



Figure 1: This is a caption for this figure.

Abstract

Copyright

Disclaimer

The views discussed in this master thesis are that of the author's. They do not in anyway represent the views of the United States Air Force Research Laboratory (AFRL) in Rome, NY or Professor Shiu-Kai Chin from the College of Engineering and Computer Science at Syracuse University. They are also the sole written work of the author and thus do not represent the views of the other participants in this research.

Acknowledgements

This research began in the summer of 2017 as part of the Assured by Design (ABD) program funded by the United States Air Force Research Laboratory (AFRL) in Rome, NY and managed by the principal investigator Professor Shiu-Kai Chin from the College of Engineering and Computer Science at Syracuse University. This project was envisioned by Professor Shiu-Kai Chin to satisfy the needs of the ABD program. This master thesis evolved directly from this work.

Thanks and recognition go to the following people for their contribution to this project. Professor Shiu-kai Chin for providing me with the opportunity and for his faith in me and my capabilities on this project. Erich Devendorf at AFRL for making the ABD program happen. Mizra Tihic for making this happen, especially with respect to funding.

To properly acknowledge the contribution of others requires some description of the workflow. The actual work began as a collaboration between the subject matter expert from the United States Army and me, the author of this master thesis. The subject matter expert was Jesse Nathaniel Hall, a Captain [rank?] in the United States Army and also a graduate student in the iSchool (School of Information Science) at Syracuse University. Given the objective of demonstrating CSBD on the patrol base operation (or demonstrating its failure), we collaborated on the Systems Security Engineering (SSE) goals of the project. This work comprised a significant part of this research and is described in the chapter on Systems Security Engineering. From thereon, the work was divided among the two of us with weekly updates and collaboration to resolve any potential conflicts. Jesse modeled the patrol base operations in Visio based on his interpretations of the patrol base operations in the Ranger Handbook [1]. A diagram of his work was included as a Visio file with this project. In addition, a squished version of this diagram was included in the chapter on the Patrol Base Operations. The result of this work was discussed in this context. On the other hand, I focused on the actual application of CSBD to the model as it was being developed. I continued to work on this aspect of the project after collaboration ceased.

In addition to the work done by Jesse and myself, another student worked with us on the project. This was YiHong Guo, an undergraduate student in the College of Engineering and Computer Science at Syracuse University. He helped us organize the original documentation of this work in LaTeX, a rather large project. (That documentation is separate from this master thesis.)

Table Of Contents

Abstract	i
List of Figures	x
List of Tables	xii
List of Acronyms	xiii
1 Introduction	1
1.1 In Context of Systems Security Engineering	2
1.2 Certified Security by Design (CSBD)	3
1.3 Extending The Range of Applicability	4
1.4 This Master Thesis	6
2 Background	2
2.1 Complete Mediation	2
2.2 Confidentiality, Integrity, and Availability	3
2.3 Formal Methods	4
2.4 Functional Programming	6
2.5 Algebraic Data Types in ML	7
2.6 Higher Order Logic (HOL) Interactive Theorem Prover	10
2.7 Other Interactive Theorem Provers	11
2.8 How to Compile The Included Files	11
3 Systems Security Engineering & Patrol Base Operations	12
3.1 The Systems Perspective	12
3.2 Systems Engineering	13
3.3 NIST Special Publication 800-160	16
3.4 Systems Security Engineering	17
3.5 Systems Security Engineering Framework	19
3.5.1 Problem	20
3.5.2 Solution	21
3.5.3 Trustworthiness	24
3.6 Verification & Documentation	25
3.6.1 Verification	25
3.6.2 Documentation	25
3.6.2.1 Accountability	26
3.6.2.2 Reproducibility	26

4 Certified Security by Design (CSBD) & Access-Control Logic (ACL)	27
4.1 Certified Security by Design (CSBD)	27
4.2 Access-Control Logic (ACL)	28
4.2.1 ACL: A Command and Control (C2) Calculus	28
4.2.2 Principals	29
4.2.3 Propositional Variables, Requests, Authority, and Jurisdiction	29
4.2.4 Well-formed Formulas	30
4.2.5 Kripke Structures & Semantics	31
4.2.5.1 Kripke Structures	31
4.2.5.2 Kripke Semantics	33
4.2.5.3 Satisfies	33
4.2.5.4 Soundness	34
4.2.6 Inference Rules	34
4.2.7 Complete mediation	35
4.3 ACL in HOL	37
4.3.1 Principals	38
4.3.2 Well-Formed Formulas	39
4.3.3 Kripke structures	40
4.3.4 ACL Formulas	41
4.3.5 Kripke Semantics: The Evaluation Function	43
4.3.6 Satisfies	43
4.3.7 Soundness	44
5 Patrol Base Operations	45
5.1 Motivation	45
5.2 Patrol Base Operations	45
5.3 Modeling the Patrol Base Operations from the Ranger Handbook	46
5.4 Overview of The Hierarchy of Secure State Machines	47
5.5 Hierarchy of Secure State Machines	50
5.5.1 Diagrammatic Description in Visio	50
5.5.2 Descriptions of Individual Modules	52
5.5.3 OMNI-Level	54
5.5.4 Escape	55
5.5.5 Top Level	56
5.5.6 Horizontal Slice	57
5.5.6.1 ssmPlanPB	57
5.5.6.2 ssmMoveToORP	59
5.5.6.3 ssmConductORP	60
5.5.6.4 ssmMoveToPB	61
5.5.6.5 ssmConductPB	62
5.5.7 Vertical Slice	63
5.5.7.1 ssmSecureHalt	64
5.5.7.2 ssmORPRecon	65
5.5.7.3 ssmMoveToORP4L	66
5.5.7.4 ssmFormRT	67

6 Secure State Machine Model	68
6.1 State Machines	68
6.1.1 States	68
6.1.2 Transition Commands	69
6.1.3 Next-state Function	70
6.1.4 Next-output Function	70
6.1.5 Configuration	71
6.1.6 TR Relations	71
6.2 Secure State Machines	73
6.2.1 State Machine Versus Secure State Machine	73
6.2.2 Monitors	74
6.2.3 Transition Types	74
6.2.3.1 <i>exec</i>	74
6.2.3.2 <i>trap</i>	75
6.2.3.3 <i>discard</i>	75
6.2.4 Commands	76
6.2.5 Authentication	77
6.2.6 Authorization	78
6.2.7 Next-state And Next-output Functions	79
6.2.8 Configurations	79
6.2.9 TR Relations	79
6.2.10 Configuration Interpretation	80
6.3 Secure State Machines in HOL	82
6.3.1 Parameterizable Secure State Machine	82
6.3.2 Input Stream	83
6.3.3 Commands	84
6.3.3.1 Transition Types	87
6.3.4 Authentication	88
6.3.5 Authorization	89
6.3.6 Next-state And Next-output Functions	89
6.3.7 Configurations	90
6.3.8 Configuration Interpretation	91
6.3.9 TR Rules	92
7 Patrol Base Operations as Secure State Machines	98
7.1 ssmPB: A Typical Example from the Hierarchy	98
7.1.1 Principals	100
7.1.2 States	100
7.1.3 Outputs	101
7.1.4 Commands	102
7.1.5 Next-State Function	103
7.1.6 Next-Output Function	104
7.1.7 Authentication	104
7.1.8 Authorization	107
7.1.9 Proved Theorems	109
7.2 ssmConductORP: Multiple Principals	121
7.2.1 Principals	122
7.2.2 States	122

7.2.3	Outputs	122
7.2.4	Commands	123
7.2.5	Next-State Function	124
7.2.6	Next-Output Function	125
7.2.7	Authentication	125
7.2.8	Authorization	126
7.2.9	Proved Theorems	128
7.3	ssmPlanPB: Non-sequential Transitions	139
7.3.1	Principals	141
7.3.2	States	141
7.3.3	Commands	142
7.3.4	Output	143
7.3.5	Next-State Function	143
7.3.6	Next-Output Function	150
7.3.7	Authentication	151
7.3.8	Authorization	154
7.3.9	Proved Theorems	156
8	Non-implemented modules	169
8.1	Authentication & Roles	169
8.2	Soldier, Squad, and Platoon Theories	170
9	Discussion	172
9.1	Summary	172
9.2	Challenges And Limitations of The Approach	174
10	Future Work	175
10.1	Applicability	175
10.1.1	Accountability Systems	176
Appendices		178
A	Access Control Logic Theories: Pretty-Printed Theories	179
B	Secure State Machine & Patrol Base Operations: Pretty-Printed Theories	202
C	Secure State Machine Theories: HOL Script Files	317
C.1	ssm	317
C.2	satList	322
D	Secure State Machine Theories Applied to Patrol Base Operations: HOL Script Files	324
D.1	OMNILevel	324
D.2	escapeLevel	325
D.3	TopLevel	325
D.3.1	PBTypeIntegrated Theory: Type Definitions	325
D.3.2	PBIntegratedDef Theory: Authentication & Authorization Definitions	327
D.3.3	ssmPlanPBIntegrated Theory: Theorems	329

D.4	Horizontal Slice	334
D.4.1	ssmPlanPB	334
D.4.1.1	PlanPBType Theory: Type Definitions	334
D.4.1.2	PlanPBDef Theory: Authentication & Authorization Definitions	334
D.4.1.3	ssmPlanPB Theory: Theorems	334
D.4.2	ssmMoveToORP	334
D.4.2.1	MoveToORPType Theory: Type Definitions	334
D.4.2.2	MoveToORPDef Theory: Authentication & Authorization Definitions	334
D.4.2.3	ssmMoveToORP Theory: Theorems	334
D.4.3	ssmConductORP	334
D.4.3.1	ConductORPType Theory: Type Definitions	334
D.4.3.2	ConductORPDef Theory: Authentication & Authorization Definitions	334
D.4.3.3	ssmConductORP Theory: Theorems	334
D.4.4	ssmMoveToPB	334
D.4.4.1	MoveToPBType Theory: Type Definitions	334
D.4.4.2	MoveToPBDef Theory: Authentication & Authorization Definitions	334
D.4.4.3	ssmMoveToPB Theory: Theorems	334
D.4.5	ssmConductPB	334
D.4.5.1	ConductPBType Theory: Type Definitions	334
D.4.5.2	ConductPBDef Theory: Authentication & Authorization Definitions	334
D.4.5.3	ssmConductPB Theory: Theorems	334
D.5	Vertical Slice	334
D.5.1	ssmSecureHalt	334
D.5.1.1	SecureHaltType Theory: Type Definitions	334
D.5.1.2	SecureHaltDef Theory: Authentication & Authorization Definitions	334
D.5.1.3	ssmSecureHalt Theory: Theorems	334
D.5.2	ssmORPRecon	334
D.5.2.1	ORPReconType Theory: Type Definitions	334
D.5.2.2	ORPReconDef Theory: Authentication & Authorization Definitions	334
D.5.2.3	ssmORPRecon Theory: Theorems	334
D.5.3	ssmMoveToORP4L	334
D.5.3.1	MoveToORP4LType Theory: Type Definitions	334
D.5.3.2	MoveToORP4LDef Theory: Authentication & Authorization Definitions	334
D.5.3.3	ssmMoveToORP4L Theory: Theorems	334
D.5.4	ssmFormRT	334
D.5.4.1	FormRTType Theory: Type Definitions	334
D.5.4.2	FormRTDef Theory: Authentication & Authorization Definitions	334
D.5.4.3	ssmFormRT Theory: Theorems	334

E Map of The File Folder Structure	335
References	336

List of Figures

1	This is a caption for this figure.	i
2.1	An implementation of the boolean datatype in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [2]	7
2.2	An implementation of a tree datatype in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [2]	8
2.3	An implementation of the weekday datatype in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [2]	8
2.4	An implementation of the function is_weekend in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [2]	8
2.5	A parameterized implementation of a tree datatype in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [2]	9
3.1	Systems security engineering in relation to systems engineering. (Image from NIST Special Publication 800-160: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.)	17
3.2	Systems security engineering Framework. (Image from NIST Special Publication 800-160: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.)	20
4.1	Kripke semantics. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [3]	33
4.2	The access-control logic (ACL) inference rules. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [3]	35
4.3	The <i>Controls</i> inference rule. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [3]	35
4.4	The <i>Reps</i> inference rule. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [3]	36
4.5	The <i>Derived Speaks For</i> inference rule. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [3]	36
4.6	The HOL implementation of principle (<i>Princ</i>). Image from <i>Certified Security by Design Using Higher Order Logic</i> [4]	38
4.7	The definition for Form in HOL. <i>Certified Security by Design Using Higher Order Logic</i> [4]	39
4.8	The HOL implementation of a Kripke structure (<i>Princ</i>). Image from <i>Certified Security by Design Using Higher Order Logic</i> [4]	41
4.9	The ACL formulas in HOL. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [3]	42

4.10	The HOL implementation of the Kripke semantics (evaluation function). Image from <i>Certified Security by Design Using Higher Order Logic</i> [4]	43
4.11	The HOL implementation of the "satisfies" property. Image from <i>Certified Security by Design Using Higher Order Logic</i> [4]	44
4.12	The HOL representation of the "soundness" property. Image from <i>Certified Security by Design Using Higher Order Logic</i> [4]	44
5.1	A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel as part of the research involved in this master thesis. Shall I get his permission?	47
5.2	Diagrammatic description of patrol base operations as a hierarchy of secure state machines. (Generated by Jesse Nathaniel Hall.)	50
5.3	Escape level diagram.	55
5.4	Top level diagram.	56
5.5	Horizontal slice: PlanPB diagram.	57
5.6	Horizontal slice: MoveToORP diagram.	59
5.7	Horizontal slice: ConductORP diagram.	60
5.8	Horizontal slice: MoveToPB diagram.	61
5.9	Horizontal slice: ConductPB diagram.	62
5.10	Vertical slice: SecureHalt diagram.	64
5.11	Vertical slice: ORPRecon diagram.	65
5.12	Vertical slice: MoveToORP4L diagram.	66
5.13	Vertical slice: FormRT diagram.	67
6.1	A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel as part of the research involved in this master thesis.	69
6.2	State machine versus secure state machine with a monitor. Image taken from <i>Certified Security by Design Using Higher Order Logic</i> [4].	73
6.3	Top level diagram.	76
6.4	Transition rules in HOL. Image taken from <i>Certified Security by Design Using Higher Order Logic</i> [4]	93
7.1	Top level diagram.	99
7.2	Horizontal slice: ConductORP diagram.	121
7.3	Horizontal slice: PlanPB diagram.	139

List of Tables

5.1 Mission Activity Specification for Patrol Base Operations. Adapted from the U.S. Army Ranger Handbook 2017 [1].	46
--	----

List of Acronyms

ACL access-control logic.

WFF well-formed formula.

