Figure 1: This is a caption for this figure.

Abstract

This is the abstract for my master's thesis on Certified Security by Design (CSBD) and Access-control logic (ACL) because ACL is so cool and effective...because effective is cool. Here is some text for formatting.

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

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

Copyright

Disclaimer

Acknowledgements

Table Of Contents

List of Figures

List of Tables

List of Acronyms

ACL access-control logic.

CSBD Certified Security by Design.

WFF well-formed formula.

Chapter 1

Introduction

Some text here.[1] testing citations from the references.

1.1 Motivation

abelsec:intro:motivation

1.1.1 Systems Are Everywhere

1.1.2 CIA: Confidentiality, Integrity, and Accountability

1.2 This Master Thesis

This master thesis describes a method for designing secure systems. The method is called Certified Security by Design (CSBD). CSBD has been successfully demonstrated

on automated systems such as ... and .... But, until this research, it has not been demonstrated on non-automated, human-centered systems.

Systems span the range of fully automated to fully non-automated. This master thesis focuses on one end of this range: non-automated, human-centered systems.

The question addressed in this master thesis is whether CSBD could be successfully applied to non-automated, human-centered systems. An example of a non-automated, human-centered system is the patrol base operations defined in the United States Army Ranger Handbook[2]. Patrol base operations exemplify a non-automated, human-centered system wherein security is critical to mission success. In this master thesis, the results of applying CSBD to patrol base operations is discussed.

The patrol base operations are also an example of a predefined system. This means that this thesis also addresses the question of whether CSBD could be successfully applied to a pre-designed, non-automated, human-centered system. This is important because many such systems in use today are already designed and implemented. This means that a method for verifying and documenting the security properties of current, in-use systems is needed.

The successful application of CSBD to patrol base operations also suggests its use in combining automation with human-centered systems. The approach employed by this master thesis involves describing the patrol base operations as a hierarchy of secure state machines. This hierarchy has the property that it is easy to demonstrate security properties of the system, which is the goal of CSBD. But, it also has the property that it describes the patrol base operations as a system that is amiable to automation. Such automations of pre-defined non-automated, human-centered systems could include, for example, accountability systems for tracking supplies and personel. In the not-so-distant future, the miliatary, in particular, will most likely seek tracking and accountability systems for pre-existing, non-automated military operations. These

systems, like all security-sensitive military systems, should be designed according to NIST 800-160 standards. These standards require the formal verification and documentation provided by CSBD and demonstrated in this thesis.

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 Systems

3.2 Systems Engineering

This is some text with figures. There is a figure here and text referencing figure ?. More text before the figure call.

3.2.1 ISO/IEC/IEEE 15288

3.3 Systems Security Engineering

3.3.1 NIST Special Publication 800-160

3.3.1.1 Systems Security Engineering Framework
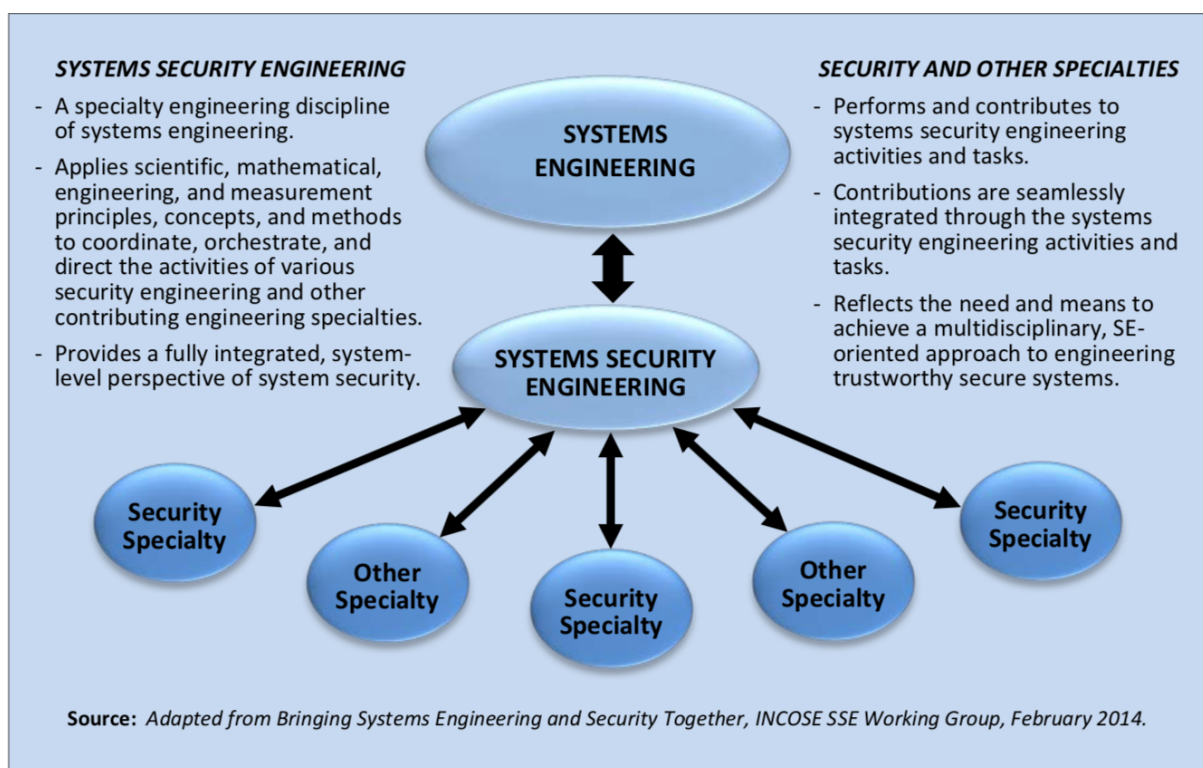
Figure 3.1: This is a clip from NIST 800-160.

This is some text after the figure.

3.3.2   Trustworthiness

3.3.2.1   Complete Mediation

3.3.3   Verification

3.3.4   Documentation

3.3.5   Reproducibility

3.4   Verification & Documentation

3.5   Principle of Complete Mediation

3.5.1   Formal Verification Using Computer-Aided Reasoning

Chapter 4

Certified Security by Design (CSBD)

&

Access-Control Logic (ACL)

4.1 Certified Security by Design (CSBD)

CSBD is a method for verifying and documenting the security properties of a systems. It uses an access-control logic (ACL) to reason about access to security sensitive objects of a system. It uses computer-aided reasoning such as the Higher Order Logic (HOL) Interactive theorem prover to formally verify and document these security properties. The outcomes of CSBD applied to a system conform to the guildelines set fourth in NIST 800-160 [verify and discuss this.]

In addition to providing formal proofs that demonstrate satisfiability of complete mediation, CDBD is reproducible. This means that third parties can also verify the formal proofs. This touches on the heart of formal verification of satisfiability: "don't just take my word for it, prove it for yourself."

In 1970 The Rand Corporation published a report[3] for the Office of The Director of Defense Research And Engineering. This report titled, Security Controls For Computer Systems, noted that "Providing satisfactory security controls in a computer system is in itself a system design problem." NIST 800-160 also highlights the importance of incorporating security into the design phase of the system engineering process. Certified Security by Design (CSBD) focuses on this aspect of systems security engineering. It does not describe[1] an implementation of a system. Instead, CSBD applies formal methods of verification to the design phase of systems security engineering.

4.1.1 The Principle of Complete Mediation

4.2 Access-Control Logic (ACL)

4.2.1 ACL: A Command and Control (C2) Calculus

This section discusses the access-control logic in sufficient detail to understand the research reported in this master thesis. The material is adapted from *Access Control, Security, and Trust: A Logical Approach*[1]. For more indepth coverage of the ACL, read the text. Any references to "the text" in this section refer to the aforementioned text book.

ACL is a logic for reasoning about access to object. In the jargon of the day it is a command and control (C2) calculus[2].

---

[1]Although, not the topic of this master thesis, there is currently ample research in the area of formal verification on computer programs. See references [].

[2]command and control being self-evident and calculus being a method for reasoning

4.2.2 Principals

Principals should be thought of as actors in the access-control logic. Principals can make statements or requests. They can be assigned privileges or authority over objects or actions. The text defines allowable principals using the identifier **Princ**:

$$\textbf{Princ ::= PName / Princ \& Princ / Princ |Princ}$$

This is a recursive definition. ***PName*** refers to the name of a principal (i.e., Jane, PlatoonLeader, sensor1). ***Princ & Princ*** is read "Princ with Princ" or "Princ and Princ" (i.e., Principal1 with Principal2). ***Princ |Princ*** is read as "Princ quoting Princ" (i.e., Principal1 quoting Principal2).

4.2.3 Propositional Variables, Requests, Authority, and Jurisdiction

To reason about access-control and trust, the ACL uses propositional variables, requests, authority, and jurisdiction to make statements.

Propositions in logic are assertions that are either true or false. For example, "I am reading this master thesis" is a proposition because either you are or you are not reading this. Propositional variables are just place holders for propositions. For example, "I am reading something", where the propositional variable "something" is what you are reading.

Principals can make requests. In the ACL, principals make requests using the *says* operator. Requests have the form *P says φ*, where *P* represents some principal and φ represents some assertion. For example, *PlatoonLeader says platoonHalt.* In this example, the Platoon Leader is issuing a command (or request) for the platoon to halt.

Principals can have authority over assertions. In the ACL, authority is conveyed using the *controls* operator. Statements of authority have the form *P controls φ*, where *P* represents some principal and *φ* represents some assertion. For example, *PlatoonLeader controls platoonHalt*. This example states that the Platoon Leader has the authority to issue the command (or request) for the platoon to halt.

Principals can also have jurisdiction over assertions. Both authority and jurisdiction use the *controls* operator. Statements of jurisdiction have the same form as statements of authority. Statements of authority are typically defined in an organization's policy. Statements of jurisdiction are statements that are readily believed given the context. For example, *PresidentOfUS controls (PlatoonLeader controls platoonHalt)*. In this example, the President of the United States, per the U.S. Constitution, has jurisdiction over the authority invested in the Platoon Leader. In particular, the President of the United States has the jurisdiction to give the Platoon Leader the authority to command her platoon to halt.

In addition, principals can speak for other principals. Principals do this using the *speaks for* operator. The ACL represents the *speaks for* operator with the symbol $\Rightarrow$. These types of statements have the form $P \Rightarrow Q$, where both *P* and *Q* are principals. For example, *PlatoonLeader $\Rightarrow$ PresidentOfUS*. This example states that the Platoon Leader speaks for the President of the United States.

4.2.4   Well-formed Formulas

Well-formed formulas (WFFs) are valid statements in the ACL. All statements must be a WFF. The text book defines the set of WFFs using the identifier **Form**:

$$\textbf{Form} ::= \textbf{PropVar} \ / \ \neg \ \textbf{Form} \ / \ (\textbf{Form} \lor \textbf{Form}) \ /$$
$$(\textbf{Form} \land \textbf{Form}) \ / \ (\textbf{Form} \supset \textbf{Form}) \ / \ (\textbf{Form} \equiv \textbf{Form}) \ /$$

$$(\textbf{Princ} \Rightarrow \textbf{Princ}) \text{ / } (\textbf{Princ says Princ}) \text{ / } (\textbf{Princ controls Form})$$

This is a recursive definition. ***PropVar*** is a propositional variable. The symbols $\neg$, $\vee$, $\wedge$, $\subset$, and $\equiv$ are the standard set and logical symbols. They represent "not", "or", "and", "subset", and "equivalence", respectively. This master thesis primarily reasons with statements (WFFs) of the form ***Princ*** *says* ***Princ*** and ***Princ*** *controls* ***Form***.

4.2.5   Kripke Structure

A Kripke structure deals with three things: worlds, propositions, and principals. The worlds can be thought of as possible states or configurations of some system. Propositions are just statements that are either true or false. And, principals are just actors. A Kripke structure $\mathcal{M} = \langle W, I, J \rangle$ is defined as a three-tuple consisting of a set of worlds W, a function I that maps propositions to worlds, and a functions J that maps principals to relations on worlds. A more formal definition is definition 2.1 in the text:

- *W is a nonempty set, whose elements are called worlds.*

- *$I : \textbf{PropVar} \to \mathcal{P}(W)$ is an interpretation function that maps each propositional variable to a set of worlds.*

- *$J : \textbf{PName} \to \mathcal{P}((W \times W)$ is a function that maps each principal name to a relation on worlds.*

$$
\begin{aligned}
\mathcal{E}_{\mathcal{M}}[\![p]\!] &= I(p) \\
\mathcal{E}_{\mathcal{M}}[\![\neg\varphi]\!] &= W - \mathcal{E}_{\mathcal{M}}[\![\varphi]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \wedge \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \vee \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!] \cup \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \supset \varphi_2]\!] &= (W - \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!]) \cup \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\
\mathcal{E}_{\mathcal{M}}[\![\varphi_1 \equiv \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1 \supset \varphi_2]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\varphi_2 \supset \varphi_1]\!] \\
\mathcal{E}_{\mathcal{M}}[\![P \Rightarrow Q]\!] &= \begin{cases} W, & \text{if } \hat{J}(Q) \subseteq \hat{J}(P) \\ \emptyset, & \text{otherwise} \end{cases} \\
\mathcal{E}_{\mathcal{M}}[\![P \text{ says } \varphi]\!] &= \{w | \hat{J}(P)(w) \subseteq \mathcal{E}_{\mathcal{M}}[\![\varphi]\!]\} \\
\mathcal{E}_{\mathcal{M}}[\![P \text{ controls } \varphi]\!] &= \mathcal{E}_{\mathcal{M}}[\![(P \text{ says } \varphi) \supset \varphi]\!] \\
\mathcal{E}_{\mathcal{M}}[\![P \text{ reps } Q \text{ on } \varphi]\!] &= \mathcal{E}_{\mathcal{M}}[\![(P \mid Q \text{ says } \varphi) \supset Q \text{ says } \varphi]\!]
\end{aligned}
$$

Figure 4.1: Kripke semantics. Image taken from *Access Control, Security, and Trust: A Logical Approach*[1]

4.2.5.1 satisies

4.2.5.2 soundness

4.2.6 Inference Rules

The inference rules for the access-control logic (ACL) are shown in figure 4.2. All the inference rules are sound. Details of proofs of soundness can be found in *Access Control, Security, and Trust: A Logical Approach*[1].

