovement} \neq \text{unAuthenticated} \wedge \\ \text{InitiateMovement} \neq \text{unAuthorized} \wedge \text{Recon} \neq \text{Report1} \wedge \\ \text{Recon} \neq \text{CompletePlan} \wedge \text{Recon} \neq \text{Opoid} \wedge \text{Recon} \neq \text{Supervise} \wedge \\ \text{Recon} \neq \text{Report2} \wedge \text{Recon} \neq \text{Complete} \wedge \\ \text{Recon} \neq \text{unAuthenticated} \wedge \text{Recon} \neq \text{unAuthorized} \wedge \\ \text{Report1} \neq \text{CompletePlan} \wedge \text{Report1} \neq \text{Opoid} \wedge \\ \text{Report1} \neq \text{Supervise} \wedge \text{Report1} \neq \text{Report2} \wedge \\ \text{Report1} \neq \text{Complete} \wedge \text{Report1} \neq \text{unAuthenticated} \wedge \\ \text{Report1} \neq \text{unAuthorized} \wedge \text{CompletePlan} \neq \text{Opoid} \wedge \\ \text{CompletePlan} \neq \text{Supervise} \wedge \text{CompletePlan} \neq \text{Report2} \wedge \\ \text{CompletePlan} \neq \text{Complete} \wedge \text{CompletePlan} \neq \text{unAuthenticated} \wedge \\ \text{CompletePlan} \neq \text{unAuthorized} \wedge \text{Opoid} \neq \text{Supervise} \wedge \\ \text{Opoid} \neq \text{Report2} \wedge \text{Opoid} \neq \text{Complete} \wedge \\ \text{Opoid} \neq \text{unAuthenticated} \wedge \text{Opoid} \neq \text{unAuthorized} \wedge \\ \text{Supervise} \neq \text{Report2} \wedge \text{Supervise} \neq \text{Complete} \wedge \\ \text{Supervise} \neq \text{unAuthenticated} \wedge \text{Supervise} \neq \text{unAuthorized} \wedge \\ \text{Report2} \neq \text{Complete} \wedge \text{Report2} \neq \text{unAuthenticated} \wedge$$

Report2 \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 PLAN_PB \neq RECEIVE_MISSION \wedge PLAN_PB \neq WARNO \wedge
 PLAN_PB \neq TENTATIVE_PLAN \wedge PLAN_PB \neq INITIATE_MOVEMENT \wedge
 PLAN_PB \neq RECON \wedge PLAN_PB \neq REPORT1 \wedge
 PLAN_PB \neq COMPLETE_PLAN \wedge PLAN_PB \neq OPOID \wedge
 PLAN_PB \neq SUPERVISE \wedge PLAN_PB \neq REPORT2 \wedge
 PLAN_PB \neq COMPLETE \wedge RECEIVE_MISSION \neq WARNO \wedge
 RECEIVE_MISSION \neq TENTATIVE_PLAN \wedge
 RECEIVE_MISSION \neq INITIATE_MOVEMENT \wedge
 RECEIVE_MISSION \neq RECON \wedge RECEIVE_MISSION \neq REPORT1 \wedge
 RECEIVE_MISSION \neq COMPLETE_PLAN \wedge RECEIVE_MISSION \neq OPOID \wedge
 RECEIVE_MISSION \neq SUPERVISE \wedge RECEIVE_MISSION \neq REPORT2 \wedge
 RECEIVE_MISSION \neq COMPLETE \wedge WARNO \neq TENTATIVE_PLAN \wedge
 WARNO \neq INITIATE_MOVEMENT \wedge WARNO \neq RECON \wedge WARNO \neq REPORT1 \wedge
 WARNO \neq COMPLETE_PLAN \wedge WARNO \neq OPOID \wedge WARNO \neq SUPERVISE \wedge
 WARNO \neq REPORT2 \wedge WARNO \neq COMPLETE \wedge
 TENTATIVE_PLAN \neq INITIATE_MOVEMENT \wedge TENTATIVE_PLAN \neq RECON \wedge
 TENTATIVE_PLAN \neq REPORT1 \wedge TENTATIVE_PLAN \neq COMPLETE_PLAN \wedge
 TENTATIVE_PLAN \neq OPOID \wedge TENTATIVE_PLAN \neq SUPERVISE \wedge
 TENTATIVE_PLAN \neq REPORT2 \wedge TENTATIVE_PLAN \neq COMPLETE \wedge
 INITIATE_MOVEMENT \neq RECON \wedge INITIATE_MOVEMENT \neq REPORT1 \wedge
 INITIATE_MOVEMENT \neq COMPLETE_PLAN \wedge
 INITIATE_MOVEMENT \neq OPOID \wedge INITIATE_MOVEMENT \neq SUPERVISE \wedge
 INITIATE_MOVEMENT \neq REPORT2 \wedge INITIATE_MOVEMENT \neq COMPLETE \wedge
 RECON \neq REPORT1 \wedge RECON \neq COMPLETE_PLAN \wedge RECON \neq OPOID \wedge
 RECON \neq SUPERVISE \wedge RECON \neq REPORT2 \wedge RECON \neq COMPLETE \wedge
 REPORT1 \neq COMPLETE_PLAN \wedge REPORT1 \neq OPOID \wedge
 REPORT1 \neq SUPERVISE \wedge REPORT1 \neq REPORT2 \wedge
 REPORT1 \neq COMPLETE \wedge COMPLETE_PLAN \neq OPOID \wedge
 COMPLETE_PLAN \neq SUPERVISE \wedge COMPLETE_PLAN \neq REPORT2 \wedge
 COMPLETE_PLAN \neq COMPLETE \wedge OPOID \neq SUPERVISE \wedge
 OPOID \neq REPORT2 \wedge OPOID \neq COMPLETE \wedge SUPERVISE \neq REPORT2 \wedge
 SUPERVISE \neq COMPLETE \wedge REPORT2 \neq COMPLETE