Chapter 1

Introduction

Imagine this scene from the movie titled "The Kingdom" [5]. It is a bright, sunny day in Saudi Arabia. The compound is isolated from the rest of the Kingdom and heavily guarded by Saudi police. The American civilian contractors are free to enjoy the American culture they are accustomed to without offending the locals. A large group of contractors and their families are enjoying their day off watching a game of little league on the compound.

Unexpectedly, two men dressed in Saudi police uniforms begin showering the crowd with machine gun fire. Civilian men, women, and children run screaming for cover. The two gun men are eventually neutralized by Saudi police officers. Saudi police yell to the survivors "come this way" to lure them away from the carnage. One of these men dressed in a police uniform shouts "Allah Akbar" and detonates, killing himself and several others near him.

What went wrong? How did these terrorists gain access to a secured compound?

Movies like this and the news are bringing the reality of security to the forefront of everyone's mind. Everyone is concerned with controlling access to their information,

their property, and more importantly their lives.

To drive it home, consider the importance of access control in computer driven cars, bank accounts, medical records, pace makers, etc. Access control can be as simple as providing your daughter with a secret password for who can pick her up from school or as complicated as managing access to national security secrets. In any and every system that requires some form of security, access-control is one of the foremost considerations. The work discussed in this master thesis focuses on the critical property of access control in systems security engineering.

This chapter provides an introduction to this master thesis. It briefly touches upon the field of systems security engineering and the subspecialty of access-control by way of complete mediation. It then introduces an approach to verifying and documenting this property using the Certified Security by Design (CSBD) approach. After introducing the CSBD, this chapter presents an application of CSBD that aims to close the gap in its field of applicability. It discusses how CSBD is applied to a non-automated, human-centered military system (patrol base operations). Finally, this chapter offers the reader a glimpse at what to expect in the subsequent chapters.

1.1 In Context of Systems Security Engineering

The authority on standards in systems engineering is the International Organization for Standardization (ISO). In one of its published standards (ISO/IEC/IEEE 15288 [6]) it describes systems engineering guidelines for managing man-made systems. These man-made systems span the range from human-centered systems to fully automated systems. An example of a human-centered system is a partially- or non-automated system for controlling access to a base or compound. An example of a fully automated system is the computer that controls a driverless car. In all these systems, security is

often a critical component of the systems engineering process.

The recognized authority in systems security engineering, a subspecialty of systems engineering, is the National Institute for Standards and Technology (NIST) Special Publication 800-160 (vol 1 and vol 2) [7]. It describes a framework for designing trustworthy systems by engineering security into the system from the start.

A subspecialty of systems security engineering manages access to security-sensitive objects using the principle of complete mediation. Complete mediation broadly refers to verifying and documenting authentication and authorization of access to security sensitive objects.

1.2 Certified Security by Design (CSBD)

CSBD is an approach to verifying and documenting that a system's design satisfies the property of complete mediation. It uses formal methods by way of an access-control logic (ACL) that is implemented in an interactive theorem prover. Formal methods add a mathematical rigor to the verification process. The ACL is a formal propositional logic. The interactive theorem prover takes the rigor a step further by using computer-aided reasoning in a trusted implementation. The result is a set of formal proofs that both verify and document that the system's design is trustworthy with respect to complete mediation. This set of formal proofs is verifiable and reproducible by independent third parties.

It is a straight forward process to apply CSBD. First, security-sensitive objects are identified. For example, a secured compound or bank account information are security-sensitive objects. Then properties of access control are described in terms of authentication and authorization. For example, authentication may rely on ID badges or atm cards. Authorization may be described in a policy that determines who is

authorized to access the compound or who is authorized to access bank account funds. Additionally, modification and access rights may also be described in terms of ordered security and integrity labels. For example, security labels for a bank account may include r (read only) $\leq d$ (deposit) $\leq w$ (withdraw). Account owners may be granted the highest integrity level of withdraw, whereas bank managers may only be granted read access. The owner can read, deposit, and withdraw funds from the account. The bank manager can only read funds from the account. Integrity labels may include st (standard service) $\leq gl$ (gold service) $\leq pr$ (premium service). If standard services include checking account fees and gold services do not and the account owner has a gold services account, then the owner should not be subject to the standard services checking account fees. Furthermore, standard services account holders should not be granted gold service privileges because it would lessen the value of those service.

Next, the system is described in HOL wherein properties of complete mediation are verified. This verification is a set of formal proofs that are included with the system's documentation. Full disclosure of methods are included so that the proofs can be verified and reproduced by independent third parties.

1.3 Extending The Range of Applicability

CSBD is a core requirement for the Master of Science degree in Cybersecurity at Syracuse University. It is also part of the Air Force Research Laboratory's STPA-Sec doctrine. But, up until now, CSBD has been applied to automated systems. For example, CSBD has been used with JPMC's SWIFT protocols [8] and the F-18 Viper UAV payload controller and secure memory loader verifier (SMLV). But, systems engineering spans the range of fully automated to non-automated, human-centered systems. This master thesis aims to close the gap of CSBD applicability by applying it to a non-automated, human-centered system. The aim is to determine if it can be done

and if so to demonstrate it.

To accomplish this goal, patrol base operations are chosen as an example of a non-automated, human centered system. Patrol base operations are described in the U.S. Army Ranger Handbook [1]. A security matter expert from the U.S. Army is employed to interpret the patrol base operations.

The patrol base operations are first discussed with regards to security, assets, and unacceptable asset losses. From there, the patrol base operations are described as a hierarchy of secure state machines. The top level represents the most abstract description of the patrol base operations. The descriptions become less abstract as the hierarchy descends. Once the hierarchy is designed, it is implemented in HOL. Properties of complete mediation are proved and documented.

Application of CSBD to these patrol base operations is a success. The hierarchical approach is an easy way to describe large systems in a manageable way. Secure state machines ease the process of proving complete mediation because it is built into the structure of the secure state machines. In addition, a variety of insights are obtained including multiple representations of the operations, possible application to partially automated accountability systems, and a greater understanding of the operations themselves.

Most of the work comes in the creativity of designing the system in a manner that affords itself to verification of complete mediation. Thus, there were no limitations found in this work as to the applicability of the CSBD.

In summary, this work finds that CSBD is effective on non-automated, human-centered systems by way of example on patrol base operations. This closes the gap in CSBD applicability by including these types of systems at one end of the range from fully automated to fully non-automated systems. Its importance is derived from the

importance of access-control in systems security engineering. It is simply a method that works and does so effectively.

1.4 This Master Thesis

Subsequent chapters in this master thesis describe relevant background and details of CSBD applied to patrol base operations. The chapters begin with abstract descriptions of the patrol base operations in the context of the specific chapter. Each chapter adds sufficient detail that subsequent chapters should be more easily understood if read in order. Nonetheless, there are sufficient references to previous chapters if the reader wishes to skip around.

Chapter 2 discusses background material that may be helpful to the reader in understanding the following chapters. Chapter 3 discusses the hierarchical model of the patrol base operations in the context of systems engineering and systems security engineering. Chapter 4 describes Certified Security by Design and the access-control logic in detail. It also discusses the HOL implementation of the ACL. Chapter 5 describes the hierarchical model of the patrol base operations. Chapter 6 describes secure state machines and their HOL implementation. Chapter 7 describes the implementation of several patrol base operations secure state machines in HOL. Chapter 8 describes additional models that are discussed throughout the project, but not implemented in HOL. Chapter 9 provides some discussion of the findings that are not covered in previous chapters. Finally, Chapter 10 discusses future work and implications.

The appendices include pretty-printed HOL-generated output of all the ACL theorems. They also contain the code for all the HOL theorems both as pretty-printed HOL-generated output and the actual code. There is also an appendix describing the

folder structure of the files included with this thesis.

As the example at the start of this chapter makes clear, access control is a serious matter. Systems should be designed with security from the start. Certified Security by Design (CSBD) is an effective approach to designing trustworthy systems with respect to access control. It does this using formal methods and computer-aided reasoning to verify and document the property of complete mediation.

The next chapter provides some background material that is relevant to subsequent chapters.

Chapter 2

Background

This section aims to provide some background on subjects discussed in this master thesis. These subjects are not directly addressed in other areas of this master thesis. Nevertheless, knowledge of them is either necessary or useful to understanding what follows.

2.1 Complete Mediation

The seminal paper quoted on the principle of complete mediation is "The Protection of Information in Computer Systems" by Saltzer and Schroeder.

Complete mediation: Every access to every object must be checked for authority. This principle, when systematically applied, is the primary underpinning of the protection system. It forces a system-wide view of access control, which in addition to normal operation includes initialization, recovery, shutdown, and maintenance. It implies that a foolproof method of identifying the source of every request must be devised. It also requires that

proposals to gain performance by remembering the result of an authority check be examined skeptically. If a change in authority occurs, such remembered results must be systematically updated. [9]

In summary, what this means is that any accessor attempting to access or modify a protected object must satisfy two conditions: (1) the accessor must be authenticated, and (2) the accessor must be authorized to access or modify that object. This involves a three-step process: (1) defining the protected objects, (2) declaring who has what rights on these objects, and (3) defining how to verify the identity of the individual(s) accessing or modifying the objects.

Complete mediation is the underpinning of systems requiring access control. From access to bank accounts to whose text messages get blocked, complete mediation is a critical component of security. Complete mediation is also the subject of the access-control logic discussed in this master thesis.

2.2 Confidentiality, Integrity, and Availability

At the core of security are the concepts of confidentiality, integrity, and availability. Confidentiality refers to limiting access to objects (including information) by ensuring that only the right people (etc.¹) have access to these things. Confidentiality is the realm of things such as authentication and authorization. Authentication means verifying a person's identity. Authorization means describing a person's access rights.

Integrity, on the other hand, refers to the whole of the object (or information, etc.). It means controlling who or what can modify the object. Integrity is the realm of things such as authorization. In this case, authorization means describing a person's right to modify an object.

¹or any other "entity" that can request access to an object.

Availability refers to accessibility. Wikipedia describes availability as a measure of operability or degree to which the system is mission capable [10].

Both confidentiality and integrity are guarded by the principle of complete mediation. This master thesis focuses on this, the realm of confidentiality and integrity.

2.3 Formal Methods

(Primary source for this section is [11])

Formal methods are aimed at improving the reliability and correctness of systems[12]. They are particularly useful in the design phase of systems engineering. But, they are also employed to varying degrees throughout all aspects of the systems engineering process².

Formal methods analysis is a three phase process: (1) formal specification of the system using a modeling language, (2) verification of the specification, and (3) implementation [11]. Specification consists of describing the system in a mathematical-based modeling language [11]. Verification entails proving properties of the system, typically using either model checking or theorem proving techniques. Implementation depends on the type of system (i.e., software, hardware, human-centered system, etc.).

The two most-noted formal methods are model checking and theorem proving. Model checking involves testing possible states of a system for correctness. Model checking is touted more as an error checking tool. It exhausts all possible states (or test states) in search of failures. But, the absence of failure is not proof of correctness. Furthermore, for large systems, model checking is resource intensive. It is subject to the state explosion problem, wherein the number of states of the system grows exponentially with

²For example, there is notable progress being made in formally verifying c code. See for example [13] and [14]

the complexity of the system [15].

Theorem proving, on the other hand, employs a formal logic to prove properties of the system. Formal proofs remain true for any test case [11]. This means that these are proofs of correctness and not just proofs of the absence of failure. Theorem proving is usually partially or fully automated [16]. It often requires specialized knowledge and a sufficient degree of competency in mathematics³. However, formal theorem proving techniques have been successfully taught to undergraduates⁴.

Model checking and theorem proving often work synergistically. Both methods offer benefits that the other does not. As a whole, they improve the reliability of the system's design. As an example, ElasticSearch successfully applies both methods to its search methodology on distributed systems [17].

Formal methods are used in some areas of systems engineering more than others. Some engineers are reluctant to use them because of the additional work and level of expertise required. However, when correctness is non-negotiable, such as safety and security, formal methods become essential. Formal methods are predicted to increase in usage as tools become easier to use and engineering curricula increasingly offer formal methods as part of their core[11].

Specification has its benefits, not only as a precursor to verification, but also as a tool for understanding. In cases wherein properties of the system are not proved, the act of formally specifying the system adds a degree of rigor to the process. This rigor often brings new insights about the system because it causes the engineer to think more systematically about the system's design. In addition, the conversion of a field's jargon into a precise specification language also aids in reproducibility and communicability⁵

³and a good amount of patience, in the author's opinion.

⁴The PI Professor Shiu-Kai Chin teaches CSBD, which includes theorem proving, to both undergraduates and graduates at Syracuse University.

⁵The later ideas are not specifically stated in the main cite, but follow logically from the author's perspective.

[11].

This master thesis applies formal verification methods, specifically theorem proving, to prove the security properties of a system. Theorem proving is partially automated using the Higher Order Logic (HOL) Interactive Theorem Prover.

2.4 Functional Programming

(Primary source for this section is [18])

Functional programming is a style of programming that uses functions to define program behavior. Functional programming is inherently different than procedural or object-oriented programming. These styles of program use procedures or objects and classes to define program behavior. C and Pascal are examples of procedural programming languages. C++ and Java are examples of object-oriented programming languages. Haskell and ML⁶ (meta language) are examples of functional programming languages.

Functional programming languages are thought to be more pure. They have fewer side effects than procedural or object-oriented programming. They produce fewer bugs. Functional programming languages are thus considered more reliable. This is why theorem provers are typically implemented in functional programming languages.

This master thesis relies on the Higher Order Logic (HOL) Interactive theorem prover. HOL is implemented in a functional programming language. HOL is thought to be very reliable and trustworthy. Part of that trustworthiness is a function of the meta language in which it is described, namely polyML.

⁶ML is not considered a purely functional.

2.5 Algebraic Data Types in ML

The primary source for this section is *Introduction to Programming Languages/Algebraic Data Types* [2]. This section is helpful in understanding section 4.3. The reader may skip this section and return to it before reading that section.

The online book *Introduction to Programming Languages/Algebraic Data Types* describes types as sets. For example, the boolean type is a set composed of two values "True" and "False".

An algebraic data type (ADT) is a composite data type. The boolean type is also an ADT. In ML, ADTs are preceded by the `datatype` keyword. The boolean data type would be defined as:

A note on type constructors: in ML⁷ a type constructors should be thought of as a function that takes a type as its parameter and returns a type. In the declaration `Node of 'a tree * 'a * 'a tree`. `Node` behaves like a function which returns something of type `'a tree` [19].

```
datatype Boolean = True  
                  | False
```

Figure 2.1: An implementation of the boolean datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [2]

The boolean datatype can be thought of as the union of the two singleton sets "True" and "False". The `|` symbol denotes union in this case.

An example of a more complicated datatype is the tree datatype defined below.