4.2.7 Complete mediation

Fundamental to this work is the concept of complete mediation (discussed in section 4.1.1). In the ACL, this means that each principal must be authenticated and authorized on each request. ACL does this primarily by the *Controls* inference rule in

$$P \text{ controls } \varphi \;\overset{\text{def}}{=}\; (P \text{ says } \varphi) \supset \varphi \qquad P \text{ reps } Q \text{ on } \varphi \overset{\text{def}}{=} P \mid Q \text{ says } \varphi \supset Q \text{ says } \varphi$$

$$\textit{Modus Ponens} \quad \frac{\varphi \quad \varphi \supset \varphi'}{\varphi'} \qquad \textit{Says} \quad \frac{\varphi}{P \text{ says } \varphi} \qquad \textit{Controls} \quad \frac{P \text{ controls } \varphi \quad P \text{ says } \varphi}{\varphi}$$

$$\textit{Derived Speaks For} \quad \frac{P \Rightarrow Q \quad P \text{ says } \varphi}{Q \text{ says } \varphi} \qquad \textit{Reps} \quad \frac{Q \text{ controls } \varphi \quad P \text{ reps } Q \text{ on } \varphi \quad P \mid Q \text{ says } \varphi}{\varphi}$$

$$\textit{\& Says (1)} \quad \frac{P \,\&\, Q \text{ says } \varphi}{P \text{ says } \varphi \wedge Q \text{ says } \varphi} \qquad \textit{\& Says (2)} \quad \frac{P \text{ says } \varphi \wedge Q \text{ says } \varphi}{P \,\&\, Q \text{ says } \varphi}$$

$$\textit{Quoting (1)} \quad \frac{P \mid Q \text{ says } \varphi}{P \text{ says } Q \text{ says } \varphi} \qquad \textit{Quoting (2)} \quad \frac{P \text{ says } Q \text{ says } \varphi}{P \mid Q \text{ says } \varphi}$$

$$\textit{Idempotency of} \Rightarrow \quad \frac{}{P \Rightarrow P} \qquad \textit{Monotonicity of} \Rightarrow \quad \frac{P' \Rightarrow P \quad Q' \Rightarrow Q}{P' \mid Q' \Rightarrow P \mid Q}$$

Figure 4.2: The ACL inference rules. Image taken from *Access Control, Security, and Trust: A Logical Approach*[1]

figure 4.2 and shown again here in figure 4.3.

$$\textit{Controls} \quad \frac{P \text{ controls } \varphi \quad P \text{ says } \varphi}{\varphi}$$

Figure 4.3: The *Controls* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[1]

ACL refers to the left statement as an authorization[3]. The principal P controls (is authorized on) some action φ. ACL refers to the right statement in this inference rule as a request [4]. The principal P requests some action φ. The conjunction of the authorization and the request of P on φ results in the action φ. That is, if *P controls φ* and *P says φ* then φ is true.

The *Reps* rule also demonstrates complete mediation. It follows a similar logic. However, the *Reps* rule is not used in this master thesis.

---

[3]or a *control* in the C2 calculus

[4]or a *command* in the C2 calculus

4.3   ACL in HOL

The equivalence of the ACL formulas implemented in HOL are shown in figure 4.4.

Access-Control Logic Formula	HOL Syntax
$\langle jump \rangle$	`prop jump`
$\neg \langle jump \rangle$	`notf (prop jump)`
$\langle run \rangle \wedge \langle jump \rangle$	`prop run andf prop jump`
$\langle run \rangle \vee \langle stop \rangle$	`prop run orf prop stop`
$\langle run \rangle \supset \langle jump \rangle$	`prop run impf prop jump`
$\langle walk \rangle \equiv \langle stop \rangle$	`prop walk eqf prop stop`
$Alice$ says $\langle jump \rangle$	`Name Alice says prop jump`
$Alice$ & Bob says $\langle stop \rangle$	`Name Alice meet Name Bob says prop stop`
$Bob \mid Carol$ says $\langle run \rangle$	`Name Bob quoting Name Carol says prop run`
Bob controls $\langle walk \rangle$	`Name Bob controls prop walk`
Bob reps $Alice$ on $\langle jump \rangle$	`reps (Name Bob) (Name Alice) (prop jump)`
$Carol \Rightarrow Bob$	`Name Carol speaks_for Name Bob`

Figure 4.4: The ACL formulas in HOL. Image taken from *Access Control, Security, and Trust: A Logical Approach*[1]

Using this syntax, an ACL request of the form *P says φ* would have the form

Name P says prop φ

4.3.1   Complete Mediation

4.3.2   satList

Chapter 5

Patrol Base Operations

5.1 Motivation

The patrol base operations described in the patrol base Ranger Handbook

5.2 Ranger Handbook Description

5.3 Modeling the Patrol Base Operations from the Ranger Handbook

Modeling a system requires the knowledge of an expert on the system. This is necessary because only someone who is familiar with the system, especially with regards to security, can detail its nuances. For this reason, a subject matter expert from the United States Army (Jesse Nathaniel Hall) is employed to develop a model of the patrol base operations.

The model of the patrol base operations needs to be amiable to complete mediation and verification using an access-control logic (section 4.2). This is necessary to prove security properties of the patrol base operations. To do this, the patrol base operations are abstracted from the Ranger Manual and modeled in Visio[1]. The result of doing this is a hierarchy of secure state machines (SSMs). (SSMs are described in section 6.2.)

Each level of the hierarchy of SSMs represents a level of abstraction of the patrol base operations. The most abstract level of the hierarchy is the top level SSM. A diagram of this most abstract level is shown in figure 5.1.
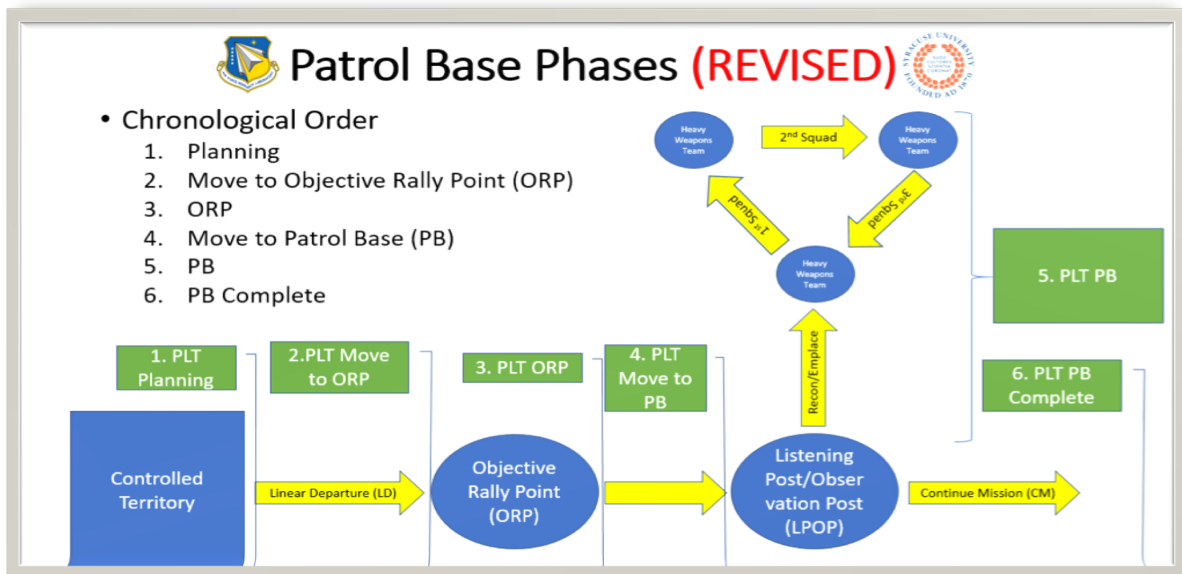
Figure 5.1: A diagram of the most abstract level in the hierarchy of secure state machines.

The diagram describes a chronological order of abstract phases (modeled as states) of the patrol base operations. The operations begin with the planning phase (1). Next, they move to the objective rally point (ORP) (2). At the ORP, operations commence (3). When these are complete, the patrol base operations move to the actual patrol base (4). At the patrol base, operations proceed (5). Finally, the patrol base operations are

---

[1]This work began as a collaboration between Jesse Nathaniel Hall and the author. Once the hierarchy of secure state machines was decided upon, the abstraction of the Ranger Handbook was done by Jesse Nathaniel Hall with only structural consultation with the author. Concurrently, the author focused on proving the properties of complete mediation in the ACL using HOL. Thus, there was a great deal of separation of work. Jesse's work is described here because it is necessary to put the entire system into context for this master thesis. This means that Jesse's work provided the model of the system for which the principle of complete mediation was proved, verified and documented.

complete (6). These are the six states in the top level SSM.

The next level of abstraction in the hierarchy of SSMs represents a horizontal slice through the patrol base operations. This is the second level of the hierarchical description of the patrol base operations. It is referred to as the sub level. In this documentation, SSMs at this level are referred to as the sub-level, sublevel, or subLevel SSMs. This slice describes the patrol base operations at a lower level of abstraction. It expands each of the states in the top level (except for the last state PB Complete). For example, the planning phase (1) in figure 5.1 is expanded into an SSM of its own. This is called ssmPlanPB. It consists of several states (see section 5.4.5.1) which detail activities conducted during the planning phase of the patrol base operations. Each state in the top level (except for PB Complete) has it's own SSM (see the next section).

At yet another lower level of abstraction is the sub-sub (3rd) level. In this documentation, SSMs at this level are referred to as the sub-sub-level, subsublevel, or subsubLevel SSMs. This level expands upon the states in the sub level (one level above) in the same manner that the sub level expands upon the states in the top level SSM. In this manner, each level is a lower level of abstraction than the level above it.

A vertical slice through the diagram is also modeled. This slice models the patrol base operations from the top level down to the most detailed level (level 8). This vertical slice consists of a series of SSMs. Each SSM expands upon only one state in the level above it. This differs from the horizontal slice which expands upon all states in the level above it. Expanding upon only one state focuses on a vertical slice through all 8 states of the hierarchy of SSMs.

The vertical slice begins at the top level SSM. Next, it expands upon one state at this level, the *move to ORP* state (2). This results in a sub level SSM named ssmMoveToORP. The vertical slice progresses in this manner, by expanding one state at each level into a new SSM. From ssmMoveToORP, the state *secure halt* is expanded to

ssmSecureHalt. From within this SSM, the state *ORP Recon* is expanded into ssmORPRecon. From within this SSM, the state *Move to ORP 4L* (fourth level move to ORP state) is expanded into ssmMoveToORP4L. Finally, from within this SSM, the state *Form RT* is expanded into ssmFormRT.

The vertical slice spans the all eight levels. However, not all levels are represented with an SSM. The last SSM in the vertical slice, ssmFormRT, is actually at the 5th level of the hierarchy. This SSM, consists of three states. These three states reside at the 7th level because ssmFormRT does not have states at the 6th level (it skips the 6th level). Furthermore, the 7th level states are not expanded into an SSM because each of these 7th level states expand into only one state at the 8th level.

In addition to the horizontal and vertical slices, an escape level is also modeled. Actions in the escape level are reachable from any phase of the patrol base operations. These actions are the unacceptable circumstances that require the patrol base operations to abort. For example, if the patrol base contacts the enemy in any phase of the operations, then the command *react to combat* is issued. The patrol base operations are subsequently aborted.

Excluding the escape level, there are eight levels of the hierarchy of SSMs.

Note that, the purpose of this master thesis is to demonstrate that the properties of complete mediation could be applied and verified on a non-automated, human-centered systems. Thus, it is sufficient to demonstrate this on a horizontal and vertical slice of the patrol base operations.
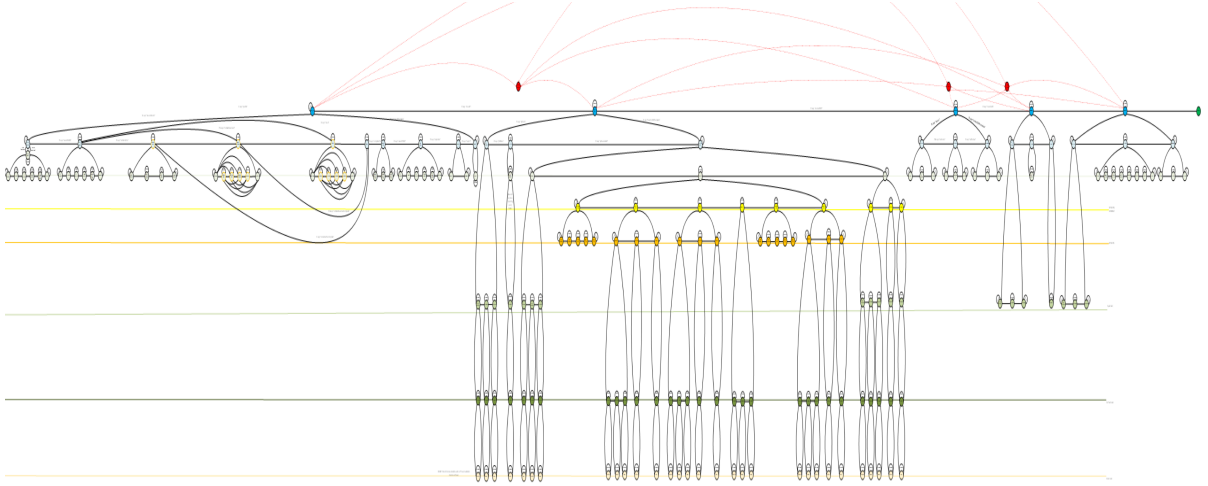
Figure 5.2: Diagrammatic description of patrol base operations as a hierarchy of secure state machines. (Generated by Jesse Nathaniel Hall.)

5.4 Hierarchy of Secure State Machines

5.4.1 Diagrammatic Description in Visio

The enormity of the hierarchy of SSMs is evident in figure 5.2. This is a squashed version of the Visio diagram for the hierarchy of SSMs. The diagram is included as a Visio file with the files for this project (LaTeX/figures/diagram.vis).

The straight, colored lines that span the diagram in figure 5.2 delineate levels of the hierarchy of SSMs. The top lines are obscured by the size of this squashed version of the diagram. The most visible bright yellow line delineates the sub-sub-sub (4th) level of the hierarchy, for example.

The small, colored dots in figure 5.2 represent states (phases) of the patrol base operations. The red dots are an exception. The labels for these states are not readable in this diagram. The dots are color coded. The colors correspond to the level of those states. For example, the dots at the top level (level below the red dots) are all dark blue.

In figure 5.2, the red dots at the top of the diagram represent the escape level SSM.

But, they do not represent states in the SSM. This is because the escape level SSM is accessible by all phases of the patrol base operations. This means that if can not be ascribed to any one level of the hierarchy.

The dots representing states are connected to each other by lines. These lines represent allowable transitions from one state to another. The escape level is again an exception. If no line connects one state to another then no transition is allowed.