Index

ConductORPType Theory, 38

Datatypes, 38

Theorems, 39

plCommand_distinct_clauses, 39

psgCommand_distinct_clauses, 39

slCommand_distinct_clauses, 39

slCommand_one_one, 39

slOutput_distinct_clauses, 39

slRole_distinct_clauses, 39

slState_distinct_clauses, 39

ConductPBType Theory, 45

Datatypes, 45

Theorems, 45

plCommandPB_distinct_clauses, 45

psgCommandPB_distinct_clauses, 45

slCommand_distinct_clauses, 45

slCommand_one_one, 45

slOutput_distinct_clauses, 46

slRole_distinct_clauses, 46

slState_distinct_clauses, 46

MoveToORPType Theory, 51

Datatypes, 51

Theorems, 51

slCommand_distinct_clauses, 51

slOutput_distinct_clauses, 51

slState_distinct_clauses, 51

MoveToPBType Theory, 56

Datatypes, 56

Theorems, 56

slCommand_distinct_clauses, 56

slOutput_distinct_clauses, 57

slState_distinct_clauses, 57

OMNIType Theory, 3

Datatypes, 3

Theorems, 3

command_distinct_clauses, 3

command_one_one, 3

escCommand_distinct_clauses, 3

escOutput_distinct_clauses, 3

escState_distinct_clauses, 3

output_distinct_clauses, 4

output_one_one, 4

principal_one_one, 4

state_distinct_clauses, 4

state_one_one, 4

PBIntegratedDef Theory, 28

Definitions, 28

secAuthorization_def, 28

secHelper_def, 28

Theorems, 28

getOmniCommand_def, 28

getOmniCommand_ind, 31

secContext_def, 32

secContext_ind, 33

PBTypeIntegrated Theory, 26

Datatypes, 26

Theorems, 27

omniCommand_distinct_clauses, 27

plCommand_distinct_clauses, 27

slCommand_distinct_clauses, 27

slCommand_one_one, 27

slOutput_distinct_clauses, 27

slState_distinct_clauses, 28

stateRole_distinct_clauses, 28

PlanPBType Theory, 67

Datatypes, 67

Theorems, 68

plCommand_distinct_clauses, 68

psgCommand_distinct_clauses, 68

slCommand_distinct_clauses, 68

slCommand_one_one, 69

slOutput_distinct_clauses, 69

slRole_distinct_clauses, 70

slState_distinct_clauses, 70

satList Theory, 21

Definitions, 21

satList_def, 21

Theorems, 21

- satList_conj, 21
- satList_CONS, 21
- satList_nil, 21
- ssm Theory**, 11
 - Datatypes, 11
 - Definitions, 12
 - authenticationTest_def, 12
 - commandList_def, 12
 - inputList_def, 12
 - propCommandList_def, 12
 - TR_def, 12
 - Theorems, 13
 - CFGInterpret_def, 13
 - CFGInterpret_ind, 13
 - configuration_one_one, 13
 - extractCommand_def, 13
 - extractCommand_ind, 13
 - extractInput_def, 14
 - extractInput_ind, 14
 - extractPropCommand_def, 15
 - extractPropCommand_ind, 15
 - TR_cases, 16
 - TR_discard_cmd_rule, 17
 - TR_EQ_rules_thm, 17
 - TR_exec_cmd_rule, 17
 - TR_ind, 18
 - TR_rules, 18
 - TR_strongind, 19
 - TR_trap_cmd_rule, 20
 - TRrule0, 20
 - TRrule1, 20
 - trType_distinct_clauses, 20
 - trType_one_one, 21
- ssm11 Theory**, 4
 - Datatypes, 4
 - Definitions, 4
 - TR_def, 4
 - Theorems, 5
 - CFGInterpret_def, 5
 - CFGInterpret_ind, 6
 - configuration_one_one, 6
 - order_distinct_clauses, 6
 - order_one_one, 6
 - TR_cases, 6
 - TR_discard_cmd_rule, 7
 - TR_EQ_rules_thm, 7