The first element of the tree datatype is "Leaf". The second element is a "Node". The

⁷Standard ML according to the author [19].

```

datatype tree = Leaf
              | Node of tree * int * tree;

```

Figure 2.2: An implementation of a tree datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [2]

node has three components: tree, int, and another tree. The "of" keyword in ML indicates that what follows is a type. Thus, this is a "Node" **of** type tree * int * tree. The * symbol represents the cartesian product. The easiest way to think of this particular example in ML is that the "Node" type is a three-tuple consisting of (tree, int, tree).

ADTs exhibit the property of pattern matching. This is typical of functional programming languages. For example, consider the datatype weekday shown below.

```

datatype weekday = Monday
                  | Tuesday
                  | Wednesday
                  | Thursday
                  | Friday
                  | Saturday
                  | Sunday

```

Figure 2.3: An implementation of the weekday datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [2]

With pattern matching, it is possible to define a function `is_weekend` which returns "True" if it is a weekend and "False" otherwise. Consider the following definition for a function that takes one argument (or parameter) of type "weekday."

```

fun is_weekend Sunday    = True
  | is_weekend Saturday = True
  | is_weekend _        = False

```

Figure 2.4: An implementation of the function `is_weekend` in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [2]

The keyword "fun" must precede any function definition in ML. Next, is the name of the function, in this case "is_weekend". The next word "Sunday" is an element of the datatype "weekday." When ML evaluates the "is_weekend" function, it will check the argument. If the argument is "Sunday" then ML will return "True", otherwise it will check the next line. If the argument equals "Saturday" ML will return "True", otherwise it will check the next line. The next line contains the underscore character. In ML, this is similar to the wildcard * character in Unix. In this case, the underscore character tells ML to return "False" for any argument. The novelty of pattern matching is that function evaluation proceeds in order, allowing for an easy and clean way to define an if-then-else evaluation.

A second property of ADTs is parameterization. Consider a more general definition of the tree datatype shown below.

```
datatype 'a tree = Leaf
                  | Node of 'a tree * 'a * 'a tree;
```

Figure 2.5: A parameterized implementation of a tree datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [2]

In this definition, the datatype definition is preceded by the type variable '`'a`'. To declare something of type '`'a Tree`', the '`'a`' is instantiated. For example, `int tree`, produces the same tree is the original definition.

`'a` is called a polymorphic type or type variable. A type variable is just a place holder for some other type or type variable. Type variables are always preceded with a forward tick mark (apostrophe) in ML.

2.6 Higher Order Logic (HOL) Interactive Theorem Prover

The Higher Order Logic (HOL) Interactive theorem prover is a proof assistant. HOL has proved to be a very reliable theorem proving system. It is widely trusted in the interactive theorem proving community.

At its core, HOL implements a small set of axioms and a formal logic. All inferences and theorems must be derived from this small set of axioms using the formal logic. Reasoning logically with a small set of axioms contributes to the trustworthiness of the system. The user only has to trust the small set of axioms and the logic (in addition to HOL's implementation). Beyond the competence of the programmer, it is said that if it can't be proved in HOL then it can be proved.

HOL is a strongly-typed system. This means that data has a predefined type. As in all functional programming languages, the type of these data can not change. This adds to the reliability of HOL by preventing side-effects. HOL has several built-in data types. But, the user can also define her own data type. In addition to datatypes, the user can define her own set of axioms and definitions.

With user-defined types, axioms, and definitions, the user can describe a system in HOL and then use HOL's formal logic to prove properties of this system. This is the basis for theorem proving in formal methods.

The version of HOL used for this master thesis is HOL4. HOL4 is free software that is BSD licensed [20]. Download instructions can be found at the HOL website hol-theorem-prover.org [21]. According to Wikipedia, HOL4 is descended from HOL88. The HOL88 project is Mike Gordon's effort.. HOL4 is implemented in polyML which is an implementation of Standard ML [22].

2.7 Other Interactive Theorem Provers

It should be noted that HOL is not unique. There are other interactive theorem provers on the market. Each has its own niche and dedicated user base. Choice of theorem prover is typically guided by personal preference and familiarity. The later follows from the somewhat steep learning curve for theorem provers.

For example, Isabelle/HOL⁸ is another popular interactive theorem prover. The access-control logic (ACL) has been partially implemented in Isabelle/HOL by Scott Constable, a PhD student in the College of Engineering and Computer Science and Syracuse University.

2.8 How to Compile The Included Files

All the files necessary to compile the theories discussed in this master thesis are included. A diagram of the folder structure is included in the appendix E. To compile the theories and the LaTeX files go to the folder titled MasterThesis. Open up a terminal. Type *make* and then hit enter. Note that to compile the theories HOL and LaTeX must be first be installed.

⁸The author is interested in implementing more of the ACL and secure state machines in Isabelle/HOL.

Chapter 3

Systems Security Engineering & Patrol Base Operations

3.1 The Systems Perspective

A system is a set of interacting and interdependent components that act as a whole to perform some behavior or function. Examples of systems include the human body, socio-political systems, and computer systems.

The patrol base operations satisfy this definition of a system. As a whole, the patrol base operations perform some function(s). This function is described in the Ranger Handbook [1] and discussed in chapter 5. The patrol base operations are comprised of interdependent and interacting components. In general these components are the individual soldiers. But, the way this master thesis defines the patrol base operations, the definition of a component varies.

This master thesis defines the patrol base operations as a system of systems. More specifically, this thesis models the patrol base operations as a hierarchy of secure state

machines. Chapter 6 describes SSMs in general. Section 5.3 describes this model of the patrol base operations.

This model presents the patrol base operations as a hierarchy wherein each level of the hierarchy represents a decreasing level of abstraction. At the top and most abstract level, the components are phases of the patrol base operations. These phases commence in a sequential order to achieve the goal of the patrol base operations. Each lower level of the hierarchy is composed of less abstract phases. At each level, the components function sequentially (typically) to achieve the ultimate goal.

This system of systems also contains non-hierarchically defined components. For example, an escape-level component models situations wherein the patrol base operations are aborted. The escape level component is reachable from any component at any level of the hierarchy. Soldiers (as in a soldier model) also function within this system of systems in a non-hierarchical manner. However, the soldier module is not verified using the ACL because of time constraints. Nevertheless, a soldier module is discussed in the Discussions chapter 9 of this thesis.

In this way, the patrol base operations represent a system and are amiable to the systems engineering perspective.

3.2 Systems Engineering

The recognized authoritative standard on systems engineering is ISO/IEC/IEEE 15288 [6]. This document is titled "Systems And Software Engineering–System Life Cycle Processing." A precursor to this standard is ISO/IEC TR 24748 1 [23] titled "Systems And Software Engineering–Life Cycle Management–Part 1: Guide for Life Cycle Management." These are the primary sources for this section.

Systems engineering is an interdisciplinary approach aimed at solving problems involved in the various phases of the life cycle of a system. ISO/IEC/IEEE 15288 and ISO/IEC TR 24748 1 define five major phases of the this life cycle: concept, development, production, utilization, support, and retirement.

This master thesis focuses on the Certified Security by Design (CSBD). The key premise of CSBD is that security should be built into the systems from the start, i.e., the design phase. This is the leading notion in systems engineering and its sub-discipline Systems Security Engineering. Nevertheless, it is instructive to see how CSBD, this thesis, and the patrol base operations fit into the systems engineering framework described by the ISO standards.

Concept The concept phase describes stakeholders and the Concept of Operations (ConOps). Stakeholders are those who have a stake in the system. ConOps are defined in a variety of ways. Wikipedia defines ConOps as "...a document describing the characteristics of a proposed system from the viewpoint of an individual who will use that system [24]. U.S. Air Force Policy Directive 10-28 defines ConOps as "a high level concept whose purpose is to describe operational mission areas, enablers and effects necessary to achieve desired results/outcomes" [25]. ConOps are context specific.

The patrol base operations are already modeled and implemented. The mission areas, enablers, and effects to achieve the desired goal are already in place. The aim for this thesis is to maintain the structure of the patrol base operations and model it in a way that is amiable to verification and documentation of complete mediation.

It is possible to look at the work in this thesis as the modification of an already in-use system. For example, an accountability system involving the patrol base operations may require a remodeling of the operations with few, if any, changes in the actual operations themselves. This is discussed in more detail in the Future Works & Implications Chapter in section 10.1.1.

Certified Security by Design (CSBD) applies to these two areas: the concept phase and the re-conceptualization phase.

Development The development stage involves refinement of the ConOps and production of products. The development of the patrol base operations took place a long time ago. Regardless, the products for the patrol base operations are the description of the operations themselves. The product with regards to this master thesis is demonstrated satisfaction of the principle of complete mediation.

CSBD does not add anything new to the development of the system. However, it adds constraints to the concept phase which must be adhered to throughout the development phase.

Production Production is self-explanatory, however not so obvious with a system such as the patrol base operations. There are no products save for the trained soldier. In this aspect, the production phase would entail educating the soldiers. Neither this master thesis nor CSBD cover¹ education of soldiers. (Of course, one could imagine a lecture on the importance of authentication and authorization as a critical component of any soldier's primary training.)

Utilization Utilization is also self-explanatory. The patrol base operations are utilized in the field as soldiers are deployed and assigned missions that involve patrol base operations. Again, neither this master thesis nor CSBD focus on utilization.

Support Support provides for maintenance of the system. In the case of the patrol base operations, support would entail documenting feedback from soldiers and assessing the efficacy of the operations. This could also entail updating the operations based on

¹Although, CSBD is a required course at Syracuse University in the Cyber Security program.

this feedback and based on advances in technology. Neither this master thesis nor CSBD directly address support for the system. However, any updates to the system would likely entail consideration of authentication and authorization. Herein, this master thesis and CSBD are relevant and useful.

Retirement Retirement refers to the end stages of the operations. For the patrol base operations and similar systems, retirement would most likely entail replacement with other operations. Again, neither this master thesis nor CSBD focus on retirement. However, both are relevant to the conceptualization of the replacement system.

3.3 NIST Special Publication 800-160

One work in the field of Systems Security Engineering is such a critical component to the field of SSE that it warrants pointing out. The National Institute of Standards and Technology (NIST) defines its mission as such: "To promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve our quality of life."
[26]. As their name suggests, NIST develops standards for various industries of relevance to the nation.

The relevant standard for Systems Security Engineering (SSE) is NIST Special Publication 800-160 Volumes 1 and 2. The title of volume 1 is "Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems." The title of volume 2 is "Systems Security Engineering Cyber Resiliency Considerations for the Engineering of Trustworthy Secure Systems." These, in particular volume 1, are the primary sources for the following two sections.

3.4 Systems Security Engineering

Systems security engineering (SSE) is a sub-discipline of systems engineering. Figure 3.1 shows SSE in relation to systems engineering and other sub-disciplines of SSE. This master thesis falls into one of the Security Specialties in this diagram.

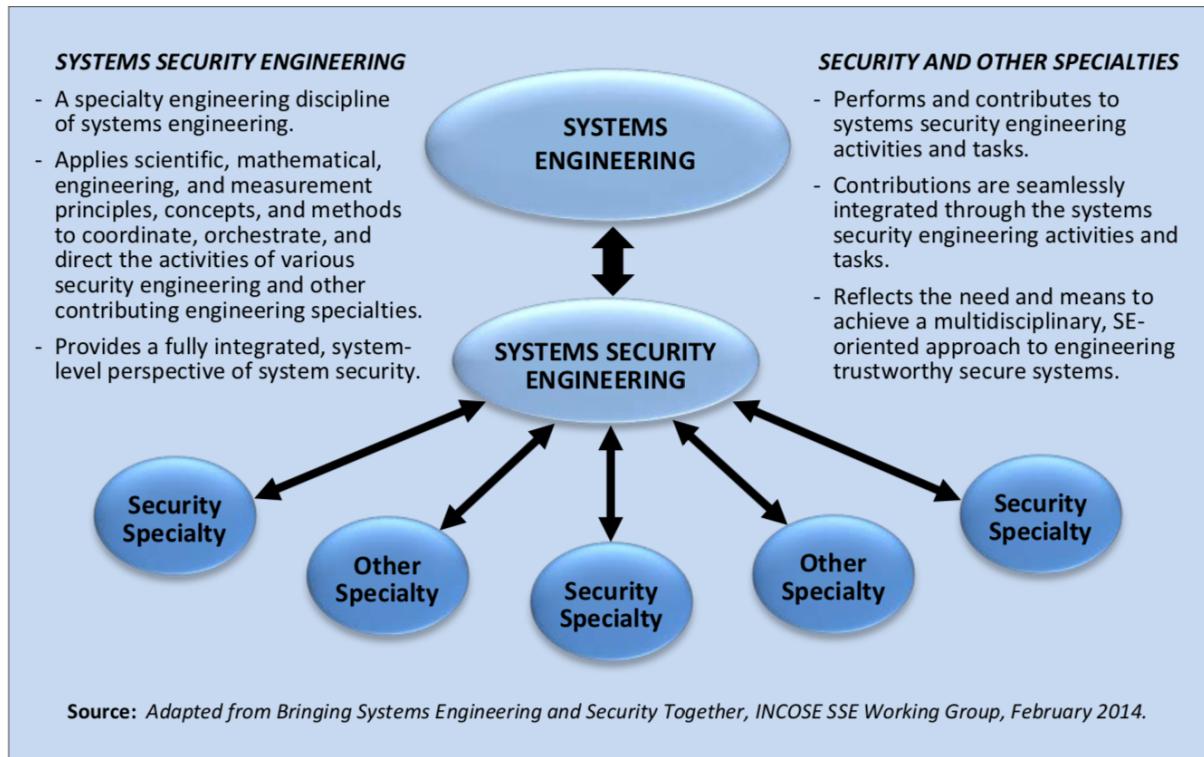


Figure 3.1: Systems security engineering in relation to systems engineering. (Image from NIST Special Publication 800-160: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.)

According to NIST Special Publication 800-160, "Systems security engineering focuses on the protection of stakeholder and system assets so as to exercise control over asset loss and the associated consequences." Three key concepts in SSE are stakeholder, asset, and unacceptable losses. In modeling the patrol base operations, this thesis first defines these key concepts.

Stakeholder The stakeholder governs the ConOps and conceptualization of the system. The stakeholder defines what the system should do. The stakeholder also defines what the system should not do and what are unacceptable losses. The

stakeholder for the patrol base operations are ultimately the U.S. military. But, this thesis has a different purpose, that of demonstrating specific security properties of the patrol base operations using CSBD. For this master thesis, the stakeholders can be thought of as those who are funding and managing this research (see the Acknowledgements section). The ConOps for this model of the patrol base operations require that the system should be modeled in a way amiable to verification of complete mediation.

Asset An asset is anything that is of value to the stakeholder. In the patrol base operations, this includes soldiers, equipment, and the mission. For this master thesis, the assets remain the same. But, they also include phases (or states) of the model. These can be thought of as sub-missions.