The red dots representing the escape level SSM are best thought of as multiple copies of a floating SSM. The escape level SSM acts as a sub-SSM for all SSMs in the hierarchy. During patrol base operations, abortion of the patrol base operations can occur at any action from any state at any level. This means that any state at any level can be terminated by the escape level SSM. Drawing lines from all states to the escape level SSM and drawing really long lines clutters the diagram. Therefore, only lines at the top level are drawn and the red dots are duplicated.

The lines in the diagram are annotated by SSM requests. (Annotations are visible in the original Visio diagram, but not in this squashed version.) For example, a line connecting the top level state PLAN_PB is annotated with the request *PlatoonLeader says crossLD*. crossLD is an abbreviation for "cross the line of discrimination" and it is the command to transition to the MOVE_TO_ORP state. Lines are not annotated beyond the sub level.

Details of each level follow in the next section.

5.4.2 OMNI-Level

5.4.3 Escape

5.4.4 Top Level

5.4.5 Horizontal Slice

5.4.5.1 ssmPlanPB

5.4.5.2 ssmMoveToORP

labelsssec:ssmMoveToORP

5.4.5.3 ssmConductORP

labelsssec:ssmConductORP

5.4.5.4 ssmMoveToPB

labelsssec:ssmMoveToPB

5.4.5.5 ssmConductPB

5.4.6 Vertical Slice

5.4.6.1 ssmSecureHalt

5.4.6.2 ssmORPRecon

5.4.6.3 ssmMoveToORP4L

5.4.6.4 ssmFormRT

secure state machine (SSM)–glesentryfull

secure state machines (SSMs)–glesentryfullpl

SSM–glsentrytext

SSMs–glsentryshortpl

Chapter 6

Secure State Machine Model

6.1 State Machines

6.1.1 Next-state Function

6.1.2 Next-output Function

6.1.3 Transition Commands

6.2 Secure State Machines

6.2.1 State Machine Versus Secure State Machine

6.2.2 Transition Types

6.2.3 Authentication

6.2.4 Authorization

Chapter 7

Patrol Base Operations as Secure State Machines

7.1 ssmPB: An Example from the Hierarchy

7.1.1 Principals

7.1.2 States

7.1.3 Commands

7.1.4 Next-State Function

7.1.5 Next-Output Function

7.1.6 Authentication

7.1.7 Authorization

Chapter 8

Discussion

8.1 Recap

8.2 Mission Accomplished

8.3 Stop-Gaps, Lessons Learned, & Advice

8.4 Other Verifiable Theories

8.4.1 Platoon Theory, Soldier Theory, Squad Theory, etc.

8.4.2 Soldiers in Roles

Chapter 9

Future Work & Implications

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

9.1   The Devil Is in The Details

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

9.2   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.

9.3 Applicability

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

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

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

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

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

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

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

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

Appendices

Appendix A

Access Control Logic Theories: Pretty-Printed Theories

Contents

1 aclfoundation Theory

Built: 25 February 2018
Parent Theories: indexedLists, patternMatches

1.1 Datatypes

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

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

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

IntLevel = iLab 'il | il 'apn ;