 - TR_exec_cmd_rule, 8
 - TR_ind, 8
 - TR_rules, 9
 - TR_strongind, 9
 - TR_trap_cmd_rule, 10
 - TRrule0, 10
 - TRrule1, 11
 - trType_distinct_clauses, 11
 - trType_one_one, 11
- ssmConductORP Theory**, 33
 - Definitions, 33
 - secContextConductORP_def, 33
 - ssmConductORPStateInterp_def, 33
 - Theorems, 33
 - authTestConductORP_cmd_reject_lemma, 33
 - authTestConductORP_def, 34
 - authTestConductORP_ind, 34
 - conductORPNS_def, 35
 - conductORPNS_ind, 35
 - conductORPOut_def, 36
 - conductORPOut_ind, 36
 - PlatoonLeader_exec_plCommand_justified_thm, 37
 - PlatoonLeader_plCommand_lemma, 37
 - PlatoonSergeant_exec_psgCommand_justified_thm, 38
 - PlatoonSergeant_psgCommand_lemma, 38
- ssmConductPB Theory**, 39
 - Definitions, 40
 - secContextConductPB_def, 40
 - ssmConductPBStateInterp_def, 40
 - Theorems, 40
 - authTestConductPB_cmd_reject_lemma, 40
 - authTestConductPB_def, 40
 - authTestConductPB_ind, 41
 - conductPBNS_def, 42
 - conductPBNS_ind, 42

conductPBOut_def, 43
 conductPBOut_ind, 43
 PlatoonLeader_exec_plCommandPB_justified_thm, 44
 PlatoonLeader_plCommandPB_lemma, 44
 PlatoonSergeant_exec_psgCommandPB_justified_thm, 44
 PlatoonSergeant_psgCommandPB_lemma, 45
ssmMoveToORP Theory, 46
 Definitions, 46
 secContextMoveToORP_def, 46
 ssmMoveToORPStateInterp_def, 46
 Theorems, 46
 authTestMoveToORP_cmd_reject_lemma, 46
 authTestMoveToORP_def, 47
 authTestMoveToORP_ind, 47
 moveToORPNS_def, 48
 moveToORPNS_ind, 48
 moveToORPOut_def, 49
 moveToORPOut_ind, 49
 PlatoonLeader_exec_slCommand_justified_thm, 50
 PlatoonLeader_slCommand_lemma, 50
ssmMoveToPB Theory, 52
 Definitions, 52
 secContextMoveToPB_def, 52
 ssmMoveToPBStateInterp_def, 52
 Theorems, 52
 authTestMoveToPB_cmd_reject_lemma, 52
 authTestMoveToPB_def, 52
 authTestMoveToPB_ind, 53
 moveToPBNS_def, 53
 moveToPBNS_ind, 54
 moveToPBOut_def, 54
 moveToPBOut_ind, 55
 PlatoonLeader_exec_slCommand_justified_thm, 55
 PlatoonLeader_slCommand_lemma, 56
ssmPB Theory, 21
 Definitions, 21
 secContext_def, 21
 ssmPBStateInterp_def, 21
 Theorems, 22
 authenticationTest_cmd_reject_lemma, 22
 authenticationTest_def, 22
 authenticationTest_ind, 22
 PBNS_def, 23
 PBNS_ind, 23
 PBOut_def, 24
 PBOut_ind, 25
 PlatoonLeader_exec_slCommand_justified_thm, 26
 PlatoonLeader_slCommand_lemma, 26
ssmPlanPB Theory, 57
 Theorems, 57
 inputOK_def, 57
 inputOK_ind, 58
 planPBNS_def, 59
 planPBNS_ind, 59
 planPBOut_def, 59
 planPBOut_ind, 60
 PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma, 60
 PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm, 61
 PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma, 61
 PlatoonLeader_psgCommand_notDiscard_thm, 62
 PlatoonLeader_trap_psgCommand_justified_lemma, 62
 PlatoonLeader_trap_psgCommand_lemma, 63
 PlatoonLeader_WARNO_exec_report1_justified_lemma, 63
 PlatoonLeader_WARNO_exec_report1_justified_thm, 64
 PlatoonLeader_WARNO_exec_report1_lemma, 66
 PlatoonSergeant_trap_plCommand_justified_lemma, 66