Unacceptable losses Unacceptable losses are self-defining. Unacceptable losses for the patrol base operations are defined broadly as any event that would cause the patrol base operation as a whole to abort. These are: contact with the enemy, casualties, a change in mission from higher-up. Unacceptable losses from the perspective of this master thesis include situations wherein complete mediation can not be verified. In these cases, the model should be revised until it does satisfy the property of complete mediation.

It is a critical objective of SSE to identify assets and unacceptable losses according to the stakeholder, and then design the system in a way that minimizes asset loss and avoids unacceptable losses. This thesis begins with an assumption that these are already explored in the original design of the patrol base operations (see Ranger Handbook [1]). This also means that this thesis assumes that the security properties of the patrol base operations are already built-in to the design. These assumptions are necessary because the goal of this thesis is not to design the patrol base operations, but to describe them in manner amiable to verification of complete mediation. Ideally, this would be applied to any future such operations during their inception.

Nevertheless, this master thesis includes the unacceptable losses described above in this model...because they are part of the patrol base operations. To cover the unacceptable losses, this master thesis models an escape-level secure state machine (SSM). If at any phase in the patrol base operations any authenticated principal (i.e., the platoon leader) reports an abortable event, the escape-level SSM will abort the patrol base operations. This includes casualties or unacceptable equipment failure. By creating one escape-level SSM, this thesis creates a modularized yet expandable treatment of unacceptable losses.

To further describe the model of the patrol base operations in the context of SSE, a few more concepts are necessary.

3.5 Systems Security Engineering Framework

NIST 800-160 describes the systems security engineering framework shown in figure 3.2 as "contexts within which systems security engineering activities are conducted." CSBD focuses primarily on demonstrating trustworthiness. Nevertheless, this master thesis also addressed other aspects of the framework.

The problem and solution phases for the patrol base operations take a different approach in the context of this master thesis. It is not our goal to outline the potential security threats and then to find solutions for them. This was, hopefully, already done when the patrol base operations were originally defined. Nonetheless, it is necessary to identify the problem and solution within the patrol base operations in order to verify their security properties.

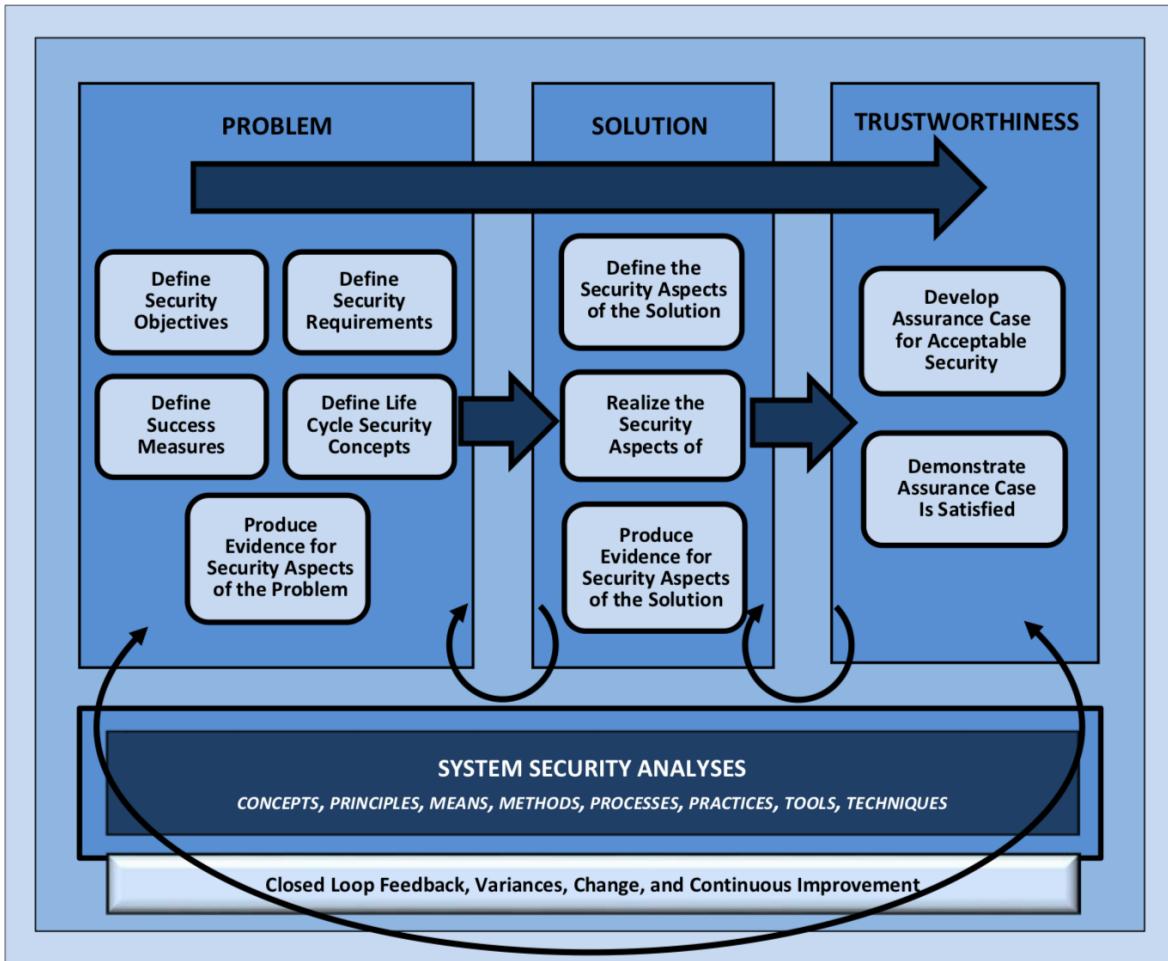


Figure 3.2: Systems security engineering Framework. (Image from NIST Special Publication 800-160: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.)

3.5.1 Problem

A problem (or goal) for any Ranger operation is suggested by the following statement in the Ranger Handbook [1]: "To survive on the battlefield, stealth, dispersion, and security is enforced in all tactical movements." Thus, enforcing security is a primary problem inherent in the patrol base operations. This is very relevant to the goal of this master thesis and more generally CSBD approach.

CSBD focuses on a subspecialty in SSE, that of complete mediation. In the conceptualization of any real system, CSBD would be applied in conjunction with other subspecialty areas of SSE. But, for this thesis, the problem at hand is to model the

patrol base operations in a way that is amiable to verification of complete mediation.

The problem inherent to the application of complete mediation is defining the objects to be accessed. Complete mediation refers to control over who or what has access to these objects. Thus, the protected objects must be clearly defined.

We the problem defined, this master thesis moves on to the solution phase.

3.5.2 Solution

The simplest solution to this problem is to model the patrol base operations as a hierarchy of secure state machines (SSM). The hierarchy simplifies the design of the solution by creating decreasing levels of abstraction with multiple modules at each level. This allows for focus on one level and one module at time. This essentially amounts to a system of systems approach. Furthermore, this allows for greater separation of work. After one level of the hierarchy is complete, CSBD is applied to that level while the next level of the hierarchy is being designed.

SSMs are readily defined with complete mediation embedded into their design. SSMs are discussed in detail in Chapter 6².

The objects to be accessed varies at each level of abstraction. At the top and most abstract level, the objects are the transitions. That is, control over the transitions between phases (or states) of the patrol base operations are to be protected. Objects are defined similarly for each module and at each level for the entire hierarchy of the model. These are deducible from the descriptions of each module (or SSM) in their appropriate section (see section 5.5.3 to start). For the sake of brevity, this section only

²In fact, a parametrizable model of an SSM³ was already designed and implemented in HOL prior to the start of this project. The constraints of this project suggested Improvements⁴ to this parametrizable SSM. Much to the chagrin of the author, a significant portion of the SSMs had to be redone with the new parametrizable SSM.

discusses the top level to illustrate the key concepts.

The accessors are the leaders designated at each level of abstraction. The best way to negotiate access to objects is by the principle of least privilege. This principle assigns privilege to only those who are deemed necessary and to no others. At the top level, the platoon leader (and only the platoon leader) is designated as the accessor. This means that only the platoon leader can issue commands to move-on to the next phase (or state) in the patrol base operations. Some of the other modules have more than one accessor. These accessors are typically assigned privileges to specific objects within the module. The rights of each accessor are determined by the policy.

In the real patrol base operations, soldiers recognize each other by face. Thus, it is sufficient to just authenticate the platoon leader at each level without requiring any passwords or other means of authentication⁵.

Policy is determined for each module and at each level of the patrol base operations model. The model includes not only "who" is authorized to do what, but also what preconditions are necessary prior to authorization. This is more complicated than authentication.

At the top level, there are only two requirements for the patrol base operations to move to the next phase of the operations: (1) the previous phase is complete, and (2) the platoon leader issues the command to move to the next phase. This creates a potential conflict between the goal of modularizing each component and creating a hierarchy wherein each level of the hierarchy expands upon the one above it. In general, If the top level requires some signal from the lower level that it is complete, then these two levels are not independent.

To solve this problem, an OMNI level module is created. The OMNI level module is

⁵The author and subject matter expert discussed authentication in the context of an accountability system wherein passwords or chips were required for authentication. But, we did not implement such authentication techniques in this project.

thought of as all-knowing. This module relays information from all modules as needed. With this construct, the top level does not need to "know" if the lower level module is complete. It only needs to know that the OMNI level says that it is complete. The OMNI level is given the authority to determine when any module is complete.

With this solution in mind, the top level policy looks like this.

if OMNI-Level says the previous phase is complete⁶, and

Platoon Leader says move to the next phase

then Platoon Leader has the authority to move to the next phase

The most useful property of the SSM is that complete mediation is made a condition for transitions among states. That is, to transition from one state to another the accessor (requestor) must be properly authenticated and authorized. For example, at the top level, transition commands in an SSM look like this.

if Platoon Leader's identity is authenticated, and

Platoon Leader says move to the next phase, and

Platoon Leader has the authority to move to the next phase

then move to the next phase.

The first line ensures authentication of the Platoon Leader. The second line is a request from the Platoon Leader to access the object (i.e., move to the next phase). The third line is the output of a policy shown in the box above. The final line is the conclusion which justifies transition to the next phase.

Note that this is an "if and only if" conditional. This means that if the Platoon Leader is authenticated, authorized on, and issues the command then the command is executed. It also means the converse. If the command is executed, then the Platoon Leader is authenticated, authorized on, and issues the command to do so. The former is required for transitions to occur. The later (converse) is critical for accountability.

3.5.3 Trustworthiness

NIST SP 800-160 states that *assurance cases* are the basis for trustworthy context. It states, "An assurance case is a well-defined and structured set of arguments and a body of evidence showing that a system satisfies specific claims with respect to a given quality attribute." The key points here are that the assurance case must be clearly defined and also clearly demonstrated.

Trustworthiness in the context of this master thesis and more generally CSBD means that all protected objects are accessed if and only if that accessor is both authenticated and authorized. The processes of authentication and authorization are clearly defined in the Solution section above. Therefore, what remains is to demonstrate that actually does what it claims to do. This is where the access-control logic (ACL) and computer-aided reason take center stage.

A key concern in Systems Security Engineering (SSE) is measuring trustworthiness. For this thesis and CSBD, trustworthiness is measured by whether or not the system satisfies the principle of complete mediation. The evidence that it does is formally verified proofs using an access-control logic (ACL). The ACL is described in detail in chapter 4. Application of ACL to the patrol base operations follows that chapter.

In essence, the ACL formally describes the informal logic shown in the two boxes in the Solution section. This logic is implemented in the Higher Order Logic Theorem Prover (HOL) Interactive Theorem Prover. In addition, the hierarchy of state machines are described in HOL. With these, it is possible to prove theories in HOL that pertain to complete mediation. These are formal proofs generated by a trusted theorem proving assistant.

With these proofs, it is accepted that the model of the patrol base operations satisfies the principle of complete mediation. These are considered demonstrated proofs, or

assurance cases,

3.6 Verification & Documentation

3.6.1 Verification

Verification amounts to proof of trustworthiness. NIST SP 800-160 often refers to "security-focused" verification. Verification is built-in to the CSBD method. It focuses on the use of authentication and authorization in the design of the system. CSBD demonstrates its usefulness not only as a guide to complete mediation, but also as a tool that applies formal methods using computer-aided reasoning to verify any claims of complete mediation. This is referred to as formal verification. Thus, CSBD is a complete approach to designing trustworthy systems.

3.6.2 Documentation

A common concept in science and engineering is "if you didn't document it then it didn't happen." Documentation is specifically important to claims of trustworthiness because such claims are not always readily apparent in the system. Systems may run smoothly without adequate security ... until something finally goes wrong? Then, everyone scrambles to find the cause of the problem.

Documentation is often the least exciting aspect of systems engineering. But, documentation is one of the most important activities in the systems engineering processes. Most people find that documentation reveals aspects of the system that are overlooked. It checks the accuracy of the work. Documentation describes the system for posterity and communicates the system for the benefit of others.

3.6.2.1 Accountability

Internet Hall of Fame Inductee Vint Cerf commented on the importance of accountability in SSE at a recorded conference in 2016. The conference titled "Exploring the Dimensions of Trustworthiness: Challenges and Opportunities" is presented by NIST. Accountability is incredibly important. When things go wrong, the first thing that people do is ask "what happened and who is responsible."

Documentation is a great accountability tool. Documentation demonstrates that the system is realized to the satisfaction of the stakeholders. If something goes wrong, documentation aids in tracing the source of the problem. It saves time and resources to identify and eliminate areas where security is adequately demonstrated. Furthermore, everyone wants to document their work for their protection, in case something does go wrong.

Documentation is a primary component of CSBD. The HOL formal proofs are all printable. It is standard to generate a printout of all theories and generate a pdf.

3.6.2.2 Reproducibility

In addition to formal verification and documentation of complete mediation, all proofs are fully reproducible. The access-control logic proofs can be worked-out by hand. In addition to HOL, ACL can also be implemented in other theorem provers.

Chapter 4

Certified Security by Design (CSBD)

&

Access-Control Logic (ACL)

4.1 Certified Security by Design (CSBD)

In 1970 The Rand Corporation published a report[27] 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. CSBD focuses on the design phase of systems engineering, applying formal methods to verify that a system satisfies the principle of complete mediation.

More specifically, CSBD is a method for formally verifying and documenting the security properties of a systems. It focuses on designing systems that satisfy the principle of complete mediation. 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 guidelines 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."

ACL is described below. A description of the implementation of ACL in HOL is also described below. The implementation of ACL is also presented in the appendix in pretty-printed. The actual HOL code is included with the other files for this master thesis. It is included under /MasterThesis/HOL/ACL.