SecLevel = sLab 'sl | sl 'apn
```

1.2 Definitions

[imapKS_def]

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

[intpKS_def]

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

[jKS_def]

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

[O1_def]

$\vdash$ O1 = PO one_weakorder

[one_weakorder_def]

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

[po_TY_DEF]

$\vdash \exists$ *rep*. TYPE_DEFINITION WeakOrder *rep*

[po_tybij]

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

[prod_PO_def]

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

[smapKS_def]

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

[Subset_PO_def]

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

1.3 Theorems

[abs_po11]

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

[absPO_fn_onto]

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

[antisym_prod_antisym]

⊢ ∀ r s.
    antisymmetric r ∧ antisymmetric s ⇒
    antisymmetric (RPROD r s)

[EQ_WeakOrder]

⊢ WeakOrder (=)

[KS_bij]

⊢ ∀ M. M = KS (intpKS M) (jKS M) (imapKS M) (smapKS M)

[one_weakorder_WO]

⊢ WeakOrder one_weakorder

[onto_po]

⊢ ∀ r. WeakOrder r ⟺ ∃ a. r = repPO a

[po_bij]

⊢ (∀ a. PO (repPO a) = a) ∧
  ∀ r. WeakOrder r ⟺ (repPO (PO r) = r)

[PO_repPO]

⊢ ∀ a. PO (repPO a) = a

[refl_prod_refl]

⊢ ∀ r s. reflexive r ∧ reflexive s ⇒ reflexive (RPROD r s)

[repPO_iPO_partial_order]

⊢ (∀ x. repPO iPO x x) ∧
  (∀ x y. repPO iPO x y ∧ repPO iPO y x ⇒ (x = y)) ∧
  ∀ x y z. repPO iPO x y ∧ repPO iPO y z ⇒ repPO iPO x z

[repPO_O1]

⊢ repPO O1 = one_weakorder

[repPO_prod_PO]

⊢ ∀ po_1 po_2.
    repPO (prod_PO po_1 po_2) = RPROD (repPO po_1) (repPO po_2)

[repPO_Subset_PO]

⊢ repPO Subset_PO = (⊆)

[RPROD_THM]

⊢ ∀ r s a b.
    RPROD r s a b ⟺ r (FST a) (FST b) ∧ s (SND a) (SND b)

[SUBSET_WO]

⊢ WeakOrder (⊆)

[trans_prod_trans]

⊢ ∀ *r s*. transitive *r* ∧ transitive *s* ⇒ transitive (RPROD *r s*)

[WeakOrder_Exists]

⊢ ∃ *R*. WeakOrder *R*

[WO_prod_WO]

⊢ ∀ *r s*. WeakOrder *r* ∧ WeakOrder *s* ⇒ WeakOrder (RPROD *r s*)

[WO_repPO]

⊢ ∀ *r*. WeakOrder *r* ⟺ (repPO (PO *r*) = *r*)

2 aclsemantics Theory

Built: 25 February 2018

Parent Theories: aclfoundation

2.1 Definitions

[Efn_def]

⊢ (∀ *Oi Os M*. Efn *Oi Os M* TT = 𝒰(:'v)) ∧
  (∀ *Oi Os M*. Efn *Oi Os M* FF = { }) ∧
  (∀ *Oi Os M p*. Efn *Oi Os M* (prop *p*) = intpKS *M p*) ∧
  (∀ *Oi Os M f*.
    Efn *Oi Os M* (notf *f*) = 𝒰(:'v) DIFF Efn *Oi Os M f*) ∧
  (∀ *Oi Os M* f_1 f_2.
    Efn *Oi Os M* (f_1 andf f_2) =
    Efn *Oi Os M* f_1 ∩ Efn *Oi Os M* f_2) ∧
  (∀ *Oi Os M* f_1 f_2.
    Efn *Oi Os M* (f_1 orf f_2) =
    Efn *Oi Os M* f_1 ∪ Efn *Oi Os M* f_2) ∧
  (∀ *Oi Os M* f_1 f_2.
    Efn *Oi Os M* (f_1 impf f_2) =
    𝒰(:'v) DIFF Efn *Oi Os M* f_1 ∪ Efn *Oi Os M* f_2) ∧
  (∀ *Oi Os M* f_1 f_2.
    Efn *Oi Os M* (f_1 eqf f_2) =
    (𝒰(:'v) DIFF Efn *Oi Os M* f_1 ∪ Efn *Oi Os M* f_2) ∩
    (𝒰(:'v) DIFF Efn *Oi Os M* f_2 ∪ Efn *Oi Os M* f_1)) ∧
  (∀ *Oi Os M P f*.
    Efn *Oi Os M* (*P* says *f*) =
    { *w* | Jext (jKS *M*) *P w* ⊆ Efn *Oi Os M f*}) ∧
  (∀ *Oi Os M P Q*.
    Efn *Oi Os M* (*P* speaks_for *Q*) =

        if Jext (jKS M) Q RSUBSET Jext (jKS M) P **then** $\mathcal{U}$(:'v)
        else { }) $\wedge$
   ($\forall\, Oi\ Os\ M\ P\ f$.
     Efn $Oi\ Os\ M$ (P controls f) =
     $\mathcal{U}$(:'v) DIFF $\{w \mid$ Jext (jKS M) $P\ w \subseteq$ Efn $Oi\ Os\ M\ f\} \cup$
     Efn $Oi\ Os\ M\ f$) $\wedge$
   ($\forall\, Oi\ Os\ M\ P\ Q\ f$.
     Efn $Oi\ Os\ M$ (reps $P\ Q\ f$) =
     $\mathcal{U}$(:'v) DIFF
     $\{w \mid$ Jext (jKS M) (P quoting Q) $w \subseteq$ Efn $Oi\ Os\ M\ f\} \cup$
     $\{w \mid$ Jext (jKS M) $Q\ w \subseteq$ Efn $Oi\ Os\ M\ f\}$) $\wedge$
   ($\forall\, Oi\ Os\ M\ intl_1\ intl_2$.
     Efn $Oi\ Os\ M$ ($intl_1$ domi $intl_2$) =
     if repPO Oi (Lifn $M\ intl_2$) (Lifn $M\ intl_1$) **then** $\mathcal{U}$(:'v)
     else { }) $\wedge$
   ($\forall\, Oi\ Os\ M\ intl_2\ intl_1$.
     Efn $Oi\ Os\ M$ ($intl_2$ eqi $intl_1$) =
     (**if** repPO Oi (Lifn $M\ intl_2$) (Lifn $M\ intl_1$) **then** $\mathcal{U}$(:'v)
      else { }) $\cap$
     if repPO Oi (Lifn $M\ intl_1$) (Lifn $M\ intl_2$) **then** $\mathcal{U}$(:'v)
     else { }) $\wedge$
   ($\forall\, Oi\ Os\ M\ secl_1\ secl_2$.
     Efn $Oi\ Os\ M$ ($secl_1$ doms $secl_2$) =
     if repPO Os (Lsfn $M\ secl_2$) (Lsfn $M\ secl_1$) **then** $\mathcal{U}$(:'v)
     else { }) $\wedge$
   ($\forall\, Oi\ Os\ M\ secl_2\ secl_1$.
     Efn $Oi\ Os\ M$ ($secl_2$ eqs $secl_1$) =
     (**if** repPO Os (Lsfn $M\ secl_2$) (Lsfn $M\ secl_1$) **then** $\mathcal{U}$(:'v)
      else { }) $\cap$
     if repPO Os (Lsfn $M\ secl_1$) (Lsfn $M\ secl_2$) **then** $\mathcal{U}$(:'v)
     else { }) $\wedge$
   ($\forall\, Oi\ Os\ M\ numExp_1\ numExp_2$.
     Efn $Oi\ Os\ M$ ($numExp_1$ eqn $numExp_2$) =
     if $numExp_1$ = $numExp_2$ **then** $\mathcal{U}$(:'v) **else** { }) $\wedge$
   ($\forall\, Oi\ Os\ M\ numExp_1\ numExp_2$.
     Efn $Oi\ Os\ M$ ($numExp_1$ lte $numExp_2$) =
     if $numExp_1 \leq numExp_2$ **then** $\mathcal{U}$(:'v) **else** { }) $\wedge$
  $\forall\, Oi\ Os\ M\ numExp_1\ numExp_2$.
    Efn $Oi\ Os\ M$ ($numExp_1$ lt $numExp_2$) =
    if $numExp_1 < numExp_2$ **then** $\mathcal{U}$(:'v) **else** { }

[Jext_def]
$\vdash$ ($\forall\, J\ s$. Jext J (Name s) = $J\ s$) $\wedge$
  ($\forall\, J\ P_1\ P_2$.
    Jext J (P_1 meet P_2) = Jext $J\ P_1$ RUNION Jext $J\ P_2$) $\wedge$
  $\forall\, J\ P_1\ P_2$. Jext J (P_1 quoting P_2) = Jext $J\ P_2$ O Jext $J\ P_1$

[Lifn_def]
$\vdash$ ($\forall\, M\ l$. Lifn M (iLab l) = l) $\wedge$
  $\forall\, M\ name$. Lifn M (il $name$) = imapKS $M\ name$

$[$`Lsfn_def`$]$

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

2.2 Theorems

$[$`andf_def`$]$

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

$[$`controls_def`$]$

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

$[$`controls_says`$]$

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

$[$`domi_def`$]$

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

$[$`doms_def`$]$

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

$[$`eqf_def`$]$

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

$[$`eqf_impf`$]$

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

[eqi_def]

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

[eqi_domi]

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

[eqn_def]

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

[eqs_def]

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

[eqs_doms]

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

[FF_def]

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

[impf_def]

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

[lt_def]

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

[lte_def]

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

[meet_def]

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

[name_def]

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

[notf_def]

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

[orf_def]

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

[prop_def]

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

[quoting_def]

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

[reps_def]

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

[says_def]

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

[speaks_for_def]

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

[TT_def]

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

3   aclrules Theory

Built: 25 February 2018
Parent Theories: aclsemantics

3.1   Definitions

[sat_def]

⊢ ∀ M Oi Os f. (M,Oi,Os) sat f ⟺ (Efn Oi Os M f = $\mathcal{U}$(:'world))

3.2   Theorems

[And_Says]

⊢ ∀ M Oi Os P Q f.
   (M,Oi,Os) sat P meet Q says f eqf P says f andf Q says f

[And_Says_Eq]

⊢ (M,Oi,Os) sat P meet Q says f ⟺
   (M,Oi,Os) sat P says f andf Q says f

[and_says_lemma]

⊢ ∀ M Oi Os P Q f.
   (M,Oi,Os) sat P meet Q says f impf P says f andf Q says f

[Controls_Eq]

⊢ ∀ M Oi Os P f.
   (M,Oi,Os) sat P controls f ⟺ (M,Oi,Os) sat P says f impf f

[DIFF_UNIV_SUBSET]

⊢ ($\mathcal{U}$(:'a) DIFF s ∪ t = $\mathcal{U}$(:'a)) ⟺ s ⊆ t

[domi_antisymmetric]

⊢ ∀ M Oi Os l_1 l_2.
   (M,Oi,Os) sat l_1 domi l_2 ⟹
   (M,Oi,Os) sat l_2 domi l_1 ⟹
   (M,Oi,Os) sat l_1 eqi l_2

[domi_reflexive]

⊢ ∀ M Oi Os l. (M,Oi,Os) sat l domi l

[domi_transitive]

⊢ ∀ M Oi Os l_1 l_2 l_3.
   (M,Oi,Os) sat l_1 domi l_2 ⟹
   (M,Oi,Os) sat l_2 domi l_3 ⟹
   (M,Oi,Os) sat l_1 domi l_3

[doms_antisymmetric]

⊢ ∀ M Oi Os l_1 l_2.
   (M,Oi,Os) sat l_1 doms l_2 ⟹
   (M,Oi,Os) sat l_2 doms l_1 ⟹
   (M,Oi,Os) sat l_1 eqs l_2

[doms_reflexive]

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

[doms_transitive]

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

[eqf_and_impf]

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

[eqf_andf1]

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

[eqf_andf2]

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

[eqf_controls]

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

[eqf_eq]

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

[eqf_eqf1]

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

[eqf_eqf2]

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

[eqf_impf1]

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

[eqf_impf2]

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

[eqf_notf]

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

[eqf_orf1]

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

[eqf_orf2]

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

[eqf_reps]

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

[eqf_sat]

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

[eqf_says]

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

[eqi_Eq]

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

[eqs_Eq]

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

[Idemp_Speaks_For]

$\vdash \forall M\ Oi\ Os\ P.\ (M, Oi, Os)$ sat P speaks_for P

[Image_cmp]

$\vdash \forall R_1\ R_2\ R_3\ u.\ (R_1\ \mathbf{O}\ R_2)\ u \subseteq R_3 \iff R_2\ u \subseteq \{\,y\ \mid\ R_1\ y \subseteq R_3\,\}$

[Image_SUBSET]

$\vdash \forall R_1\ R_2.\ R_2$ RSUBSET $R_1 \Rightarrow \forall w.\ R_2\ w \subseteq R_1\ w$

[Image_UNION]

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

[INTER_EQ_UNIV]

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

[Modus_Ponens]

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

[Mono_speaks_for]

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

[MP_Says]

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

[Quoting]

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

[Quoting_Eq]

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

[reps_def_lemma]

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

[Reps_Eq]

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

[sat_allworld]

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

[sat_andf_eq_and_sat]

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

[sat_TT]

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

[Says]

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

[says_and_lemma]

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

[Speaks_For]

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

[speaks_for_SUBSET]

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

[SUBSET_Image_SUBSET]

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

[Trans_Speaks_For]
$\vdash \forall M\ Oi\ Os\ P\ Q\ R.$
    (M, Oi, Os) `sat` P `speaks_for` $Q \Rightarrow$
    (M, Oi, Os) `sat` Q `speaks_for` $R \Rightarrow$
    (M, Oi, Os) `sat` P `speaks_for` R

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

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

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

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

[world_F]
$\vdash \forall M\ Oi\ Os\ w.\ w \notin$ `Efn` $Oi\ Os\ M$ `FF`

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

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

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

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

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

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

[world_T]
$\vdash \forall M\ Oi\ Os\ w.\ w \in$ `Efn` $Oi\ Os\ M$ `TT`

4   aclDrules Theory

Built: 25 February 2018
Parent Theories: aclrules

4.1   Theorems

[Conjunction]
$\vdash \forall M \ Oi \ Os \ f_1 \ f_2.$
     $(M, Oi, Os) \ \text{sat} \ f_1 \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ f_2 \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ f_1 \ \text{andf} \ f_2$

[Controls]
$\vdash \forall M \ Oi \ Os \ P \ f.$
     $(M, Oi, Os) \ \text{sat} \ P \ \text{says} \ f \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ P \ \text{controls} \ f \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ f$

[Derived_Controls]
$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$
     $(M, Oi, Os) \ \text{sat} \ P \ \text{speaks\_for} \ Q \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ Q \ \text{controls} \ f \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ P \ \text{controls} \ f$

[Derived_Speaks_For]
$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$
     $(M, Oi, Os) \ \text{sat} \ P \ \text{speaks\_for} \ Q \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ P \ \text{says} \ f \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ Q \ \text{says} \ f$

[Disjunction1]
$\vdash \forall M \ Oi \ Os \ f_1 \ f_2. \ (M, Oi, Os) \ \text{sat} \ f_1 \Rightarrow (M, Oi, Os) \ \text{sat} \ f_1 \ \text{orf} \ f_2$

[Disjunction2]
$\vdash \forall M \ Oi \ Os \ f_1 \ f_2. \ (M, Oi, Os) \ \text{sat} \ f_2 \Rightarrow (M, Oi, Os) \ \text{sat} \ f_1 \ \text{orf} \ f_2$

[Disjunctive_Syllogism]
$\vdash \forall M \ Oi \ Os \ f_1 \ f_2.$
     $(M, Oi, Os) \ \text{sat} \ f_1 \ \text{orf} \ f_2 \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ \text{notf} \ f_1 \Rightarrow$
     $(M, Oi, Os) \ \text{sat} \ f_2$

[Double_Negation]
$\vdash \forall M \ Oi \ Os \ f. \ (M, Oi, Os) \ \text{sat} \ \text{notf} \ (\text{notf} \ f) \Rightarrow (M, Oi, Os) \ \text{sat} \ f$

[eqn_eqn]

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

[eqn_lt]

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

[eqn_lte]

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

[Hypothetical_Syllogism]

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

[il_domi]

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

[INTER_EQ_UNIV]

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

[Modus_Tollens]

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

[Rep_Controls_Eq]

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

[Rep_Says]

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

[Reps]

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

[Says_Simplification1]

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

[Says_Simplification2]

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

[Simplification1]

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

[Simplification2]

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

[sl_doms]

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

Index

Appendix B

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

Contents

1 OMNIType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

1.1 Datatypes

command = ESCc escCommand | SLc 'slCommand

escCommand = returnToBase | changeMission | resupply
        | reactToContact

escOutput = ReturnToBase | ChangeMission | Resupply
        | ReactToContact

escState = RTB | CM | RESUPPLY | RTC

output = ESCo escOutput | SLo 'slOutput

principal = SR 'stateRole

state = ESCs escState | SLs 'slState

1.2 Theorems

[command_distinct_clauses]

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

[command_one_one]

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

[escCommand_distinct_clauses]

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

[escOutput_distinct_clauses]

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

[escState_distinct_clauses]

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

[output_distinct_clauses]

$\vdash \forall a'\ a.$ ESCo $a \neq$ SLo a'

[output_one_one]

$\vdash (\forall a\ a'.$ (ESCo a = ESCo a') $\iff$ (a = a')) $\land$
    $\forall a\ a'.$ (SLo a = SLo a') $\iff$ (a = a')

[principal_one_one]

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

[state_distinct_clauses]

$\vdash \forall a'\ a.$ ESCs $a \neq$ SLs a'

[state_one_one]

$\vdash (\forall a\ a'.$ (ESCs a = ESCs a') $\iff$ (a = a')) $\land$
    $\forall a\ a'.$ (SLs a = 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]

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

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

2.3 Theorems

[CFGInterpret_def]

$\vdash$ CFGInterpret (M, Oi, Os)
(CFG *authenticationTest* *stateInterp* *securityContext*
(*input*::*ins*) *state* *outputStream*) $\iff$

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

[CFGInterpret_ind]

$\vdash \forall P.$
  $(\forall M\ Oi\ Os\ authenticationTest\ stateInterp\ securityContext$
      $input\ ins\ state\ outputStream.$
      $P\ (M, Oi, Os)$
        $($`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]

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

[order_one_one]

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

[TR_cases]

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

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

[TR_discard_cmd_rule]

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

[TR_EQ_rules_thm]

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

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

[**TR_exec_cmd_rule**]

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

[**TR_ind**]

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

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

[TR_rules]

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

[TR_strongind]

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

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

[TR_trap_cmd_rule]
 $\vdash$ $\forall$ $authenticationTest$ $stateInterp$ $securityContext$ P cmd ins s
    $outs$.
   ($\forall$ M Oi Os .
    CFGInterpret (M, Oi, Os)
      (CFG $authenticationTest$ $stateInterp$ $securityContext$
        (P says prop (SOME cmd)::ins) s $outs$) $\Rightarrow$
    (M, Oi, Os) sat prop NONE) $\Rightarrow$
  $\forall$ NS Out M Oi Os .
    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 $outs$) $\wedge$
    (M, Oi, Os) sat prop NONE

[TRrule0]
 $\vdash$ TR (M, Oi, Os) (exec cmd)
    (CFG $authenticationTest$ $stateInterp$ $securityContext$

$(P$ says prop (SOME $cmd)::ins)$ s $outs)$
   (CFG *authenticationTest* *stateInterp* *securityContext* *ins*
      $(NS$ s (exec $cmd))$ $(Out$ s (exec $cmd)::outs))$ $\iff$
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]

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

[trType_one_one]

$\vdash$ $(\forall a\ a'.$ (discard a = discard $a')$ $\iff$ $(a = a'))$ $\wedge$
   $(\forall a\ a'.$ (trap a = trap $a')$ $\iff$ $(a = a'))$ $\wedge$
   $\forall a\ a'.$ (exec a = 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]

$\vdash \forall \, elementTest \ x.$
    `authenticationTest` $elementTest \ x \iff$
    `FOLDR` $(\lambda \, p \ q. \ p \land q)$ `T` `(MAP` $elementTest \ x)$

[commandList_def]

$\vdash \forall \, x.$ `commandList` x `= MAP extractCommand` x

[inputList_def]

$\vdash \forall \, xs.$ `inputList` xs `= MAP extractInput` xs

[propCommandList_def]

$\vdash \forall \, x.$ `propCommandList` x `= MAP extractPropCommand` x

[TR_def]

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

$$outs)) \lor$$
$$(\exists \, elementTest \ NS \ M \ Oi \ Os \ Out \ s \ context \ stateInterp \ x$$
$$ins \ outs.$$
$$(a_0 = (M, Oi, Os)) \land (a_1 = \mathtt{discard \ (inputList} \ x)) \land$$
$$(a_2 =$$
$$\mathtt{CFG} \ elementTest \ stateInterp \ context \ (x::ins) \ s$$
$$outs) \land$$
$$(a_3 =$$
$$\mathtt{CFG} \ elementTest \ stateInterp \ context \ ins$$
$$(NS \ s \ (\mathtt{discard \ (inputList} \ x))))$$
$$(Out \ s \ (\mathtt{discard \ (inputList} \ x))::outs)) \land$$
$$\neg \mathtt{authenticationTest} \ elementTest \ x) \Rightarrow$$
$$TR' \ a_0 \ a_1 \ a_2 \ a_3) \Rightarrow$$
$$TR' \ a_0 \ a_1 \ a_2 \ a_3)$$

3.3 Theorems

[CFGInterpret_def]

$\vdash$ CFGInterpret (M, Oi, Os)
    (CFG *elementTest* *stateInterp* *context* $(x::ins)$ *state*
       outStream) $\iff$
  (M, Oi, Os) satList *context* x $\land$ (M, Oi, Os) satList x $\land$
  (M, Oi, Os) satList *stateInterp* *state* x

[CFGInterpret_ind]

$\vdash \forall P.$
    $(\forall M \ Oi \ Os \ elementTest \ stateInterp \ context \ x \ ins \ state$
      outStream.
     $P \ (M, Oi, Os)$
       (CFG *elementTest* *stateInterp* *context* $(x::ins)$ *state*
         outStream)) $\land$
    $(\forall v_{15} \ v_{10} \ v_{11} \ v_{12} \ v_{13} \ v_{14}.$
      $P \ v_{15} \ (\mathtt{CFG} \ v_{10} \ v_{11} \ v_{12} \ \mathtt{[]} \ v_{13} \ v_{14})) \Rightarrow$
    $\forall v \ v_1 \ v_2 \ v_3. \ P \ (v, v_1, v_2) \ v_3$

[configuration_one_one]

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

[extractCommand_def]

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

[extractCommand_ind]

$\vdash \forall P'.$
    $(\forall P \ cmd. \ P' \ (P \ \mathtt{says} \ \mathtt{prop} \ (\mathtt{SOME} \ cmd))) \land P' \ \mathtt{TT} \land P' \ \mathtt{FF} \land$
    $(\forall v_1. \ P' \ (\mathtt{prop} \ v_1)) \land (\forall v_3. \ P' \ (\mathtt{notf} \ v_3)) \land$

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

[extractInput_def]

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

[extractInput_ind]

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

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

[extractPropCommand_def]

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

[extractPropCommand_ind]

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

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

[TR_cases]

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

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

[TR_discard_cmd_rule]

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

[TR_EQ_rules_thm]

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

[TR_exec_cmd_rule]

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

     (NS s (`exec` (`inputList` x)))
     (Out s (`exec` (`inputList` x))::$outs$)) $\Longleftrightarrow$
  `authenticationTest` *elementTest* x $\wedge$
  `CFGInterpret` (M, Oi, Os)
   (`CFG` *elementTest* *stateInterp* *context* (x::*ins*) s *outs*) $\wedge$
  (M, Oi, Os) `satList` `propCommandList` x

[TR_ind]

$\vdash$ $\forall$ *TR'*.
   ($\forall$ *elementTest* *NS* *M* *Oi* *Os* *Out* s *context* *stateInterp* x *ins*
     outs.
    `authenticationTest` *elementTest* x $\wedge$
    `CFGInterpret` (M, Oi, Os)
     (`CFG` *elementTest* *stateInterp* *context* (x::*ins*) s
      outs) $\Rightarrow$
    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*))) $\wedge$
   ($\forall$ *elementTest* *NS* *M* *Oi* *Os* *Out* s *context* *stateInterp* x *ins*
     outs.
    `authenticationTest` *elementTest* x $\wedge$
    `CFGInterpret` (M, Oi, Os)
     (`CFG` *elementTest* *stateInterp* *context* (x::*ins*) s
      outs) $\Rightarrow$
    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*))) $\wedge$
   ($\forall$ *elementTest* *NS* *M* *Oi* *Os* *Out* s *context* *stateInterp* x *ins*
     outs.
    $\neg$`authenticationTest` *elementTest* x $\Rightarrow$
    TR' (M, Oi, Os) (`discard` (`inputList` x))
     (`CFG` *elementTest* *stateInterp* *context* (x::*ins*) s *outs*)
     (`CFG` *elementTest* *stateInterp* *context* *ins*
      (NS s (`discard` (`inputList` x)))
      (Out s (`discard` (`inputList` x))::*outs*))) $\Rightarrow$
   $\forall$ a_0 a_1 a_2 a_3. `TR` a_0 a_1 a_2 a_3 $\Rightarrow$ *TR'* a_0 a_1 a_2 a_3

[TR_rules]

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

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

[TR_strongind]

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

```
            (CFG elementTest stateInterp context (x::ins) s outs)
            (CFG elementTest stateInterp context ins
               (NS s (discard (inputList x)))
               (Out s (discard (inputList x))::outs))) ⇒
       ∀ a₀  a₁  a₂  a₃. TR  a₀  a₁  a₂  a₃  ⇒  TR′ a₀  a₁  a₂  a₃
```

[TR_trap_cmd_rule]