PlatoonSergeant_trap_plCommand_justified.thm, 67
PlatoonSergeant_trap_plCommand_lemma, 67

Appendix B

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

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 from
*)
(* the input stream those inputs that are not of the form P says command. We
*)
(* 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 errors.
*)
(* -----

val _ =
Datatype
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 authentication
*)
(* function, (2) the interpretation 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 a
*)
(* 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)Form) =
  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 , ’d , ’e)Form) =
  ((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)Form
    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 list)
        ('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 =
TACPROOF(
  ([], ' '!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 list
    ('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  $\wedge$  ...  $\wedge$  (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 =
TACPROOF(
  ([ ,
    ‘((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 =
TACPROOF(
  ([ ,
    ‘!l1 l2 M Oi Os. ((M:( 'prop , 'world , 'pName , 'Int , 'Sec) Kripke) , (Oi: 'Int po) , (Os: 'Sec 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(DEPTHCONV BETA.CONV) THEN
  REWRITE.TAC[FOLDR] THEN
  CONV_TAC(DEPTHCONV BETA.CONV) THEN
  REWRITE.TAC[GSYM satList_def] THEN
  PROVE_TAC[])

val _ = save_thm("satList_conj" , satList_conj)

val satList_CONS =
TACPROOF([ [ ,
  ‘!h t M Oi Os. ((M:( 'prop , 'world , 'pName , 'Int , 'Sec) Kripke) , (Oi: 'Int po) , (Os: 'Sec po))
    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(DEPTHCONV BETA.CONV) THEN
  REWRITE.TAC[FOLDR] THEN
  CONV_TAC(DEPTHCONV 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

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 '

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 PBIIntegratedDef Theory: Authentication & Authorization Definitions

```

(*****
(* PBIIntegratedDefTheory

```

```

*)
(* 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 list
    = invalidOmniCommand:omniCommand) /\
  (getOmniCommand (((Name Omni) controls prop (SOME (SLc (OMNI cmd))))::xs)
    = (cmd:omniCommand)) /\
  (getOmniCommand ((x:((slCommand command)option , stateRole , 'd,'e)Form)::xs)
    = (getOmniCommand xs)) ‘

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

val secContext_def =
Define ‘
  (secContext (PLAN_PB) ((x:((slCommand command)option , stateRole , 'd,'e)Form list)
    [(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)Form list)
    [prop (SOME (SLc (OMNI (ssmMoveToORPComplete)))) impf
      (Name PlatoonLeader) controls prop (SOME (SLc (PL conductORP))))])
  (secContext (CONDUCT_ORP) ((x:((slCommand command)option , stateRole , 'd,'e)Form list)
    [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)Form list)
    [prop (SOME (SLc (OMNI (ssmMoveToPBComplete)))) impf
      (Name PlatoonLeader) controls prop (SOME (SLc (PL conductPB))))]) /\
  (secContext (CONDUCT_PB) ((x:((slCommand command)option , stateRole , 'd,'e)Form list)
    [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 precede
*)
(* transition to the next state at the topLevel. 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, because
*)
(* 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 aclInfRules 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 command
          :((slCommand command)option , stateRole , 'd, 'e)Form))) \ /
        (inputOK (((Name Omni) says prop (cmd:((slCommand command
          :((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.