4.2 Access-Control Logic (ACL)

4.2.1 ACL: A Command and Control (C2) Calculus

ACL is a logic for reasoning about access to object. In the jargon of the day it is a command and control (C2) calculus¹. All the axioms and definitions described above are implemented in the ACL. The actual code is included with the files for this thesis. A pretty-printed print-out of the ACL is included in appendix A.

¹command 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**:

$$\text{Princ} ::= \text{PName} / \text{Princ} \& \text{Princ} / \text{Princ} \mid \text{Princ}$$

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 $P \text{ says } \varphi$, 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 $P \text{ controls } \varphi$, 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) define the syntax (or grammatical structure) of the propositional logic. They are valid statements in the ACL. All statements must be a WFF. The text book defines the set of WFFs using the identifier **Form**:

$$\text{Form} ::= \text{PropVar} / \neg \text{Form} / (\text{Form} \vee \text{Form}) /$$

$$\begin{aligned}
 & (\text{Form} \wedge \text{Form}) / (\text{Form} \supset \text{Form}) / (\text{Form} \equiv \text{Form}) / \\
 & (\text{Princ} \Rightarrow \text{Princ}) / (\text{Princ says Form}) / (\text{Princ controls Form}) / \\
 & (\text{Princ reps Princ on Form}^2)
 \end{aligned}$$

This is a recursive definition. ***Prop Var*** is a propositional variable. The symbols \neg , \vee , \wedge , \subset , and \equiv are the standard set and logical symbols. They represent "not", "or", "and", "implication", and "equivalence", respectively.

4.2.5 Kripke Structures & Semantics

Kripke structures are named after Saul Kripke. Kripke is an influential figure in logic and philosophy. He is recognized, in particular, for inventing the Kripke semantics for modal logic [28].

Whereas WFFs define the syntax of the propositional logic, Kripke semantics describe the semantics. Semantics refers to the meaning of statements. In propositional logic, WFFs can be either true or false.

4.2.5.1 Kripke Structures

Before defining the Kripke semantics, the concept of a Kripke structure is necessary. A Kripke structure primarily 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 the following: a set of worlds W ; a function I called the assignment function that maps propositions to worlds, and ; a function J that maps principals to relations on worlds,

²The last line is from [4].

where the relation is called the accessibility relation. Formally, these are defined as follows (definition 2.1 in the text):

- W is a nonempty set, whose elements are called worlds.
- $I : \mathbf{PropVar} \rightarrow \mathcal{P}(W)$ is an interpretation function that maps each propositional variable to a set of worlds.
- $J : \mathbf{PName} \rightarrow \mathcal{P}((W \times W))$ is a function that maps each principal name to a relation on worlds.

One way to think of Kripke structures is as a logic on multiple worlds or possible states. For example, consider two planets Earth and Tatooine³. Let φ be the proposition "this is planet Tatooine" and θ be the proposition "this is a planet." Then

$I(\varphi) = \{\text{Tatooine}\}$ and $I(\theta) = \{\text{Earth}, \text{Tatooine}\} = W$, where W is the set of all worlds ($\{\text{Earth}, \text{Tatooine}\}$). Furthermore, let "Luke" and "Lea" and "Uncle Owen" be principals. Now consider the following: Luke can get from Earth to Earth and from Tatooine to Earth (and vis-a-vis). He can also get from Tatooine to Tatooine. Lea can get from Earth to Earth and from Tatooine from Earth. But, she can not get from Tatooine to Tatooine. Uncle Owen can only get from Tatooine to Tatooine. He can not get to Earth. Then, $J(\text{Luke}) =$

$$\{(\text{Earth}, \text{Earth}), (\text{Earth}, \text{Tatooine}), (\text{Tatooine}, \text{Earth}), (\text{Tatooine}, \text{Tatooine})\}.$$

$$J(\text{Lea}) = \{(\text{Earth}, \text{Earth}), (\text{Earth}, \text{Tatooine}), (\text{Tatooine}, \text{Earth})\}. \text{ And,}$$

$$J(\text{UncleOwen}) = \{(\text{Tatooine}, \text{Tatooine})\}. \text{ With } W, J, \text{ and } I \text{ defined, this forms a Kripke structure.}$$

Kripke structures are necessary to define the ACL properly. In particular, they are necessary to define the concepts of "satisfies" and "soundness" which follow. However, an in-depth understanding of Kripke structures is not necessary to understand the work in this master thesis.

³Tatooine is the fictional home planet of Luke Skywalker. Tatooine has two suns. [29]

4.2.5.2 Kripke Semantics

The Kripke semantics define the meanings of $\llbracket \cdot \rrbracket$ for Kripke structures. The semantics can be thought of as an evaluation function for a particular Kripke $\mathcal{M} = \langle W, I, J \rangle$. Figure 4.1 shows the Kripke semantics. The subscript \mathcal{M} signifies that the evaluation function is defined for a particular Kripke structure. This means there is a separate evaluation function for each Kripke structure.

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

Figure 4.1: Kripke semantics. Image taken from *Access Control, Security, and Trust: A Logical Approach*[3]

For example, for the Kripke structure described above, $\mathcal{E}_{\mathcal{M}}[\varphi] = \{Tantooine\}$ and $\mathcal{E}_{\mathcal{M}}[\theta] = \{Earth, Tantooine\} = W$ (the set of all worlds). As an another example, $\mathcal{E}_{\mathcal{M}}[\text{Luke says } \varphi] = \{Earth, Tatooine\}$ ⁴

4.2.5.3 Satisfies

The "satisfies" condition applies to a particular Kripke structure $\mathcal{M} = \langle W, I, J \rangle$. It is said the \mathcal{M} satisfies some proposition φ if the evaluation function $\mathcal{E}_{\mathcal{M}}[\varphi] = W$ (the set

⁴Think about it.

of all worlds) for \mathcal{M} . In other words, φ is true in all worlds of \mathcal{M} . Symbolically, this is denoted as $\mathcal{M} \models \varphi$. The statement \mathcal{M} does not satisfy φ is denoted as $\mathcal{M} \not\models \varphi$.

In the example above, the Kripke structure $\mathcal{M} \models \theta$, where θ = the proposition "this is a planet."

4.2.5.4 Soundness

Whereas "satisfies" describes a property of a Kripke structure \mathcal{M} , "soundness" describes a property of all Kripke structures.

Soundness refers to the logical consistency of inference rules. Inference rules consist of a set of hypothesis $\{H_1, H_2, \dots, H_n\}$ and a conclusion.

$$\frac{H_1, H_2, \dots, H_n}{C}$$

An inference rule is said to be sound if for every Kripke structure \mathcal{M} that satisfies all the hypothesis, the conclusion is true. In other words, the inference rule is sound if and only if $\forall H_i, \mathcal{M} \models H_i \rightarrow \mathcal{M} \models C$.

Soundness is verified by formal proofs that employ axioms, tautologies, and sound inference rules that are already proved.

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*[3].

$$\begin{array}{c}
P \text{ controls } \varphi \stackrel{\text{def}}{=} (P \text{ says } \varphi) \supset \varphi \quad P \text{ reps } Q \text{ on } \varphi \stackrel{\text{def}}{=} P \mid Q \text{ says } \varphi \supset Q \text{ says } \varphi \\
\\
\text{Modus Ponens} \quad \frac{\varphi \quad \varphi \supset \varphi'}{\varphi'} \quad \text{Says} \quad \frac{\varphi}{P \text{ says } \varphi} \quad \text{Controls} \quad \frac{P \text{ controls } \varphi \quad P \text{ says } \varphi}{\varphi} \\
\\
\text{Derived Speaks For} \quad \frac{P \Rightarrow Q \quad P \text{ says } \varphi}{Q \text{ says } \varphi} \quad \text{Reps} \quad \frac{Q \text{ controls } \varphi \quad P \text{ reps } Q \text{ on } \varphi}{P \mid Q \text{ says } \varphi} \\
\\
\& \text{ Says (1)} \quad \frac{P \& Q \text{ says } \varphi}{P \text{ says } \varphi \wedge Q \text{ says } \varphi} \quad \& \text{ Says (2)} \quad \frac{P \text{ says } \varphi \wedge Q \text{ says } \varphi}{P \& Q \text{ says } \varphi} \\
\\
\text{Quoting (1)} \quad \frac{P \mid Q \text{ says } \varphi}{P \text{ says } Q \text{ says } \varphi} \quad \text{Quoting (2)} \quad \frac{P \text{ says } Q \text{ says } \varphi}{P \mid Q \text{ says } \varphi} \\
\\
\text{Idempotency of } \Rightarrow \quad \frac{}{P \Rightarrow P} \quad \text{Monotonicity of } \Rightarrow \quad \frac{P' \Rightarrow P \quad Q' \Rightarrow Q}{P' \mid Q' \Rightarrow P \mid Q}
\end{array}$$

Figure 4.2: The ACL inference rules. Image taken from *Access Control, Security, and Trust: A Logical Approach*[3]

4.2.7 Complete mediation

Fundamental to this work is the concept of complete mediation (discussed in section ??). 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 figure 4.2 and shown again here in figure 4.3.

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

Figure 4.3: The *Controls* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[3]

ACL refers to the left statement as an authorization⁵. The principal P controls (is authorized on) some action φ . ACL refers to the right statement in this inference rule as a request⁶. 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 $P \text{ controls } \varphi$ and $P \text{ says } \varphi$ then φ is true.

⁵or a *control* in the C2 calculus

⁶or a *command* in the C2 calculus

The *Controls* rule is sufficient for basic authorization involving one principal and some action. It is the only rule applied in this master thesis. But, there are more complicated rules that allow for additional authentication and authorization schemes.

$$Reps \quad \frac{Q \text{ controls } \varphi \quad P \text{ reps } Q \text{ on } \varphi \quad P | Q \text{ says } \varphi}{\varphi}$$

Figure 4.4: The *Reps* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[3]

The *Reps* (shown in figure 4.4) rule also demonstrates complete mediation. It follows a similar logic. But, this rule has two principals, in this case P and Q. The idea is that Q controls (is authorized on) some action φ . And also, P represents Q on that action. In the ACL, this means that $P \text{ Reps } Q \text{ on } \varphi$. Thus, $P \text{ says } \varphi$ is irrelevant because P does not have authority on φ . But, $P | Q \text{ says } \varphi$ (short for, $P \text{ says } Q \text{ says } \varphi$) is relevant because P represents Q and Q has authority. Spelled out, the rule follows as such: if $Q \text{ controls } \varphi$ and $P \text{ reps } Q \text{ on } \varphi$ and $P | Q \text{ on } \varphi$ then φ is true.

The *Reps* rule represents a principal acting in a role. For example, soldier G.I. Jane may be acting as the Platoon Leader. In this situation, it is the Platoon Leader (Q) who is granted authority over the action φ . But, it is G.I.Jane (P) acting in the role of Platoon Leader who (Q) is actually issuing the command φ .

$$Derived Speaks For \quad \frac{P \Rightarrow Q \quad P \text{ says } \varphi}{Q \text{ says } \varphi}$$

Figure 4.5: The *Derived Speaks For* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[3]

The *Derived Speaks For* rule also allows for variation in authentication and authorization. This rule is shown in figure 4.5. This also uses two principals and deals mainly with the concept of jurisdiction and authentication. In this case, the principal P is "speaking for" the principal Q. Again, the symbol \Rightarrow means "speaks for." Thus, if $P \Rightarrow Q$ and $P \text{ says } \varphi$ then we believe that $Q \text{ says } \varphi$.

The most common use of the *Derived Speaks For* rule is in jurisdictional case. For example, the United States Army has jurisdiction over who serves in the role of Platoon Leader. The U.S. Army may, in turn, issue i.d. cards with a soldier's photo (or in the future chips) that say, for example, soldier G.I.Jane is who she says she is. If we believe the i.d. card is genuine and it belongs to G.I.Jane, then the i.d. card speaks for G.I.Jane. So, if the i.d. card says issue G.I.Jane her equipment, then it follows that G.I.Jane says to issue her the equipment. It is somewhat more complicated, because the U.S. Army has jurisdiction over who gets what equipment. The i.d. card is not saying that G.I.Jane has control over her equipment. It only says that she is asking for the equipment.

The *Derived Speaks For* rule is critical to authentication, whereas the *Controls* and *Reps* rules are critical for authentication.

4.3 ACL in HOL

The material in this section is adapted from *Access Control, Security, and Trust: A Logical Approach*[3] and *Certified Security by Design Using Higher Order Logic*[4]. In this section, the former is referred to as "the text" and the later is referred to as "the manual."

This section describes how the access-control logic (ACL) is implemented in the Higher Order Logic (HOL) Interactive Theorem Prover. The goal is to describe the implementation in sufficient detail that the reader can understand the descriptions of the patrol base operations that follow in this thesis.

4.3.1 Principals

Figure 4.6 shows the HOL representations for principals (*Princ*).

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

Figure 4.6: The HOL implementation of principle (*Princ*). Image from *Certified Security by Design Using Higher Order Logic*[4]

The concept of types are important in HOL. They are discussed in the background chapter in section 2.5. *Princ* is an algebraic data type.

The first thing to notice about *Princ* is that there are three definitions just as there are in the definition of *Princ* in section 4.2.2. The first line within the definition above corresponds to *Prop Var*, the second corresponds to **Princ & Princ**, and the third corresponds to **Princ | Princ**. In HOL the infix & operator is represented with the prefix meet operator. The infix | operator is represented with the prefix quoting operator.

The primary difference between datatype definitions in HOL and ML (the metalanguage in which HOL is implemented) is that HOL does not use the keyword "of" before the type definition. Thus, the first line of this definition is Name ' apn and not Name of ' apn. As in ML, forward tick marks (apostrophes) always precede a type variable.

In the definition for *Princ*, Name is called the type constructor. The result of the constructor and a concrete type or type variable results in something of type *Princ*. Examples of principals defined in HOL take the following form.

NamePlatoonLeader, or

(*NamePlatoonLeader*) ` meet ` (*NamePlatoonSergeant*), or

$(NamePlatoonLeader)` quoting` (NamePlatoonSergeant).$

Prefix operators may be used as infix operators if they are surrounded by a backtick `` (typically located near the esc-key). This master thesis focuses on principals defined using the `Name PlatoonLeader` construction.

4.3.2 Well-Formed Formulas

The HOL representation of well-formed formulas (WFFs) are shown in figure 4.7.

```

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

```

Figure 4.7: The definition for **Form** in HOL. *Certified Security by Design Using Higher Order Logic*[4]

Form is the datatype definition. TT and FF are the ACL representations of true and false, respectively. notf, andf, orf, impf, eqf, says, speaks_for, controls,

and `reps` are all the prefix version of the infix operators shown in the definition of **Form** in section 4.2.4. The additional elements of the **Form** (from `domi` to the end) refer to integrity and security levels, which are not discussed in this master thesis.