```
 ⊢ ∀ elementTest context stateInterp x ins s outs.
     (∀ M  Oi  Os.
        CFGInterpret (M, Oi, Os)
          (CFG elementTest stateInterp context (x::ins) s
             outs) ⇒
        (M, Oi, Os) sat prop NONE) ⇒
     ∀ NS  Out  M  Oi  Os.
       TR (M, Oi, Os) (trap (inputList x))
         (CFG elementTest stateInterp context (x::ins) s outs)
         (CFG elementTest stateInterp context ins
            (NS s (trap (inputList x)))
            (Out s (trap (inputList x))::outs)) ⟺
       authenticationTest elementTest x ∧
       CFGInterpret (M, Oi, Os)
         (CFG elementTest stateInterp context (x::ins) s outs) ∧
       (M, Oi, Os) sat prop NONE
```

[TRrule0]

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

[TRrule1]

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

[trType_distinct_clauses]

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

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

4   satList Theory

Built: 13 May 2018
Parent Theories: aclDrules

4.1   Definitions

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

4.2   Theorems

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

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

[satList_nil]
⊢ (M,Oi,Os) satList []

5   ssmPB Theory

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

5.1   Definitions

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

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

5.2   Theorems

[authenticationTest_cmd_reject_lemma]

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

[authenticationTest_def]

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

[authenticationTest_ind]

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

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

[PBNS_def]

$\vdash$ (PBNS PLAN_PB (exec (SLc crossLD)) = MOVE_TO_ORP) $\wedge$
  (PBNS PLAN_PB (exec (SLc incomplete)) = PLAN_PB) $\wedge$
  (PBNS MOVE_TO_ORP (exec (SLc conductORP)) = CONDUCT_ORP) $\wedge$
  (PBNS MOVE_TO_ORP (exec (SLc incomplete)) = MOVE_TO_ORP) $\wedge$
  (PBNS CONDUCT_ORP (exec (SLc moveToPB)) = MOVE_TO_PB) $\wedge$
  (PBNS CONDUCT_ORP (exec (SLc incomplete)) = CONDUCT_ORP) $\wedge$
  (PBNS MOVE_TO_PB (exec (SLc conductPB)) = CONDUCT_PB) $\wedge$
  (PBNS MOVE_TO_PB (exec (SLc incomplete)) = MOVE_TO_PB) $\wedge$
  (PBNS CONDUCT_PB (exec (SLc completePB)) = COMPLETE_PB) $\wedge$
  (PBNS CONDUCT_PB (exec (SLc incomplete)) = CONDUCT_PB) $\wedge$
  (PBNS s (trap (SLc cmd)) = s) $\wedge$
  (PBNS s (discard (SLc cmd)) = s)

[PBNS_ind]

$\vdash \forall P.$
  P PLAN_PB (exec (SLc crossLD)) $\wedge$

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

[PBOut_def]

$\vdash$ `(PBOut PLAN_PB (exec (SLc crossLD))) = MoveToORP)` $\wedge$
`(PBOut PLAN_PB (exec (SLc incomplete))) = PlanPB)` $\wedge$
`(PBOut MOVE_TO_ORP (exec (SLc conductORP))) = ConductORP)` $\wedge$
`(PBOut MOVE_TO_ORP (exec (SLc incomplete))) = MoveToORP)` $\wedge$
`(PBOut CONDUCT_ORP (exec (SLc moveToPB))) = MoveToPB)` $\wedge$
`(PBOut CONDUCT_ORP (exec (SLc incomplete))) = ConductORP)` $\wedge$
`(PBOut MOVE_TO_PB (exec (SLc conductPB))) = ConductPB)` $\wedge$

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

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

[PlatoonLeader_exec_slCommand_justified_thm]

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

[PlatoonLeader_slCommand_lemma]

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

6   PBTypeIntegrated Theory

Built: 13 May 2018
Parent Theories: OMNIType

6.1   Datatypes

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

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

slCommand = PL PBTypeIntegrated$plCommand | OMNI omniCommand

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

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

stateRole = PlatoonLeader | Omni

6.2   Theorems

[omniCommand_distinct_clauses]

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

[plCommand_distinct_clauses]

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

[slCommand_distinct_clauses]

⊢ ∀ a' a. PL a ≠ OMNI a'

[slCommand_one_one]

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

[slOutput_distinct_clauses]

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

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

[slState_distinct_clauses]

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

[stateRole_distinct_clauses]

```
⊢ PlatoonLeader ≠ Omni
```

7   PBIntegratedDef Theory

Built: 13 May 2018

Parent Theories: PBTypeIntegrated, aclfoundation

7.1   Definitions

[secAuthorization_def]

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

[secHelper_def]

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

7.2   Theorems

[getOmniCommand_def]

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

($\forall\, xs$. getOmniCommand (FF::xs) = getOmniCommand xs) $\land$
($\forall\, xs\ v_2$. getOmniCommand (prop v_2::xs) = getOmniCommand xs) $\land$
($\forall\, xs\ v_3$. getOmniCommand (notf v_3::xs) = getOmniCommand xs) $\land$
($\forall\, xs\ v_5\ v_4$.
   getOmniCommand (v_4 andf v_5::xs) = getOmniCommand xs) $\land$
($\forall\, xs\ v_7\ v_6$.
   getOmniCommand (v_6 orf v_7::xs) = getOmniCommand xs) $\land$
($\forall\, xs\ v_9\ v_8$.
   getOmniCommand (v_8 impf v_9::xs) = getOmniCommand xs) $\land$
($\forall\, xs\ v_{11}\ v_{10}$.
   getOmniCommand (v_{10} eqf v_{11}::xs) = getOmniCommand xs) $\land$
($\forall\, xs\ v_{13}\ v_{12}$.
   getOmniCommand (v_{12} says v_{13}::xs) = getOmniCommand xs) $\land$
($\forall\, xs\ v_{15}\ v_{14}$.
   getOmniCommand (v_{14} speaks_for v_{15}::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v_{16}$.
   getOmniCommand (v_{16} controls TT::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v_{16}$.
   getOmniCommand (v_{16} controls FF::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v134$.
   getOmniCommand (Name $v134$ controls prop NONE::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v144$.
   getOmniCommand
    (Name PlatoonLeader controls prop (SOME $v144$)::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v146$.
   getOmniCommand
    (Name Omni controls prop (SOME (ESCc $v146$))::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v150$.
   getOmniCommand
    (Name Omni controls prop (SOME (SLc (PL $v150$)))::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v_{68}\ v136\ v135$.
   getOmniCommand ($v135$ meet $v136$ controls prop v_{68}::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v_{68}\ v138\ v137$.
   getOmniCommand ($v137$ quoting $v138$ controls prop v_{68}::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v_{69}\ v_{16}$.
   getOmniCommand (v_{16} controls notf v_{69}::xs) =
   getOmniCommand xs) $\land$
($\forall\, xs\ v_{71}\ v_{70}\ v_{16}$.
   getOmniCommand (v_{16} controls (v_{70} andf v_{71})::xs) =
   getOmniCommand xs) $\land$

$(\forall xs\ v_{73}\ v_{72}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls $(v_{72}$ orf $v_{73})::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{75}\ v_{74}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls $(v_{74}$ impf $v_{75})::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{77}\ v_{76}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls $(v_{76}$ eqf $v_{77})::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{79}\ v_{78}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{78} says $v_{79}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{81}\ v_{80}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{80} speaks_for $v_{81}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{83}\ v_{82}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{82} controls $v_{83}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{86}\ v_{85}\ v_{84}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls reps $v_{84}\ v_{85}\ v_{86}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{88}\ v_{87}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{87} domi $v_{88}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{90}\ v_{89}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{89} eqi $v_{90}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{92}\ v_{91}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{91} doms $v_{92}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{94}\ v_{93}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{93} eqs $v_{94}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{96}\ v_{95}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{95} eqn $v_{96}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{98}\ v_{97}\ v_{16}.$
    getOmniCommand $(v_{16}$ controls v_{97} lte $v_{98}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{99}\ v_{16}\ v100.$
    getOmniCommand $(v_{16}$ controls v_{99} lt $v100::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{20}\ v_{19}\ v_{18}.$
    getOmniCommand (reps $v_{18}\ v_{19}\ v_{20}::xs) =$
    getOmniCommand $xs) \wedge$
$(\forall xs\ v_{22}\ v_{21}.$
    getOmniCommand $(v_{21}$ domi $v_{22}::xs) =$ getOmniCommand $xs) \wedge$
$(\forall xs\ v_{24}\ v_{23}.$
    getOmniCommand $(v_{23}$ eqi $v_{24}::xs) =$ getOmniCommand $xs) \wedge$

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

[getOmniCommand_ind]

$\vdash \forall\, P.$
$\qquad P$ [] $\wedge$
$\qquad (\forall\, cmd \;\; xs.$
$\qquad\qquad P$
$\qquad\qquad\quad$ (Name Omni controls prop (SOME (SLc (OMNI cmd)))::
$\qquad\qquad\qquad\qquad xs$)) $\wedge$ ($\forall\, xs. \; P \;\; xs \Rightarrow P$ (TT::xs)) $\wedge$
$\qquad (\forall\, xs. \;\; P \;\; xs \Rightarrow P$ (FF::xs)) $\wedge$
$\qquad (\forall\, v_2 \;\; xs. \;\; P \;\; xs \Rightarrow P$ (prop $v_2::xs$)) $\wedge$
$\qquad (\forall\, v_3 \;\; xs. \;\; P \;\; xs \Rightarrow P$ (notf $v_3::xs$)) $\wedge$
$\qquad (\forall\, v_4 \;\; v_5 \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_4 andf $v_5::xs$)) $\wedge$
$\qquad (\forall\, v_6 \;\; v_7 \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_6 orf $v_7::xs$)) $\wedge$
$\qquad (\forall\, v_8 \;\; v_9 \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_8 impf $v_9::xs$)) $\wedge$
$\qquad (\forall\, v_{10} \;\; v_{11} \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_{10} eqf $v_{11}::xs$)) $\wedge$
$\qquad (\forall\, v_{12} \;\; v_{13} \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_{12} says $v_{13}::xs$)) $\wedge$
$\qquad (\forall\, v_{14} \;\; v_{15} \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_{14} speaks_for $v_{15}::xs$)) $\wedge$
$\qquad (\forall\, v_{16} \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_{16} controls TT::xs)) $\wedge$
$\qquad (\forall\, v_{16} \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_{16} controls FF::xs)) $\wedge$
$\qquad (\forall\, v134 \;\; xs. \;\; P \;\; xs \Rightarrow P$ (Name $v134$ controls prop NONE::xs)) $\wedge$
$\qquad (\forall\, v144 \;\; xs.$
$\qquad\qquad P \;\; xs \Rightarrow$
$\qquad\qquad P$ (Name PlatoonLeader controls prop (SOME $v144$)::xs)) $\wedge$
$\qquad (\forall\, v146 \;\; xs.$
$\qquad\qquad P \;\; xs \Rightarrow$
$\qquad\qquad P$ (Name Omni controls prop (SOME (ESCc $v146$))::xs)) $\wedge$
$\qquad (\forall\, v150 \;\; xs.$
$\qquad\qquad P \;\; xs \Rightarrow$
$\qquad\qquad P$
$\qquad\qquad\quad$ (Name Omni controls prop (SOME (SLc (PL $v150$)))::
$\qquad\qquad\qquad\qquad xs$)) $\wedge$
$\qquad (\forall\, v135 \;\; v136 \;\; v_{68} \;\; xs.$
$\qquad\qquad P \;\; xs \Rightarrow P$ ($v135$ meet $v136$ controls prop $v_{68}::xs$)) $\wedge$
$\qquad (\forall\, v137 \;\; v138 \;\; v_{68} \;\; xs.$
$\qquad\qquad P \;\; xs \Rightarrow P$ ($v137$ quoting $v138$ controls prop $v_{68}::xs$)) $\wedge$
$\qquad (\forall\, v_{16} \;\; v_{69} \;\; xs. \;\; P \;\; xs \Rightarrow P$ (v_{16} controls notf $v_{69}::xs$)) $\wedge$
$\qquad (\forall\, v_{16} \;\; v_{70} \;\; v_{71} \;\; xs.$
$\qquad\qquad P \;\; xs \Rightarrow P$ (v_{16} controls (v_{70} andf v_{71})::xs)) $\wedge$
$\qquad (\forall\, v_{16} \;\; v_{72} \;\; v_{73} \;\; xs.$

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

[secContext_def]

$\vdash$ (secContext PLAN_PB $(x\texttt{::}xs)$ =
   [prop (SOME (SLc (OMNI ssmPlanPBComplete))) impf
   Name PlatoonLeader controls
   prop (SOME (SLc (PL crossLD)))]) $\wedge$
   (secContext MOVE_TO_ORP $(x\texttt{::}xs)$ =
   [prop (SOME (SLc (OMNI ssmMoveToORPComplete))) impf
   Name PlatoonLeader controls
   prop (SOME (SLc (PL conductORP)))]) $\wedge$
   (secContext CONDUCT_ORP $(x\texttt{::}xs)$ =
   [prop (SOME (SLc (OMNI ssmConductORPComplete))) impf
   Name PlatoonLeader controls

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

[secContext_ind]

⊢ ∀ *P*.
    (∀ *x* *xs*. *P* PLAN_PB (*x*::*xs*)) ∧
    (∀ *x* *xs*. *P* MOVE_TO_ORP (*x*::*xs*)) ∧
    (∀ *x* *xs*. *P* CONDUCT_ORP (*x*::*xs*)) ∧
    (∀ *x* *xs*. *P* MOVE_TO_PB (*x*::*xs*)) ∧
    (∀ *x* *xs*. *P* CONDUCT_PB (*x*::*xs*)) ∧ (∀ v_4. *P* v_4 []) ∧
    (∀ v_5 v_6. *P* COMPLETE_PB (v_5::v_6)) ⇒
    ∀ *v* v_1. *P* *v* v_1

8 ssmConductORP Theory

Built: 13 May 2018

Parent Theories: ConductORPType, ssm11, OMNIType

8.1 Definitions

[secContextConductORP_def]

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

[ssmConductORPStateInterp_def]

⊢ ∀ *slState*. ssmConductORPStateInterp *slState* = TT

8.2 Theorems

[authTestConductORP_cmd_reject_lemma]

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

[authTestConductORP_def]

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

[authTestConductORP_ind]

⊢ ∀ P.
    (∀ cmd. P (Name PlatoonLeader says prop cmd)) ∧
    (∀ cmd. P (Name PlatoonSergeant says prop cmd)) ∧ P TT ∧
    P FF ∧ (∀ v. P (prop v)) ∧ (∀ v_1. P (notf v_1)) ∧
    (∀ v_2 v_3. P (v_2 andf v_3)) ∧ (∀ v_4 v_5. P (v_4 orf v_5)) ∧

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

[conductORPNS_def]

$\vdash$ (conductORPNS CONDUCT_ORP (exec (PL secure)) = SECURE) $\wedge$
  (conductORPNS CONDUCT_ORP (exec (PL plIncomplete)) =
  CONDUCT_ORP) $\wedge$
  (conductORPNS SECURE (exec (PSG actionsIn)) = ACTIONS_IN) $\wedge$
  (conductORPNS SECURE (exec (PSG psgIncomplete)) = SECURE) $\wedge$
  (conductORPNS ACTIONS_IN (exec (PL withdraw)) = WITHDRAW) $\wedge$
  (conductORPNS ACTIONS_IN (exec (PL plIncomplete)) =
  ACTIONS_IN) $\wedge$
  (conductORPNS WITHDRAW (exec (PL complete)) = COMPLETE) $\wedge$
  (conductORPNS WITHDRAW (exec (PL plIncomplete)) = WITHDRAW) $\wedge$
  (conductORPNS s (trap (PL cmd')) = s) $\wedge$
  (conductORPNS s (trap (PSG cmd)) = s) $\wedge$
  (conductORPNS s (discard (PL cmd')) = s) $\wedge$
  (conductORPNS s (discard (PSG cmd)) = s)

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

[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)) $\wedge$

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

[PlatoonLeader_exec_plCommand_justified_thm]

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

[PlatoonLeader_plCommand_lemma]

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

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

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

9 ConductORPType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

9.1 Datatypes

plCommand = secure | withdraw | complete | plIncomplete

psgCommand = actionsIn | psgIncomplete

slCommand =
   PL ConductORPType$plCommand
  | PSG ConductORPType$psgCommand

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

slState = CONDUCT_ORP | SECURE | ACTIONS_IN | WITHDRAW
     | COMPLETE

stateRole = PlatoonLeader | PlatoonSergeant

9.2 Theorems

[plCommand_distinct_clauses]

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

[psgCommand_distinct_clauses]

$\vdash$ actionsIn $\neq$ psgIncomplete

[slCommand_distinct_clauses]

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

[slCommand_one_one]

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

[slOutput_distinct_clauses]

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

[slRole_distinct_clauses]

$\vdash$ PlatoonLeader $\neq$ PlatoonSergeant

[slState_distinct_clauses]

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

10 ssmConductPB Theory

Built: 13 May 2018

Parent Theories: ConductPBType, ssm11, OMNIType

10.1   Definitions

[secContextConductPB_def]

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

[ssmConductPBStateInterp_def]

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

10.2   Theorems

[authTestConductPB_cmd_reject_lemma]

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

[authTestConductPB_def]

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