The type definitions that follow the operator are relevant and show-up everywhere in the code. It is useful to dissect one of them. Consider the `andf` operator.

```
(andf) (('aavar, 'apn, 'il, 'sl) Form)

((('aavar, 'apn, 'il, 'sl) Form).
```

`((('aavar, 'apn, 'il, 'sl) Form)` is the type signature for another **Form**. The component types of a **Form** are a proposition (`'aavar`), a principal (`'apn`), an integrity level (`'il`), and a security level (`'sl`). The prefix operator `andf` then takes two **Forms** each of type `((('aavar, 'apn, 'il, 'sl) Form)`. In this thesis, `'aavar` and `'apn` are the only types that are specified.

4.3.3 Kripke structures

Kripke structures are part of the implementation of the ACL in HOL because they are necessary to prove the “satisfies” and “soundness” properties of the ACL. Nevertheless, the model of patrol base operations does not require any definition of Kripke structures. Therefore, they are only briefly described here. The HOL implementation of the Kripke structure is shown in figure 4.8.

Kripke structures are defined previously as a three-tuple. The Kripke structure here has four components (each surrounded by parentheses). This representation in HOL is very different from that described in section 4.1.

```

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

```

Figure 4.8: The HOL implementation of a Kripke structure (*Princ*). Image from *Certified Security by Design Using Higher Order Logic*[4]

In figure 4.8, the first set of parenthesis represents the assignment function more-or-less. It takes a proposition and world as arguments. If the proposition is true in that world, then the function returns true, otherwise it returns false. The second pair of parenthesis is similar to the accessibility function. It takes a principal, a world, and another world. The function returns true if the second world is accessible from the first for this particular principal, otherwise it returns false. The last two pairs of parentheses represent integrity and security levels. These are not discussed in this master thesis.

The key take away for the reader is to recognize Kripke structures in the HOL code. Kripke structures will either be fully typed or abbreviated. These two forms are shown below, respectively.

$$(M, Oi, Os)$$

M: ('prop, 'world, 'pName, 'Int, 'Sec) Kripke, (Oi: 'Int po), (Os:'Sec po)

The actual Kripke structure is represented by M and the integrity and security levels are represented by Oi and Os.

4.3.4 ACL Formulas

It remains now to define the access-control logic (ACL) formulas in HOL specifically. These are shown in figure 4.9.

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
<i>Alice</i> says $\langle jump \rangle$	Name Alice says prop jump
<i>Alice & Bob</i> says $\langle stop \rangle$	Name Alice meet Name Bob says prop stop
<i>Bob Carol</i> says $\langle run \rangle$	Name Bob quoting Name Carol says prop run
<i>Bob</i> controls $\langle walk \rangle$	Name Bob controls prop walk
<i>Bob</i> reps <i>Alice</i> on $\langle jump \rangle$	reps (Name Bob) (Name Alice) (prop jump)
<i>Carol</i> \Rightarrow <i>Bob</i>	Name Carol speaks_for Name Bob

Figure 4.9: The ACL formulas in HOL. Image taken from *Access Control, Security, and Trust: A Logical Approach*[3]

Triangular brackets enclose propositions. Thus, in the HOL representation of the ACL, a proposition is coded as follows:

```
prop command
```

A request in HOL is coded as:

```
(Name PlatoonLeader) says (prop command)
```

Parentheses are used when necessary⁷. A statement of authority is coded as:

```
(Name PlatoonLeader) controls (prop command)
```

These are the primary types of ACL formulas used in this master thesis. The others are readily derivable from figure 4.9.

⁷The need for parentheses is well-defined. But, the author finds the best approach to parentheses is trial and error and generosity.

4.3.5 Kripke Semantics: The Evaluation Function

The Kripke semantics described in section 4.2.5.2 and shown in figure 4.1 are also implemented in HOL. This function is lengthy and the details are beyond the scope of this master thesis. Part of the definition is shown 4.10. The entire function can be found in appendix A in the section on aclsemantics. The reader should note that the definition is defined for all Kripke structures as $\forall Oi Os M$.

[Efn_def]

```

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

```

Figure 4.10: The HOL implementation of the Kripke semantics (evaluation function). Image from *Certified Security by Design Using Higher Order Logic*[4]

4.3.6 Satisfies

The implementation of the "satisfies" property is shown in figure 4.11

The first part of the definition is $\forall M Oi Os f..$. This is the universal quantifier applied to all Kripke structure, integrity levels, security levels, and functions. The next

```
[sat_def]
 $\vdash \forall M\ Oi\ Os\ f.\ (M, Oi, Os) \text{ sat } f \iff (\text{Efn } Oi\ Os\ M\ f = \mathcal{U}(:'world))$ 
```

Figure 4.11: The HOL implementation of the "satisfies" property. Image from *Certified Security by Design Using Higher Order Logic*[4]

part of the definition is $(M, Oi, Os) \text{ sat } f$. This is what is being defined. The final part of the definition is $(\text{Efn } Oi\ Os\ M\ f = \mathcal{U}(:'world))$. This part states that the function f must evaluate to all worlds \mathcal{U} .

4.3.7 Soundness

The definition for "satisfies" in the HOL is similar to the definition for "soundness" described in section 4.2.5.4. In that section and the section preceding it, "satisfies" applied to a particular Kripke structure. But, definition for "satisfies" above is valid for all Kripke structures. To extend this to "soundness" it is only necessary to account for multiple hypotheses. This is of the form shown in figure 4.12

$$\vdash \forall M\ Oi\ Os.\ (M, Oi, Os) \text{ sat } H_1 \Rightarrow \dots \Rightarrow (M, Oi, Os) \text{ sat } H_k \Rightarrow (M, Oi, Os) \text{ sat } C,$$

Figure 4.12: The HOL representation of the "soundness" property. Image from *Certified Security by Design Using Higher Order Logic*[4]

A thorough description of the ACL in its entirety is beyond the scope of this master thesis. All the theory files necessary to use this HOL implementation of the ACL are included with the files with this master thesis. A pretty-printed EmitTeX version of the code is included in appendix A.

Chapter 5

Patrol Base Operations

5.1 Motivation

The aim of this master thesis is to demonstrate proof of applicability (or failure) of CSBD to non-automated, human centered systems. Military operations are a great example because safety and security are critical to mission success. The patrol base operations are an excellent example of non-automated, human-centered military operations.

5.2 Patrol Base Operations

Patrol base operations are described in the United States Army Ranger Handbook [1] in chapter 7 (2017 edition). The mission activity specification for the patrol base operations are shown in table 5.1

This master thesis applies properties of complete mediation to a model of these patrol

Patrol Base Operations: Mission Activity Specification		
Purpose	A system to	establish a security perimeter when a squad or platoon halts for an extended period of time
Method	by means of	planning, reconnaissance, security, control, and common sense
Goal	in order to	<ul style="list-style-type: none"> • avoid detection • hide a unit during a long, detailed reconnaissance • perform maintenance on weapons, equipment, eat, and rest • plan and issue orders • reorganize after infiltrating an enemy area • establish a base from which to execute several consecutive or concurrent operations

Table 5.1: Mission Activity Specification for Patrol Base Operations. Adapted from the U.S. Army Ranger Handbook 2017 [1].

base operations.

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 Visio¹. The result of doing this is a hierarchy of secure state machines (SSMs). (SSMs are described in section 6.2.)

5.4 Overview of The Hierarchy of Secure State Machines

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.

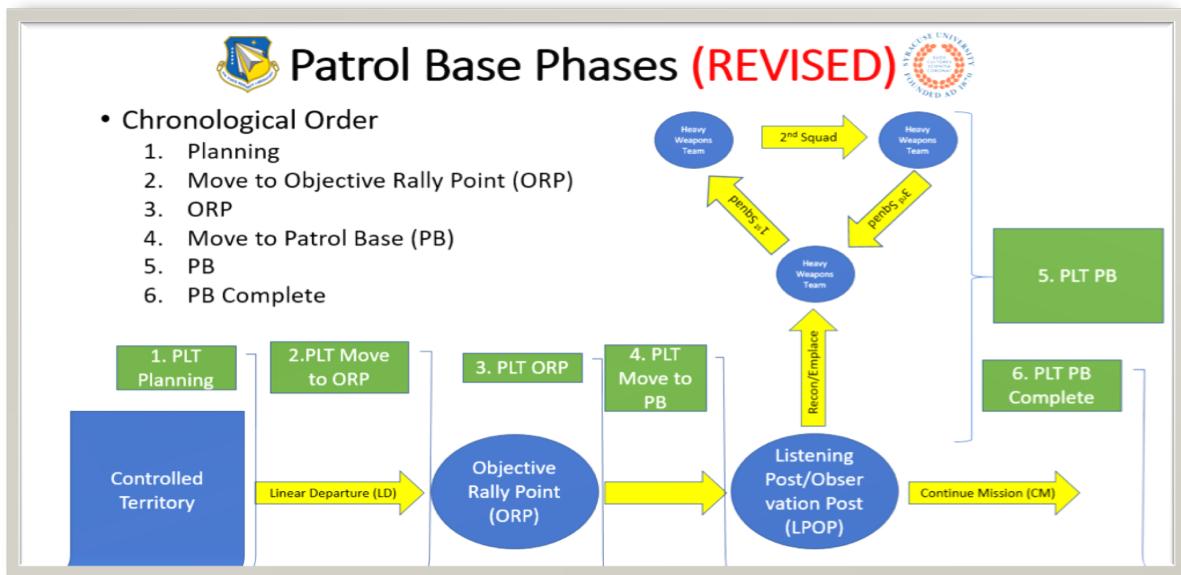


Figure 5.1: A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel as part of the research involved in this master thesis. Shall I get his permission?

The diagram describes a chronological order of abstract phases (modeled as states) of

¹This 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.

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 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 5.5.6.1) 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 only 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 operations².

²Otherwise, the author would not graduate because the patrol base operations are huge.

5.5 Hierarchy of Secure State Machines

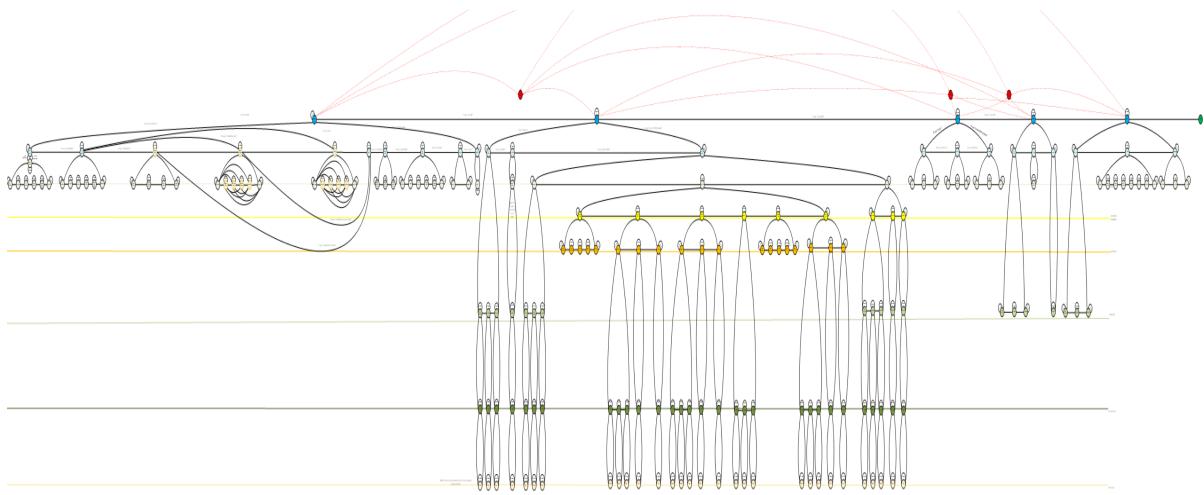


Figure 5.2: Diagrammatic description of patrol base operations as a hierarchy of secure state machines. (Generated by Jesse Nathaniel Hall.)

5.5.1 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.

This is actually only a partial representation of the patrol base operations. The middle section that extends to the bottom of the diagram represents a vertical slice through the model of the patrol base operations. For most sections, the diagram does not extend below to the yellow horizontal line spanning the diagram.

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 it 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.

5.5.2 Descriptions of Individual Modules

Each module is described diagrammatically in the following sections. They all follow a similar pattern. The general pattern is discussed in this section. Exceptions are discussed along with the diagrammatic descriptions for each individual module. (Note also that "module" and "SSM" are used interchangeably in the following sections.)

Flow Each module follows a sequential pattern. It starts at one state and then flows sequentially to the end of the module. Each module has a set of principals who are authorized on some set of transitions (or commands). Each module has its own security policy that dictates the conditions under which transition requests are granted.

Requests And Security Policies Principals make requests to transition from one state to another. Requests are of the form *Principal says command*.

Transitions at one level require confirmation of completion of that state at the lower level. For example, the PLAN_PB state at the top level is the basis for the less abstract ssmPlanPB secure state machine one level below it. Before the top level can transition from the PLAN_PB state to the MOVE_TO_ORP state, ssmPlanPB must be in the COMPLETE state.

But, encapsulation requires that each module be isolated from the other. To accommodate this, an OMNI level principal communicates when a sub level is complete. Thus, transitions from one state to another require a statement from OMNI and a request from an authorized authority. An example is *OMNI says ssmPlanPBComplete* and *PlatoonLeader says crossLD*.

The security policy for each module has a policy for OMNI and a policy for all authorized principals. For OMNI, the security policy contains the clause *OMNI controls*

omniCommands. In the example above, the command `ssmPlanPBComplete` is defined as an `omniCommand`. For everyone else, the security policy has clauses of the form $ssmPlanPBComplete \ impf (PlatoonLeader \ controls \ crossLD)$. This is sufficient to prove complete mediation for transitions.

Diagrammatic Description The colors of the states in the diagrams correspond to their colors in the overall squished diagram shown in figure 5.2.

All lines represent allowable transitions with an arrow indicating the direction of the transition. Each line is annotated with the appropriate ACL request (or command). The last line in each module is an exception. It connects the `COMPLETE` state to the initial state. This line is not annotated. It is not an actual transition but an indicator that the completion of this lower-level SSM links to the higher-level SSM. It is implemented in the SSM above it as a signal from the OMNI level³ that the lower-level SSM is complete⁴.

Naming conventions What follows are the naming conventions for the diagrams. These also apply to the HOL implementation of the SSMs.

state: all capital letters with underscores representing spaces. Examples include:

`MOVE_TO_ORP`, `PLAN_PB`, etc.

commands (or requests): first letter is lower case. The remaining letters toggle with a capital letter for each new word. Examples include: `moveToORP`, `receiveMission`, etc. Furthermore, all commands take the name of the next state. For example, the transition from the state `MOVE_TO_PB` to `CONDUCT_PB` is `conductPB`. The transition from the state `COMPLETE_PLAN` to

³OMNI is all knowing.

⁴Note, it is possible that the output for the transition to the `COMPLETE` state is *OMNI says ssmPlanPBComplete*. This may or may not have been implemented in the HOL proofs at the time this master thesis was complete.

ISSUE_OPORD is issueOPORD. The only exception is the transition from the top level state PLAN_PB to the next state MOVE_TO_ORP. The command for this transition is crossLD and not moveToORP.

principals: all begin with a capital letter then follow the convention for commands (or requests). Examples include: PlatoonLeader, PlatoonSergeant, etc.

ACL transition requests: all are of the form *Principal says command*. Examples include: *PlatoonLeader says moveToORP*, *PlatoonSergeant says actionsIn*, etc.

5.5.3 OMNI-Level

The OMNI level is not represented in the Visio diagram. It is not really a level. More specifically, the OMNI level represents an imaginary all-knowing entity. The main purpose of this entity is to relay messages from one SSM to another. This allows for greater encapsulation of the modules.

In all SSMs, OMNI is a principal who has authority over OMNI level commands. These commands communicate the completion of a lower-level SSM. For example, at the top level, before the Platoon Leader can transition from the PLAN_PB state to the MOVE_TO_ORP state, he must receive the command *OMNI says ssmPlanPBComplete*. The top level security policy contains the clause *OMNI controls ssmPlanPBComplete*. The *Controls* rule discussed in section 4.2.6 then allows the Platoon Leader to conclude that the lower-level is complete.

5.5.4 Escape

A diagram of the escape level is shown in figure 5.3. The purpose of the escape level is to model situation wherein the patrol base operations must be aborted.

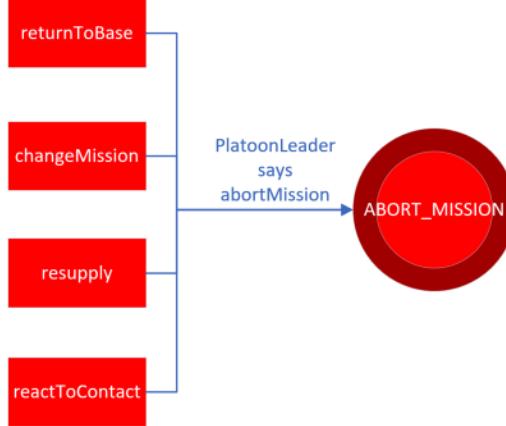


Figure 5.3: Escape level diagram.

This is also not really a level, nor is it an SSM. The square boxes in the diagram represent communication from outside the module. They represent external signals from the OMNI level. The only state is the ABORT_MISSION state. But, one state is not sufficient for an SSM⁵.

The abortable conditions are *returnToBase*, *changeMission*, *resupply*, and *reactToContact*. OMNI receives information from somewhere else that one of these conditions is true, for example *returnToBase*. In response, OMNI submits a request to the escape level: *OMNI says returnToBase* (etc.). The security policy for the escape level contains the clause *OMNI controls returnToBase*. Using the *Controls* rule discussed in section 4.2.6, the proposition *returnToBase* must be true.

Now, the Platoon Leader makes a request to abort the mission: *PlatoonLeader says abortMission*. Another clause in the security policy is *returnToBase impf PlatoonLeader controls abortMission*. There is now sufficient information to justify aborting the mission.

⁵This is not necessarily a rule. But, for the purposes of this master thesis it is a rule.

5.5.5 Top Level

The top level for the hierarchy of SSMs is shown in figure 5.4. This is a linearized version of figure 5.1.

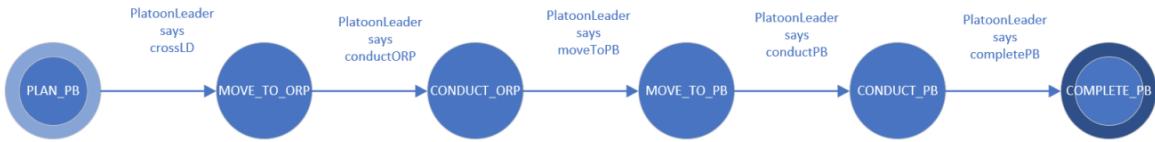


Figure 5.4: Top level diagram.

The details are discussed along with figure 5.1. The only difference in this diagram is that the state names are all capitalized with underscores substituted for spaces.

5.5.6 Horizontal Slice

The horizontal slice is an expansion of the states in the top level SSM. Each state save for the COMPLETE_PB state is expanded into an SSM. Each of these SSMs is described in the following sections.

5.5.6.1 ssmPlanPB

The top level PLAN_PB state is expanded into the ssmPlanPB SSM and shown in figure 5.5.

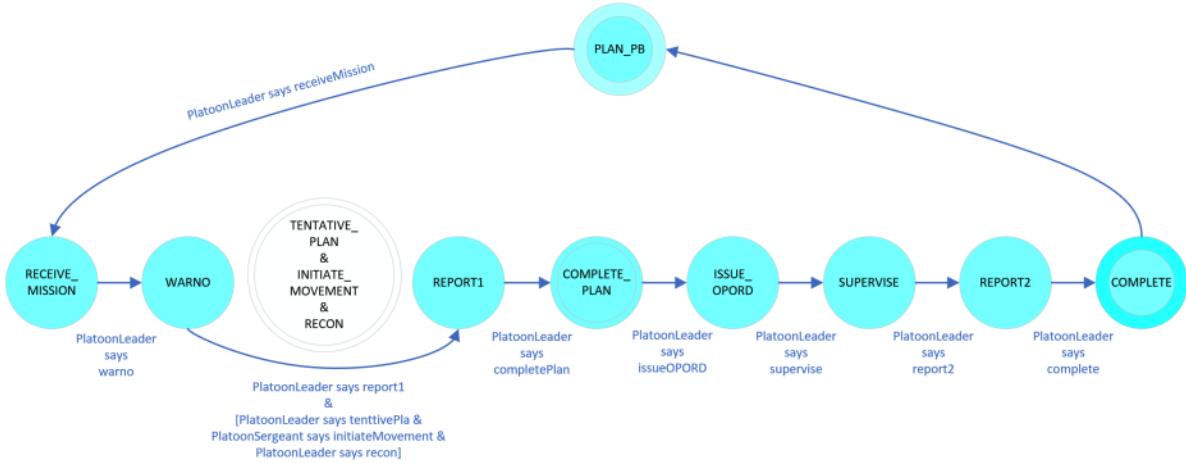


Figure 5.5: Horizontal slice: PlanPB diagram.

ssmPlanPB has two principals: PlatoonLeader and PlatoonSergeant. Only the PlatoonLeader is authorized to make transitions among states.

All transitions are sequential. However, the original module contains three non-sequential states. These are represented in the diagram as the white circle. These states are TENTATIVE_PLAN, INITIATE_MOVEMENT, and RECON. To transition from WARNO to REPORT1 requires that all three of these states be completed, but not in any specific order⁶.

⁶The complexities of this SSM lead to a revision of the original parameterized SSM discussed in chapter 6 section 6.3.

To solve this problem, the three states are not represented as states. The completion of these "tasks" is indicated by the following three statements: *PlatoonLeader says tentativePlan*, *PlatoonSergeant says initiateMovement*, and *PlatoonLeader says recon*.

Thus, the transition from WARNO to REPORT1 now requires four statements:

PlatoonLeader says tentativePlan, *PlatoonSergeant says initiateMovement*,

PlatoonLeader says recon, **AND** *PlatoonLeader says report1*. The latter-most statement is the actual request.

To be certain that the transition from WARNO to REPORT1 occurs if and only if these three statements are made, the security policy has an additional clause: *tentativePlan andf initiateMovement andf recon impf PlatoonLeader controls report1*. In the HOL a function extracts *tentativePlan* from *PlatoonLeader says tentativePlan*, and so on for all three statements. The security policy uses this function to verify that all four statements are present in the request to transition.

Note that these three statements are made without authentication and authorization. It wouldn't be too much effort to add this. But, these communications can also be thought of as the Platoon Leader receiving these statements directly from the sources. In this case, the Platoon Leader should recognize himself and his Platoon Sergeant. The actual implementation for a real world application would need to consider which is most appropriate for the situation.

5.5.6.2 ssmMoveToORP

The top level MOVE_TO_ORP state is expanded into the ssmMoveToORP SSM and shown in figure 5.6.

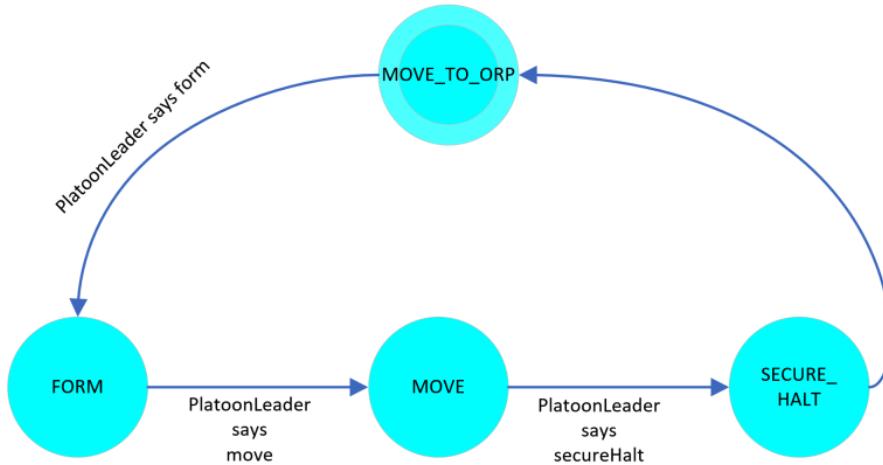


Figure 5.6: Horizontal slice: MoveToORP diagram.

ssmMoveToORP is straight forward. There is only one principal authorized to make transitions. This is the Platoon Leader. Everything follows sequentially.

5.5.6.3 ssmConductORP

The top level CONDUCT_ORP state is expanded into the ssmConductORP SSM and shown in figure 5.7.

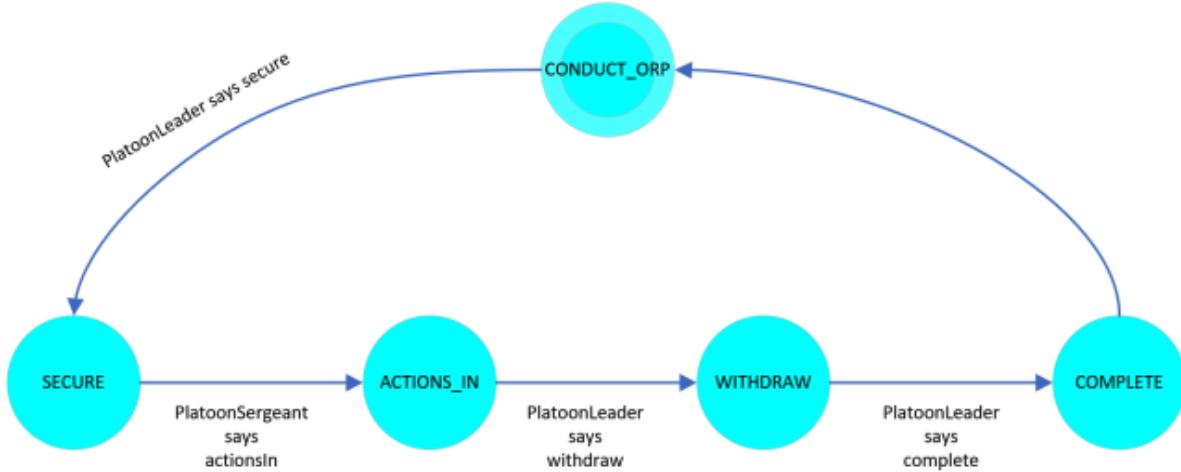


Figure 5.7: Horizontal slice: ConductORP diagram.

ssmConductORP is straight forward. But, there are two principals. The PlatoonLeader is authorized on all transitions except for the transition from SECURE to ACTIONS_IN. The PlatoonSergeant is responsible for this transition. Everything follows sequentially.

5.5.6.4 ssmMoveToPB

The top level MOVE_TO_PB state is expanded into the ssmMoveToPB SSM and shown in figure 5.8.

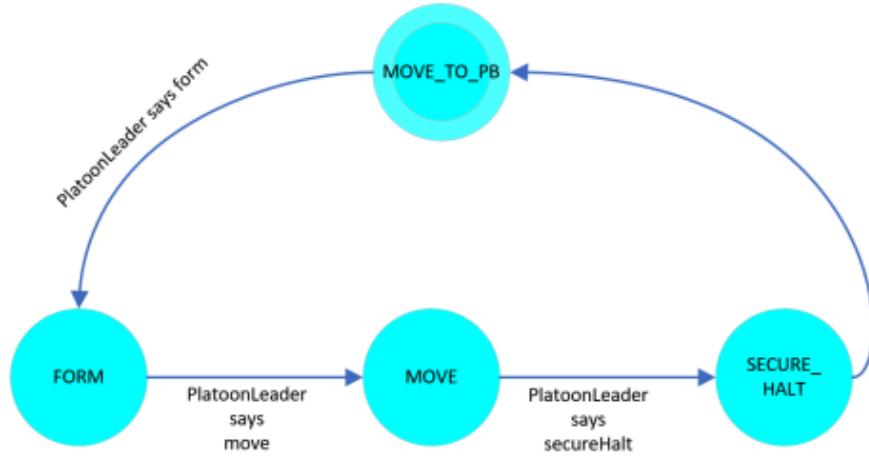


Figure 5.8: Horizontal slice: MoveToPB diagram.

ssmMoveToPB is straight forward. There is only one principal authorized to make transitions. This is the Platoon Leader. Everything follows sequentially.

5.5.6.5 ssmConductPB

The top level CONDUCT_PB state is expanded into the ssmConductPB SSM and shown in figure 5.9.