```
(authTestConductPB (v₁₀ says v₉₁ eqs v₉₂)  ⟺  F) ∧
(authTestConductPB (v₁₀ says v₉₃ eqn v₉₄)  ⟺  F) ∧
(authTestConductPB (v₁₀ says v₉₅ lte v₉₆)  ⟺  F) ∧
(authTestConductPB (v₁₀ says v₉₇ lt v₉₈)   ⟺  F) ∧
(authTestConductPB (v₁₂ speaks_for v₁₃)   ⟺  F) ∧
(authTestConductPB (v₁₄ controls v₁₅)     ⟺  F) ∧
(authTestConductPB (reps v₁₆ v₁₇ v₁₈)     ⟺  F) ∧
(authTestConductPB (v₁₉ domi v₂₀)         ⟺  F) ∧
(authTestConductPB (v₂₁ eqi v₂₂)          ⟺  F) ∧
(authTestConductPB (v₂₃ doms v₂₄)         ⟺  F) ∧
(authTestConductPB (v₂₅ eqs v₂₆)          ⟺  F) ∧
(authTestConductPB (v₂₇ eqn v₂₈)          ⟺  F) ∧
(authTestConductPB (v₂₉ lte v₃₀)          ⟺  F) ∧
(authTestConductPB (v₃₁ lt v₃₂)           ⟺  F)
```

[authTestConductPB_ind]

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

$\forall\, v.\ P\ v$

[conductPBNS_def]

⊢ (conductPBNS CONDUCT_PB (exec (PL securePB)) = SECURE_PB) ∧
  (conductPBNS CONDUCT_PB (exec (PL plIncompletePB)) =
   CONDUCT_PB) ∧
  (conductPBNS SECURE_PB (exec (PSG actionsInPB)) =
   ACTIONS_IN_PB) ∧
  (conductPBNS SECURE_PB (exec (PSG psgIncompletePB)) =
   SECURE_PB) ∧
  (conductPBNS ACTIONS_IN_PB (exec (PL withdrawPB)) =
   WITHDRAW_PB) ∧
  (conductPBNS ACTIONS_IN_PB (exec (PL plIncompletePB)) =
   ACTIONS_IN_PB) ∧
  (conductPBNS WITHDRAW_PB (exec (PL completePB)) =
   COMPLETE_PB) ∧
  (conductPBNS WITHDRAW_PB (exec (PL plIncompletePB)) =
   WITHDRAW_PB) ∧ (conductPBNS s (trap (PL cmd')) = s) ∧
  (conductPBNS s (trap (PSG cmd)) = s) ∧
  (conductPBNS s (discard (PL cmd')) = s) ∧
  (conductPBNS s (discard (PSG cmd)) = s)

[conductPBNS_ind]

⊢ ∀ P.
    P CONDUCT_PB (exec (PL securePB)) ∧
    P CONDUCT_PB (exec (PL plIncompletePB)) ∧
    P SECURE_PB (exec (PSG actionsInPB)) ∧
    P SECURE_PB (exec (PSG psgIncompletePB)) ∧
    P ACTIONS_IN_PB (exec (PL withdrawPB)) ∧
    P ACTIONS_IN_PB (exec (PL plIncompletePB)) ∧
    P WITHDRAW_PB (exec (PL completePB)) ∧
    P WITHDRAW_PB (exec (PL plIncompletePB)) ∧
    (∀ s cmd. P s (trap (PL cmd))) ∧
    (∀ s cmd. P s (trap (PSG cmd))) ∧
    (∀ s cmd. P s (discard (PL cmd))) ∧
    (∀ s cmd. P s (discard (PSG cmd))) ∧
    P CONDUCT_PB (exec (PL withdrawPB)) ∧
    P CONDUCT_PB (exec (PL completePB)) ∧
    (∀ v_{11}. P CONDUCT_PB (exec (PSG v_{11}))) ∧
    (∀ v_{13}. P SECURE_PB (exec (PL v_{13}))) ∧
    P ACTIONS_IN_PB (exec (PL securePB)) ∧
    P ACTIONS_IN_PB (exec (PL completePB)) ∧
    (∀ v_{17}. P ACTIONS_IN_PB (exec (PSG v_{17}))) ∧
    P WITHDRAW_PB (exec (PL securePB)) ∧
    P WITHDRAW_PB (exec (PL withdrawPB)) ∧
    (∀ v_{20}. P WITHDRAW_PB (exec (PSG v_{20}))) ∧
    (∀ v_{21}. P COMPLETE_PB (exec v_{21})) ⇒
    ∀ v v_1. P v v_1

[conductPBOut_def]

⊢ (conductPBOut CONDUCT_PB (exec (PL securePB)) = ConductPB) ∧
  (conductPBOut CONDUCT_PB (exec (PL plIncompletePB)) =
  ConductPB) ∧
  (conductPBOut SECURE_PB (exec (PSG actionsInPB)) =
  SecurePB) ∧
  (conductPBOut SECURE_PB (exec (PSG psgIncompletePB)) =
  SecurePB) ∧
  (conductPBOut ACTIONS_IN_PB (exec (PL withdrawPB)) =
  ActionsInPB) ∧
  (conductPBOut ACTIONS_IN_PB (exec (PL plIncompletePB)) =
  ActionsInPB) ∧
  (conductPBOut WITHDRAW_PB (exec (PL completePB)) =
  WithdrawPB) ∧
  (conductPBOut WITHDRAW_PB (exec (PL plIncompletePB)) =
  WithdrawPB) ∧
  (conductPBOut s (trap (PL cmd')) = unAuthorized) ∧
  (conductPBOut s (trap (PSG cmd)) = unAuthorized) ∧
  (conductPBOut s (discard (PL cmd')) = unAuthenticated) ∧
  (conductPBOut s (discard (PSG cmd)) = unAuthenticated)

[conductPBOut_ind]

⊢ ∀ P.
    P CONDUCT_PB (exec (PL securePB)) ∧
    P CONDUCT_PB (exec (PL plIncompletePB)) ∧
    P SECURE_PB (exec (PSG actionsInPB)) ∧
    P SECURE_PB (exec (PSG psgIncompletePB)) ∧
    P ACTIONS_IN_PB (exec (PL withdrawPB)) ∧
    P ACTIONS_IN_PB (exec (PL plIncompletePB)) ∧
    P WITHDRAW_PB (exec (PL completePB)) ∧
    P WITHDRAW_PB (exec (PL plIncompletePB)) ∧
    (∀ s cmd. P s (trap (PL cmd))) ∧
    (∀ s cmd. P s (trap (PSG cmd))) ∧
    (∀ s cmd. P s (discard (PL cmd))) ∧
    (∀ s cmd. P s (discard (PSG cmd))) ∧
    P CONDUCT_PB (exec (PL withdrawPB)) ∧
    P CONDUCT_PB (exec (PL completePB)) ∧
    (∀ v_{11}. P CONDUCT_PB (exec (PSG v_{11}))) ∧
    (∀ v_{13}. P SECURE_PB (exec (PL v_{13}))) ∧
    P ACTIONS_IN_PB (exec (PL securePB)) ∧
    P ACTIONS_IN_PB (exec (PL completePB)) ∧
    (∀ v_{17}. P ACTIONS_IN_PB (exec (PSG v_{17}))) ∧
    P WITHDRAW_PB (exec (PL securePB)) ∧
    P WITHDRAW_PB (exec (PL withdrawPB)) ∧
    (∀ v_{20}. P WITHDRAW_PB (exec (PSG v_{20}))) ∧
    (∀ v_{21}. P COMPLETE_PB (exec v_{21})) ⇒
    ∀ v v_1. P v v_1

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

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

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

[PlatoonSergeant_psgCommandPB_lemma]

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

$\vdash$ securePB $\neq$ withdrawPB $\wedge$ securePB $\neq$ completePB $\wedge$
  securePB $\neq$ plIncompletePB $\wedge$ withdrawPB $\neq$ completePB $\wedge$
  withdrawPB $\neq$ plIncompletePB $\wedge$ completePB $\neq$ plIncompletePB

[psgCommandPB_distinct_clauses]

$\vdash$ actionsInPB $\neq$ psgIncompletePB

[slCommand_distinct_clauses]

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

[slCommand_one_one]

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

[slOutput_distinct_clauses]

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

[slRole_distinct_clauses]

⊢ PlatoonLeader ≠ PlatoonSergeant

[slState_distinct_clauses]

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

12 ssmMoveToORP Theory

Built: 13 May 2018

Parent Theories: MoveToORPType, ssm11, OMNIType

12.1 Definitions

[secContextMoveToORP_def]

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

[ssmMoveToORPStateInterp_def]

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

12.2 Theorems

[authTestMoveToORP_cmd_reject_lemma]

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

[authTestMoveToORP_def]

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

[authTestMoveToORP_ind]

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

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

[moveToORPNS_def]

⊢ (moveToORPNS MOVE_TO_ORP (exec (SLc pltForm)) = PLT_FORM) ∧
  (moveToORPNS MOVE_TO_ORP (exec (SLc incomplete)) =
   MOVE_TO_ORP) ∧
  (moveToORPNS PLT_FORM (exec (SLc pltMove)) = PLT_MOVE) ∧
  (moveToORPNS PLT_FORM (exec (SLc incomplete)) = PLT_FORM) ∧
  (moveToORPNS PLT_MOVE (exec (SLc pltSecureHalt)) =
   PLT_SECURE_HALT) ∧
  (moveToORPNS PLT_MOVE (exec (SLc incomplete)) = PLT_MOVE) ∧
  (moveToORPNS PLT_SECURE_HALT (exec (SLc complete)) =
   COMPLETE) ∧
  (moveToORPNS PLT_SECURE_HALT (exec (SLc incomplete)) =
   PLT_SECURE_HALT) ∧ (moveToORPNS s (trap (SLc cmd)) = s) ∧
  (moveToORPNS s (discard (SLc cmd)) = s)

[moveToORPNS_ind]

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

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

[moveToORPOut_def]

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

[moveToORPOut_ind]

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

```
P PLT_SECURE_HALT (exec (SLc complete)) ∧
P PLT_SECURE_HALT (exec (SLc incomplete)) ∧
```
$(\forall s\ cmd.\ P\ s\ (\text{trap } (\text{SLc } cmd))) \land$
$(\forall s\ cmd.\ P\ s\ (\text{discard } (\text{SLc } cmd))) \land$
$(\forall s\ v_6.\ P\ s\ (\text{discard } (\text{ESCc } v_6))) \land$
$(\forall s\ v_9.\ P\ s\ (\text{trap } (\text{ESCc } v_9))) \land$
$(\forall v_{12}.\ P\ \text{MOVE\_TO\_ORP } (\text{exec } (\text{ESCc } v_{12}))) \land$
```
P MOVE_TO_ORP (exec (SLc pltMove)) ∧
P MOVE_TO_ORP (exec (SLc pltSecureHalt)) ∧
P MOVE_TO_ORP (exec (SLc complete)) ∧
```
$(\forall v_{15}.\ P\ \text{PLT\_FORM } (\text{exec } (\text{ESCc } v_{15}))) \land$
```
P PLT_FORM (exec (SLc pltForm)) ∧
P PLT_FORM (exec (SLc pltSecureHalt)) ∧
P PLT_FORM (exec (SLc complete)) ∧
```
$(\forall v_{18}.\ P\ \text{PLT\_MOVE } (\text{exec } (\text{ESCc } v_{18}))) \land$
```
P PLT_MOVE (exec (SLc pltForm)) ∧
P PLT_MOVE (exec (SLc pltMove)) ∧
P PLT_MOVE (exec (SLc complete)) ∧
```
$(\forall v_{21}.\ P\ \text{PLT\_SECURE\_HALT } (\text{exec } (\text{ESCc } v_{21}))) \land$
```
P PLT_SECURE_HALT (exec (SLc pltForm)) ∧
P PLT_SECURE_HALT (exec (SLc pltMove)) ∧
P PLT_SECURE_HALT (exec (SLc pltSecureHalt)) ∧
```
$(\forall v_{23}.\ P\ \text{COMPLETE } (\text{exec } v_{23})) \Rightarrow$
$\forall v\ v_1.\ P\ v\ v_1$

[PlatoonLeader_exec_slCommand_justified_thm]

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

[PlatoonLeader_slCommand_lemma]

$\vdash$ CFGInterpret (M, Oi, Os)
     (CFG authTestMoveToORP ssmMoveToORPStateInterp
       (secContextMoveToORP $slCommand$)

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

13 MoveToORPType Theory

Built: 13 May 2018

Parent Theories: indexedLists, patternMatches

13.1 Datatypes

slCommand = pltForm | pltMove | pltSecureHalt | complete
         | incomplete

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

slState = MOVE_TO_ORP | PLT_FORM | PLT_MOVE | PLT_SECURE_HALT
       | COMPLETE

stateRole = PlatoonLeader

13.2 Theorems

[slCommand_distinct_clauses]
⊢ pltForm $\neq$ pltMove $\wedge$ pltForm $\neq$ pltSecureHalt $\wedge$
   pltForm $\neq$ complete $\wedge$ pltForm $\neq$ incomplete $\wedge$
   pltMove $\neq$ pltSecureHalt $\wedge$ pltMove $\neq$ complete $\wedge$
   pltMove $\neq$ incomplete $\wedge$ pltSecureHalt $\neq$ complete $\wedge$
   pltSecureHalt $\neq$ incomplete $\wedge$ complete $\neq$ incomplete

[slOutput_distinct_clauses]
⊢ MoveToORP $\neq$ PLTForm $\wedge$ MoveToORP $\neq$ PLTMove $\wedge$
   MoveToORP $\neq$ PLTSecureHalt $\wedge$ MoveToORP $\neq$ Complete $\wedge$
   MoveToORP $\neq$ unAuthorized $\wedge$ MoveToORP $\neq$ unAuthenticated $\wedge$
   PLTForm $\neq$ PLTMove $\wedge$ PLTForm $\neq$ PLTSecureHalt $\wedge$
   PLTForm $\neq$ Complete $\wedge$ PLTForm $\neq$ unAuthorized $\wedge$
   PLTForm $\neq$ unAuthenticated $\wedge$ PLTMove $\neq$ PLTSecureHalt $\wedge$
   PLTMove $\neq$ Complete $\wedge$ PLTMove $\neq$ unAuthorized $\wedge$
   PLTMove $\neq$ unAuthenticated $\wedge$ PLTSecureHalt $\neq$ Complete $\wedge$
   PLTSecureHalt $\neq$ unAuthorized $\wedge$
   PLTSecureHalt $\neq$ unAuthenticated $\wedge$ Complete $\neq$ unAuthorized $\wedge$
   Complete $\neq$ unAuthenticated $\wedge$ unAuthorized $\neq$ unAuthenticated

[slState_distinct_clauses]
⊢ MOVE_TO_ORP $\neq$ PLT_FORM $\wedge$ MOVE_TO_ORP $\neq$ PLT_MOVE $\wedge$
   MOVE_TO_ORP $\neq$ PLT_SECURE_HALT $\wedge$ MOVE_TO_ORP $\neq$ COMPLETE $\wedge$
   PLT_FORM $\neq$ PLT_MOVE $\wedge$ PLT_FORM $\neq$ PLT_SECURE_HALT $\wedge$
   PLT_FORM $\neq$ COMPLETE $\wedge$ PLT_MOVE $\neq$ PLT_SECURE_HALT $\wedge$
   PLT_MOVE $\neq$ COMPLETE $\wedge$ PLT_SECURE_HALT $\neq$ COMPLETE

14   ssmMoveToPB Theory

Built: 13 May 2018

Parent Theories: MoveToPBType, ssm11, OMNIType

14.1   Definitions

[secContextMoveToPB_def]

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

[ssmMoveToPBStateInterp_def]

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

14.2   Theorems

[authTestMoveToPB_cmd_reject_lemma]

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

[authTestMoveToPB_def]

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

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

[authTestMoveToPB_ind]

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

[moveToPBNS_def]

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

[moveToPBNS_ind]

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

[moveToPBOut_def]

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

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

[moveToPBOut_ind]

$\vdash \forall P.$

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

[PlatoonLeader_exec_slCommand_justified_thm]

$\vdash \forall NS\ Out\ M\ Oi\ Os.$

 TR (M, Oi, Os) (exec (SLc $slCommand$))
  (CFG authTestMoveToPB ssmMoveToPBStateInterp
   (secContextMoveToPB $slCommand$)
   (Name PlatoonLeader says prop (SOME (SLc $slCommand$))::
    ins) s $outs$)
  (CFG authTestMoveToPB ssmMoveToPBStateInterp

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

[PlatoonLeader_slCommand_lemma]

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

15 MoveToPBType Theory

Built: 13 May 2018

Parent Theories: indexedLists, patternMatches

15.1 Datatypes

slCommand = pltForm | pltMove | pltHalt | complete | incomplete

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

slState = MOVE_TO_PB | PLT_FORM | PLT_MOVE | PLT_HALT | COMPLETE

stateRole = PlatoonLeader

15.2 Theorems

[slCommand_distinct_clauses]

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

[slOutput_distinct_clauses]

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

[slState_distinct_clauses]

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

16 ssmPlanPB Theory

Built: 13 May 2018

Parent Theories: PlanPBDef, ssm

16.1 Theorems

[inputOK_def]

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

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

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

[planPBNS_def]

$\vdash$ (planPBNS WARNO (exec x) =
    if
      (getRecon x = [SOME (SLc (PL recon))]) $\wedge$
      (getTenativePlan x = [SOME (SLc (PL tentativePlan))]) $\wedge$
      (getReport x = [SOME (SLc (PL report1))]) $\wedge$
      (getInitMove x = [SOME (SLc (PSG initiateMovement))])
    then
      REPORT1
    else WARNO) $\wedge$
  (planPBNS PLAN_PB (exec x) =
   if getPlCom x = receiveMission **then** RECEIVE_MISSION
   else PLAN_PB) $\wedge$
  (planPBNS RECEIVE_MISSION (exec x) =
   if getPlCom x = warno **then** WARNO **else** RECEIVE_MISSION) $\wedge$
  (planPBNS REPORT1 (exec x) =
   if getPlCom x = completePlan **then** COMPLETE_PLAN
   else REPORT1) $\wedge$
  (planPBNS COMPLETE_PLAN (exec x) =
   if getPlCom x = opoid **then** OPOID **else** COMPLETE_PLAN) $\wedge$
  (planPBNS OPOID (exec x) =
   if getPlCom x = supervise **then** SUPERVISE **else** OPOID) $\wedge$
  (planPBNS SUPERVISE (exec x) =
   if getPlCom x = report2 **then** REPORT2 **else** SUPERVISE) $\wedge$
  (planPBNS REPORT2 (exec x) =
   if getPlCom x = complete **then** COMPLETE **else** REPORT2) $\wedge$
  (planPBNS s (trap v_0) = s) $\wedge$ (planPBNS s (discard v_1) = s)

[planPBNS_ind]

$\vdash \forall P.$
    ($\forall x.\ P$ WARNO (exec x)) $\wedge$ ($\forall x.\ P$ PLAN_PB (exec x)) $\wedge$
    ($\forall x.\ P$ RECEIVE_MISSION (exec x)) $\wedge$
    ($\forall x.\ P$ REPORT1 (exec x)) $\wedge$ ($\forall x.\ P$ COMPLETE_PLAN (exec x)) $\wedge$
    ($\forall x.\ P$ OPOID (exec x)) $\wedge$ ($\forall x.\ P$ SUPERVISE (exec x)) $\wedge$
    ($\forall x.\ P$ REPORT2 (exec x)) $\wedge$ ($\forall s\ v_0.\ P\ s$ (trap v_0)) $\wedge$
    ($\forall s\ v_1.\ P\ s$ (discard v_1)) $\wedge$
    ($\forall v_6.\ P$ TENTATIVE_PLAN (exec v_6)) $\wedge$
    ($\forall v_7.\ P$ INITIATE_MOVEMENT (exec v_7)) $\wedge$
    ($\forall v_8.\ P$ RECON (exec v_8)) $\wedge$ ($\forall v_9.\ P$ COMPLETE (exec v_9)) $\Rightarrow$
    $\forall v\ v_1.\ P\ v\ v_1$

[planPBOut_def]

$\vdash$ (planPBOut WARNO (exec x) =
    if
      (getRecon x = [SOME (SLc (PL recon))]) $\wedge$
      (getTenativePlan x = [SOME (SLc (PL tentativePlan))]) $\wedge$
      (getReport x = [SOME (SLc (PL report1))]) $\wedge$
      (getInitMove x = [SOME (SLc (PSG initiateMovement))])

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**]

$\vdash \forall P.$
   $(\forall x.\ P$ WARNO (exec x)) $\wedge$ ($\forall x.\ P$ PLAN_PB (exec x)) $\wedge$
   $(\forall x.\ P$ RECEIVE_MISSION (exec x)) $\wedge$
   $(\forall x.\ P$ REPORT1 (exec x)) $\wedge$ ($\forall x.\ P$ COMPLETE_PLAN (exec x)) $\wedge$
   $(\forall x.\ P$ OPOID (exec x)) $\wedge$ ($\forall x.\ P$ SUPERVISE (exec x)) $\wedge$
   $(\forall x.\ P$ REPORT2 (exec x)) $\wedge$ ($\forall s\ v_0.\ P\ s$ (trap v_0)) $\wedge$
   $(\forall s\ v_1.\ P\ s$ (discard v_1)) $\wedge$
   $(\forall v_6.\ P$ TENTATIVE_PLAN (exec v_6)) $\wedge$
   $(\forall v_7.\ P$ INITIATE_MOVEMENT (exec v_7)) $\wedge$
   $(\forall v_8.\ P$ RECON (exec v_8)) $\wedge$ ($\forall v_9.\ P$ COMPLETE (exec v_9)) $\Rightarrow$
   $\forall v\ v_1.\ P\ v\ v_1$

[**PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma**]

$\vdash s \neq$ WARNO $\Rightarrow$
  $plCommand \neq$ invalidPlCommand $\Rightarrow$
  $plCommand \neq$ 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))  ⟺
       authenticationTest inputOK
         [Name PlatoonLeader says
          prop (SOME (SLc (PL plCommand)))] ∧
       CFGInterpret (M,Oi,Os)
         (CFG inputOK secContext secContextNull
            ([Name PlatoonLeader says
              prop (SOME (SLc (PL plCommand)))]::ins) s outs) ∧
       (M,Oi,Os) satList
       propCommandList
         [Name PlatoonLeader says
          prop (SOME (SLc (PL plCommand)))]
```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]

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

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma]

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

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

[PlatoonLeader_psgCommand_notDiscard_thm]

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

[PlatoonLeader_trap_psgCommand_justified_lemma]

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

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

[PlatoonLeader_trap_psgCommand_lemma]

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

[PlatoonLeader_WARNO_exec_report1_justified_lemma]

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

```
                        [Name PlatoonLeader says
                         prop (SOME (SLc (PL recon)));
                         Name PlatoonLeader says
                         prop (SOME (SLc (PL tentativePlan)));
                         Name PlatoonSergeant says
                         prop (SOME (SLc (PSG initiateMovement)));
                         Name PlatoonLeader says
                         prop (SOME (SLc (PL report1)))]))))
```
(Out `WARNO`
      (`exec`
        (`inputList`

```
                        [Name PlatoonLeader says
                         prop (SOME (SLc (PL recon)));
                         Name PlatoonLeader says
                         prop (SOME (SLc (PL tentativePlan)));
                         Name PlatoonSergeant says
                         prop (SOME (SLc (PSG initiateMovement)));
                         Name PlatoonLeader says
```
               `prop (SOME (SLc (PL report1)))]))::`$outs$`))` $\iff$

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

[PlatoonLeader_WARNO_exec_report1_justified_thm]

$\vdash \forall NS\ Out\ M\ Oi\ Os.$
    `TR` (M, Oi, Os)

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

[PlatoonLeader_WARNO_exec_report1_lemma]

$\vdash \forall M \ Oi \ Os.$
    CFGInterpret (M, Oi, Os)
      (CFG inputOK secContext secContextNull
        ([Name PlatoonLeader says
          prop (SOME (SLc (PL recon)));
          Name PlatoonLeader says
          prop (SOME (SLc (PL tentativePlan)));
          Name PlatoonSergeant says
          prop (SOME (SLc (PSG initiateMovement)));
          Name PlatoonLeader says
          prop (SOME (SLc (PL report1)))]::ins) WARNO $outs$) $\Rightarrow$
    (M, Oi, Os) satList
    propCommandList
      [Name PlatoonLeader says prop (SOME (SLc (PL recon)));
      Name PlatoonLeader says
      prop (SOME (SLc (PL tentativePlan)));
      Name PlatoonSergeant says
      prop (SOME (SLc (PSG initiateMovement)));
      Name PlatoonLeader says prop (SOME (SLc (PL report1)))]

[PlatoonSergeant_trap_plCommand_justified_lemma]

$\vdash \forall NS \ Out \ M \ Oi \ Os.$
    TR (M, Oi, Os)
      (trap
        (inputList
          [Name PlatoonSergeant says
           prop (SOME (SLc (PL $plCommand$)))]))
      (CFG inputOK secContext secContextNull
        ([Name PlatoonSergeant says
          prop (SOME (SLc (PL $plCommand$)))]::ins) s $outs$)
      (CFG inputOK secContext secContextNull ins
        ($NS \ s$
          (trap
            (inputList
             [Name PlatoonSergeant says
              prop (SOME (SLc (PL $plCommand$)))]))))
        ($Out \ s$
          (trap
            (inputList
             [Name PlatoonSergeant says
              prop (SOME (SLc (PL $plCommand$)))]))::
           $outs$)) $\iff$
    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*) ∧
(*M*,*Oi*,*Os*) sat prop NONE

[PlatoonSergeant_trap_plCommand_justified_thm]

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

[PlatoonSergeant_trap_plCommand_lemma]

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

17 PlanPBType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

17.1 Datatypes

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

psgCommand = initiateMovement | psgIncomplete
        | invalidPsgCommand

slCommand = PL plCommand | PSG psgCommand

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

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

$stateRole$ = PlatoonLeader | PlatoonSergeant

17.2   Theorems

[plCommand_distinct_clauses]

⊢ receiveMission $\neq$ warno $\wedge$ receiveMission $\neq$ tentativePlan $\wedge$
  receiveMission $\neq$ recon $\wedge$ receiveMission $\neq$ report1 $\wedge$
  receiveMission $\neq$ completePlan $\wedge$ receiveMission $\neq$ opoid $\wedge$
  receiveMission $\neq$ supervise $\wedge$ receiveMission $\neq$ report2 $\wedge$
  receiveMission $\neq$ complete $\wedge$ receiveMission $\neq$ plIncomplete $\wedge$
  receiveMission $\neq$ invalidPlCommand $\wedge$ warno $\neq$ tentativePlan $\wedge$
  warno $\neq$ recon $\wedge$ warno $\neq$ report1 $\wedge$ warno $\neq$ completePlan $\wedge$
  warno $\neq$ opoid $\wedge$ warno $\neq$ supervise $\wedge$ warno $\neq$ report2 $\wedge$
  warno $\neq$ complete $\wedge$ warno $\neq$ plIncomplete $\wedge$
  warno $\neq$ invalidPlCommand $\wedge$ tentativePlan $\neq$ recon $\wedge$
  tentativePlan $\neq$ report1 $\wedge$ tentativePlan $\neq$ completePlan $\wedge$
  tentativePlan $\neq$ opoid $\wedge$ tentativePlan $\neq$ supervise $\wedge$
  tentativePlan $\neq$ report2 $\wedge$ tentativePlan $\neq$ complete $\wedge$
  tentativePlan $\neq$ plIncomplete $\wedge$
  tentativePlan $\neq$ invalidPlCommand $\wedge$ recon $\neq$ report1 $\wedge$
  recon $\neq$ completePlan $\wedge$ recon $\neq$ opoid $\wedge$ recon $\neq$ supervise $\wedge$
  recon $\neq$ report2 $\wedge$ recon $\neq$ complete $\wedge$ recon $\neq$ plIncomplete $\wedge$
  recon $\neq$ invalidPlCommand $\wedge$ report1 $\neq$ completePlan $\wedge$
  report1 $\neq$ opoid $\wedge$ report1 $\neq$ supervise $\wedge$ report1 $\neq$ report2 $\wedge$
  report1 $\neq$ complete $\wedge$ report1 $\neq$ plIncomplete $\wedge$
  report1 $\neq$ invalidPlCommand $\wedge$ completePlan $\neq$ opoid $\wedge$
  completePlan $\neq$ supervise $\wedge$ completePlan $\neq$ report2 $\wedge$
  completePlan $\neq$ complete $\wedge$ completePlan $\neq$ plIncomplete $\wedge$
  completePlan $\neq$ invalidPlCommand $\wedge$ opoid $\neq$ supervise $\wedge$
  opoid $\neq$ report2 $\wedge$ opoid $\neq$ complete $\wedge$ opoid $\neq$ plIncomplete $\wedge$
  opoid $\neq$ invalidPlCommand $\wedge$ supervise $\neq$ report2 $\wedge$
  supervise $\neq$ complete $\wedge$ supervise $\neq$ plIncomplete $\wedge$
  supervise $\neq$ invalidPlCommand $\wedge$ report2 $\neq$ complete $\wedge$
  report2 $\neq$ plIncomplete $\wedge$ report2 $\neq$ invalidPlCommand $\wedge$
  complete $\neq$ plIncomplete $\wedge$ complete $\neq$ invalidPlCommand $\wedge$
  plIncomplete $\neq$ invalidPlCommand

[psgCommand_distinct_clauses]

⊢ initiateMovement $\neq$ psgIncomplete $\wedge$
  initiateMovement $\neq$ invalidPsgCommand $\wedge$
  psgIncomplete $\neq$ invalidPsgCommand

[slCommand_distinct_clauses]

⊢ $\forall\, a'\ a.$ PL $a \neq$ PSG a'

[slCommand_one_one]

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

[slOutput_distinct_clauses]

⊢ PlanPB ≠ ReceiveMission ∧ PlanPB ≠ Warno ∧
  PlanPB ≠ TentativePlan ∧ PlanPB ≠ InitiateMovement ∧
  PlanPB ≠ Recon ∧ PlanPB ≠ Report1 ∧ PlanPB ≠ CompletePlan ∧
  PlanPB ≠ Opoid ∧ PlanPB ≠ Supervise ∧ PlanPB ≠ Report2 ∧
  PlanPB ≠ Complete ∧ PlanPB ≠ unAuthenticated ∧
  PlanPB ≠ unAuthorized ∧ ReceiveMission ≠ Warno ∧
  ReceiveMission ≠ TentativePlan ∧
  ReceiveMission ≠ InitiateMovement ∧ ReceiveMission ≠ Recon ∧
  ReceiveMission ≠ Report1 ∧ ReceiveMission ≠ CompletePlan ∧
  ReceiveMission ≠ Opoid ∧ ReceiveMission ≠ Supervise ∧
  ReceiveMission ≠ Report2 ∧ ReceiveMission ≠ Complete ∧
  ReceiveMission ≠ unAuthenticated ∧
  ReceiveMission ≠ unAuthorized ∧ Warno ≠ TentativePlan ∧
  Warno ≠ InitiateMovement ∧ Warno ≠ Recon ∧ Warno ≠ Report1 ∧
  Warno ≠ CompletePlan ∧ Warno ≠ Opoid ∧ Warno ≠ Supervise ∧
  Warno ≠ Report2 ∧ Warno ≠ Complete ∧
  Warno ≠ unAuthenticated ∧ Warno ≠ unAuthorized ∧
  TentativePlan ≠ InitiateMovement ∧ TentativePlan ≠ Recon ∧
  TentativePlan ≠ Report1 ∧ TentativePlan ≠ CompletePlan ∧
  TentativePlan ≠ Opoid ∧ TentativePlan ≠ Supervise ∧
  TentativePlan ≠ Report2 ∧ TentativePlan ≠ Complete ∧
  TentativePlan ≠ unAuthenticated ∧
  TentativePlan ≠ unAuthorized ∧ InitiateMovement ≠ Recon ∧
  InitiateMovement ≠ Report1 ∧
  InitiateMovement ≠ CompletePlan ∧ InitiateMovement ≠ Opoid ∧
  InitiateMovement ≠ Supervise ∧ InitiateMovement ≠ Report2 ∧
  InitiateMovement ≠ Complete ∧
  InitiateMovement ≠ unAuthenticated ∧
  InitiateMovement ≠ unAuthorized ∧ Recon ≠ Report1 ∧
  Recon ≠ CompletePlan ∧ Recon ≠ Opoid ∧ Recon ≠ Supervise ∧
  Recon ≠ Report2 ∧ Recon ≠ Complete ∧
  Recon ≠ unAuthenticated ∧ Recon ≠ unAuthorized ∧
  Report1 ≠ CompletePlan ∧ Report1 ≠ Opoid ∧
  Report1 ≠ Supervise ∧ Report1 ≠ Report2 ∧
  Report1 ≠ Complete ∧ Report1 ≠ unAuthenticated ∧
  Report1 ≠ unAuthorized ∧ CompletePlan ≠ Opoid ∧
  CompletePlan ≠ Supervise ∧ CompletePlan ≠ Report2 ∧
  CompletePlan ≠ Complete ∧ CompletePlan ≠ unAuthenticated ∧
  CompletePlan ≠ unAuthorized ∧ Opoid ≠ Supervise ∧
  Opoid ≠ Report2 ∧ Opoid ≠ Complete ∧
  Opoid ≠ unAuthenticated ∧ Opoid ≠ unAuthorized ∧
  Supervise ≠ Report2 ∧ Supervise ≠ Complete ∧
  Supervise ≠ unAuthenticated ∧ Supervise ≠ unAuthorized ∧
  Report2 ≠ Complete ∧ Report2 ≠ unAuthenticated ∧

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

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_infRules",
          "satListTheory","ssmTheory"];
open TypeBase listTheory ssminfRules optionTheory acl_infRules satListTheory ssmTheory

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

open HolKernel boolLib Parse bossLib
open TypeBase listTheory optionTheory ssminfRules acl_infRules satListTheory
(* ********************** *)
(* 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
'trType =
  discard 'cmdlist | trap 'cmdlist | exec 'cmdlist'

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

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

(* ──────────────────────────────────────────────────────────── *)
(* Define configuration to include the security context within which the  *)
(* inputs are evaluated. The components are as follows: (1) the authentication *)
(* function, (2) the intepretation of the state, (3) the security context,  *)
(* (4) the input stream, (5) the state, and (6) the output stream.  *)
(* ──────────────────────────────────────────────────────────── *)
val _ =
Datatype
'configuration =
 CFG
  (('command option,'principal,'d,'e)Form -> bool)
  (('state -> ('command option,'principal,'d,'e)Form list ->
   ('command option,'principal,'d,'e)Form list))
  ((('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
  (((('command option,'principal,'d,'e)Form) list) list)
  ('state)
  ('output list)'

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

val _ = save_thm("configuration_one_one",configuration_one_one)

(* ──────────────────────────────────────────────────────────── *)
(* The interpretation of configuration is the conjunction of the formulas in  *)
(* 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)';
```

109

```
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:'state)
   (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:'state)
   (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:'state)
   (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]th)) THEN
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]th)) THEN
 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]th)) THEN
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) transition *)
(* 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 outs)   *)
(* are true.                                                            *)
(* ————————————————————————————————————————————————————— *)
val TR_exec_cmd_rule =
TAC_PROOF((([],
``!elementTest context stateInterp (x:('command option,'principal,'d,'e)Form list)
   ins s outs.
 (!M Oi Os.
 (CFGInterpret
  ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po), (Os :'e po))
  (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, 'e)Form list))) ==>
(!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 (inputList x)))
          (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) transition *)
(* occurs if and only if prop TRAP, elementTest, and                   *)
(* CFGInterpret (M, Oi, Os)                                             *)
(*   (CFG elementTest stateInterpret certList (P says prop (CMD cmd)::ins) *)
(*        s outs) are true.                                             *)
(* ————————————————————————————————————————————————————— *)
val TR_trap_cmd_rule =
TAC_PROOF(
([],``!elementTest context stateInterp (x:('command option,'principal,'d,'e)Form list)
   ins s outs.
 (!M Oi Os.
 (CFGInterpret
  ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po), (Os :'e po))
  (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))) ==>
```

112

```
   (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 (inputList x)))
           (Out s (trap (inputList x))::outs)) <=>
   (authenticationTest elementTest x) /\
   (CFGInterpret (M, Oi , Os)
    (CFG elementTest stateInterp context (x:: ins) s outs)) /\
    (M, Oi , Os) sat (prop NONE))''),
REWRITE_TAC[ TRrule1 ] THEN
REPEAT STRIP_TAC THEN
EQ_TAC THEN
REPEAT STRIP_TAC THEN
PROVE_TAC[ ] )

val _ = save_thm("TR_trap_cmd_rule", TR_trap_cmd_rule)

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

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

end (* structure *)
```

C.2   satList

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

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

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

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

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

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

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

val _ = save_thm("satList_nil",satList_nil)

val satList_conj =
TAC_PROOF(
([],
''!l1 l2 M Oi Os.(((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(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(DEPTH_CONV BETA_CONV) THEN
REWRITE_TAC[FOLDR] THEN
CONV_TAC(DEPTH_CONV BETA_CONV) THEN
REWRITE_TAC[GSYM satList_def] THEN
PROVE_TAC[])

val _ = save_thm("satList_conj",satList_conj)

val satList_CONS =
TAC_PROOF((([],
''!h t M Oi Os.(((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'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(DEPTH_CONV BETA_CONV) THEN
REWRITE_TAC[FOLDR] THEN
CONV_TAC(DEPTH_CONV BETA_CONV) THEN
REWRITE_TAC[])

val _ = save_thm("satList_CONS",satList_CONS)

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

end (* structure *)
```

Appendix D

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

D.1 OMNILevel

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


structure OMNIScript = struct

(* ==== Interactive Mode ====
app load ["TypeBase","listTheory", "optionTheory",
          "OMNITypeTheory",
          "acl_infRules","aclDrulesTheory","aclrulesTheory"];
open TypeBase listTheory optionTheory
     OMNITypeTheory
     acl_infRules aclDrulesTheory aclrulesTheory
 ==== End Interactive Mode ====  *)

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

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

val _=
Datatype 'stateRole = Omni'
```

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

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

val _=
Datatype 'slCommand = OMNI omniCommand'


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

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




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

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




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

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

val _= export_theory();
end
```

D.2   TopLevel

D.2.1   PBTypeIntegrated Theory: Type Definitions

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

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



open HolKernel Parse boolLib bossLib;
open TypeBase OMNITypeTheory

val _= new_theory "PBTypeIntegrated";

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

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


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

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



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


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



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



val _=
Datatype 'stateRole = PlatoonLeader | Omni'

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



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

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

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

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


val _= export_theory();
end
```

D.2.2 PBIntegratedDef Theory: Authentication & Authorization Definitions

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

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

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

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

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

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

val getOmniCommand_def =
Define '
  (getOmniCommand ([]:((slCommand command)option, stateRole, 'd,'e)Form 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))'
```

118

```
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)::xs) =
        [(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)::xs) =
        [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)::xs) =
        [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)::xs) =
        [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)::xs) =
        [prop (SOME (SLc (OMNI (ssmConductPBComplete)))) impf
          (Name PlatoonLeader) controls prop (SOME (SLc (PL completePB)))]) '

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

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

val _= export_theory();
end
```

D.2.3   ssmPlanPBIntegrated Theory: Theorems

```
(***************************************************************************)
(* ssmPBIntegratedTheory                                                   *)
(* Author: Lori Pickering                                                  *)
(* Date: 7 May 2018                                                        *)
(* This theory aims to integrate the topLevel ssm with the sublevel ssms. It *)
(* does this by adding a condition to the security context. In particular,   *)
(* it requires that the "COMPLETE" state in the subLevel ssm must preceede   *)
(* transition to the next state at the topLeve. I.e.,                        *)
(*    planPBComplete ==>                                                      *)
(*    PlatoonLeader controls crossLD.                                         *)
(* In the ssmPlanPB ssm, the last state is COMPLETE.  This is reached when the*)
(* 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 acl_infRules aclDrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfRules uavUtilities
open OMNITypeTheory PBTypeIntegratedTheory PBIntegratedDefTheory


val _ = new_theory "ssmPBIntegrated";


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

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

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


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

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



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

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


val _ = export_theory();
```

120

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.

[2] United States Army Ranger School, ATTN: ATSH-RB, 10850 Schneider Rd, Bldg 5024, Ft Benning, GA 31905. *Ranger handbook*, April 2017.

[3] The Rand Corporation. Security controls for computer systems: Report of defence science board task force on computer security. Technical report, Office of the Director of Defence Research And Engineering, Washington D.C. 20301, February 1970.