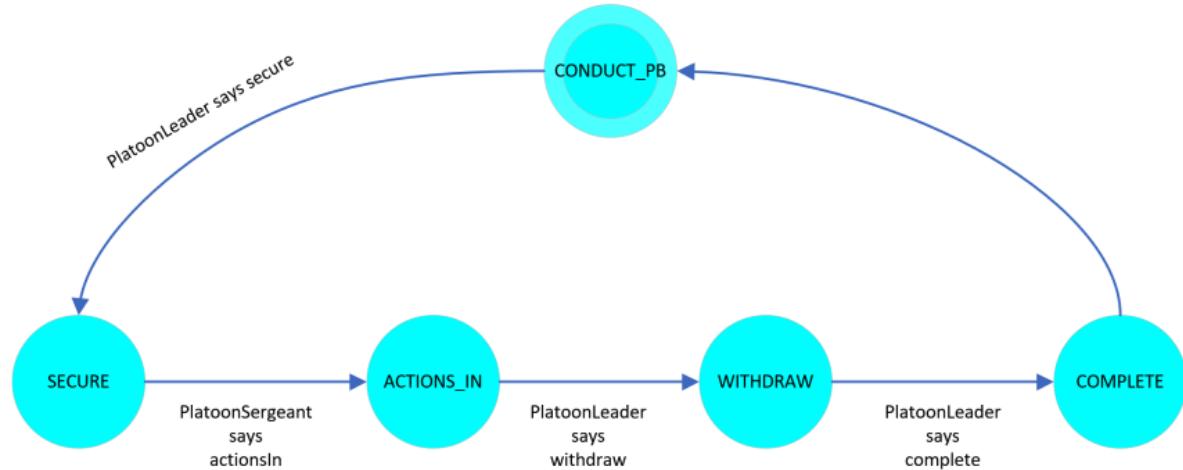


Figure 5.9: Horizontal slice: ConductPB diagram.

ssmConductPB is straight forward. But, there are two principals. The PlatoonLeader is authorized on all transitions except for the transition from SECURE to ACTIONS_IN. The PlatoonSergeant is responsible for this transition. Everything follows sequentially.

5.5.7 Vertical Slice

The vertical slice is an expansion of one state at each level of the hierarchy. It is the middle section in the overall, squished diagram in figure 5.2. This is the only section of the patrol base operations that are modeled through to the lowest level of abstraction.

The vertical slice starts at the top level state MOVE_TO_ORP. This state is expanded into the ssmMoveToORP SSM described in section 5.5.6.2. In this sub level, the state SECURE_HALTI is expanded into the ssmSecureHalt SSM described in the next section (5.5.7.1). In this sub-sub-level SSM, the state ORP_RECON is expanded into the ssmORPRecon SSM described below in section 5.5.7.2. In this SSM, the state MOVE_TO_ORP is expanded into the ssmMoveToORP4L SSM described below in section 5.5.7.3. In this SSM, the state FORM_RT is expanded into the ssmFormRT SSM described below in section 5.5.7.4.

Note that the overall diagram (the squashed diagram in figure 5.2) does not annotate transitions beyond the sub-sub-level. The principals assigned to transitions in the following modules are a best guess⁷.

⁷Describing the patrol base operations requires a subject matter expert. There was no indication at the time that he was developing the model that these lower-level transitions would be implemented in HOL.

5.5.7.1 ssmSecureHalt

The sub-sub-level SECURE_HALT state is expanded into the ssmSecureHalt SSM and shown in figure 5.10.

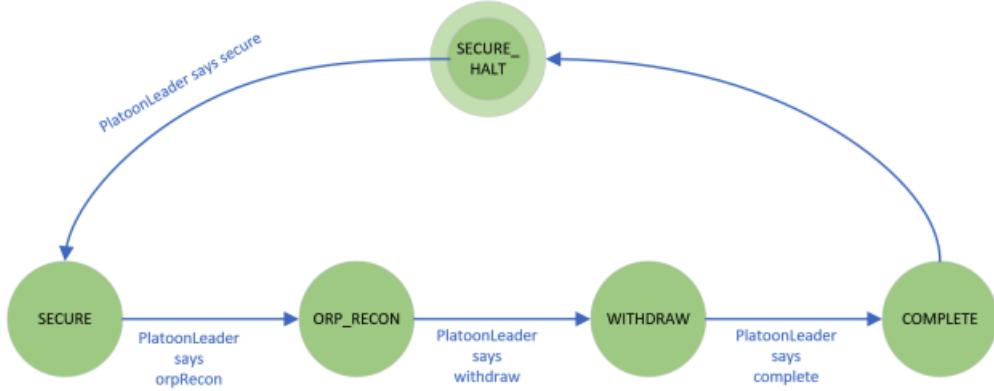


Figure 5.10: Vertical slice: SecureHalt diagram.

ssmSecureHalt is straight forward. There is only one principal authorized to make transitions. This is the Platoon Leader. Everything follows sequentially.

5.5.7.2 ssmORPRecon

The sub-sub-sub-level ORP_RECON state is expanded into the ssmORPRecon SSM and shown in figure 5.11.

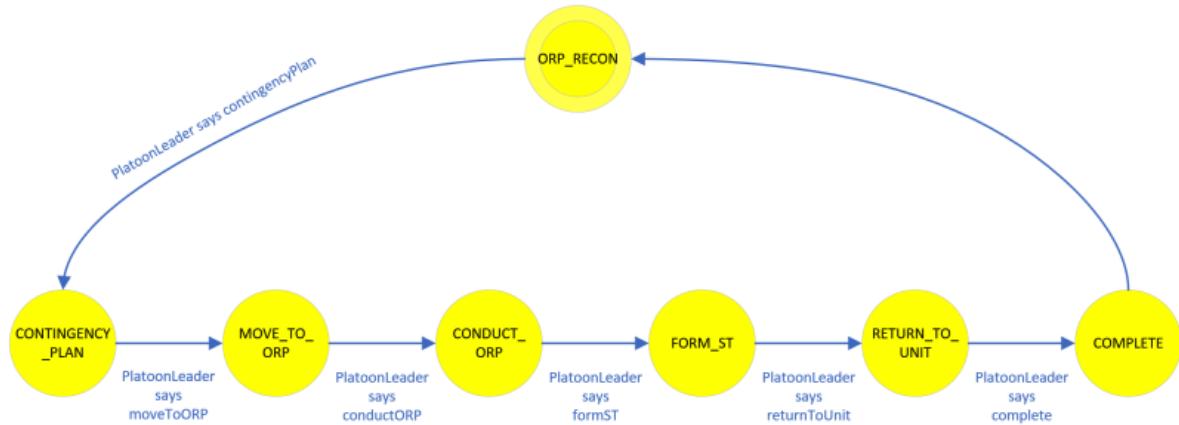


Figure 5.11: Vertical slice: ORPRecon diagram.

ssmOPRRecon is straight forward. There is only one principal authorized to make transitions. This is the Platoon Leader. Everything follows sequentially.

5.5.7.3 ssmMoveToORP4L

The 4th level MOVE_TO_ORP state is expanded into the ssmMoveToORP4L SSM and shown in figure 5.12. Recall that there is no module at the 5th level.

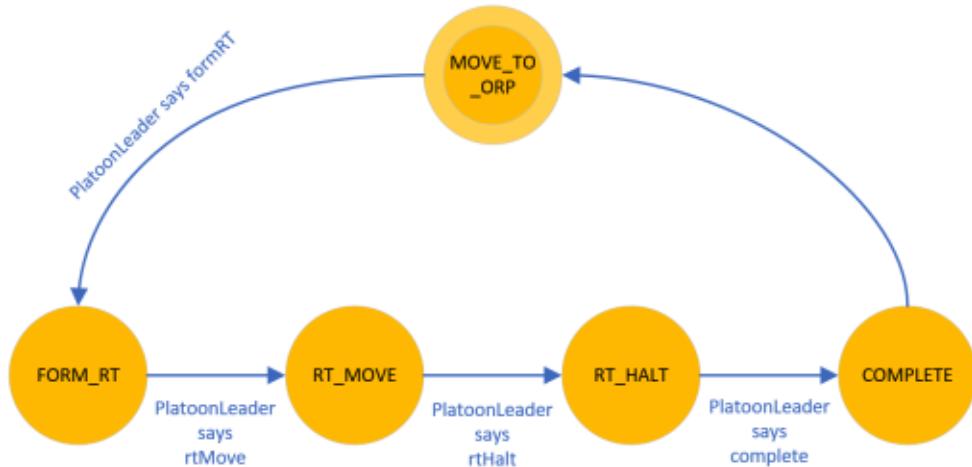


Figure 5.12: Vertical slice: MoveToORP4L diagram.

ssmMoveToORP4L is straight forward. There is only one principal authorized to make transitions. This is the Platoon Leader. Everything follows sequentially.

5.5.7.4 ssmFormRT

The 6th level FORM_RT state is expanded into the ssmFormRT SSM and shown in figure 5.12. (Recall that there is no module at the 5th level for this slice. Also, the 8th level is not modeled.)

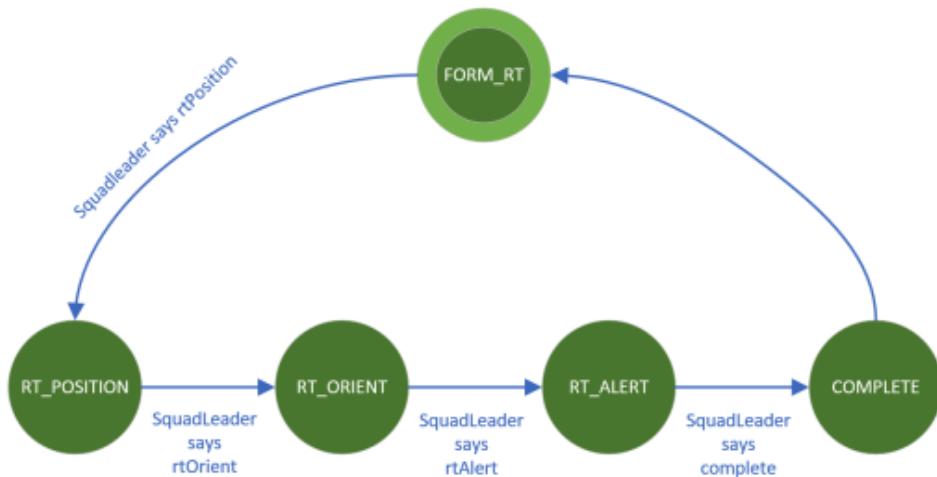


Figure 5.13: Vertical slice: FormRT diagram.

ssmFormRT is straight forward. There is only one principal authorized to make transitions. This is the Squad Leader. Everything follows sequentially.

Chapter 6

Secure State Machine Model

6.1 State Machines

State machines are models of systems. They use *states* and *transitions* among states to describe the system's behavior. To use the state machine model for a system the system must be describable as states which can change based on some input.

There are several models of state machines. A discussion of these models is beyond the scope of this master thesis. The state machine model described in this master thesis changes states and outputs based on input. This means that some input will cause the state machine to change states and produce an output.

6.1.1 States

States of a state machine can be nearly anything. For example, the state for a base could be described by the number of people on base. If a soldier is permitted to enter the base, then the state of the base increases from n people to $n + 1$ people. Or, the

state of a soldier requesting access to a base could be "not granted" or "granted."

In this master thesis, states are phases of the patrol base operations. For example, figure 6.1 shows the top level diagram depicting 6 abstract phases of the patrol base operations. These abstract phases are the states of the top level state machine.

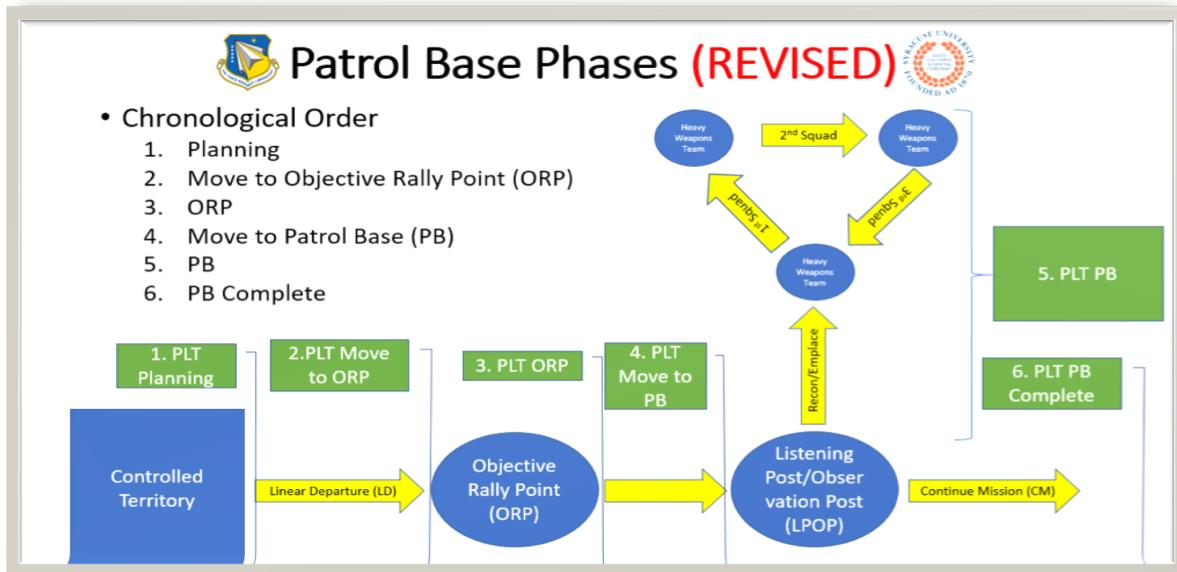


Figure 6.1: A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel as part of the research involved in this master thesis.

6.1.2 Transition Commands

Transition commands are inputs to the state machine. These inputs determine the next-state (and next-output) of the state machine. To continue with the examples above, input for the transition from n people to $n + 1$ people could be "access granted." This same input could be used to change a soldier's state from "not granted" to "granted."

In this master thesis, commands indicate the next state. For example, in figure 6.1, transition from the Move to Objective Rally Point (ORP) state to the ORP state is indicated by the command "conductORP."

6.1.3 Next-state Function

The next-state function describes the next state of the state machine given the current state and input. The function call is denoted

$$NS \ CURRENT_STATE \ input$$

For example, the next-state function to change from the n to $n + 1$ state would look like this:

$$NS \ N \ accessGranted = N + 1$$

. Similarly, for the soldier:

$$NS \ NOT_GRANTED \ accessGranted = GRANTED$$

In this master thesis, the next-state function contains the line

$$NS \ MOVE_TO_ORP \ conductORP = CONDUCT_ORP$$

6.1.4 Next-output Function

The next-output function describes the next output of the state machine given the current state and input. This function call is denoted

$$NOut \ CURRENT_STATE \ input$$

For example, the next-output function to change from the n to $n + 1$ state may look like this:

$$NOut\ N\ accessGranted = BasePopulationIncreasedByOne$$

. Similarly, for the soldier:

$$NOut\ NOT_GRANTED\ accessGranted = NowOnBase$$

In this master thesis, the next-output function contains the line

$$NS\ MOVE_TO_ORP\ conductORP = ConductORP$$

6.1.5 Configuration

A configuration describes the state machine using input and output streams [4]. A configuration has three components: (1) current state, (2) a list of inputs (input stream), and (3) a list of outputs (output stream). The information in the configuration is sufficient to instruct the state machine's next behavior.

6.1.6 TR Relations

Transition relations define the behavior of the state machine based on its input, configuration, and the next configuration. TR (transition relation) takes an input, an initial configuration, and a final configuration. This should be thought of as a proposition¹? For example, given the input and the initial configuration does the final configuration follow? If so, then the proposition is true. Otherwise, it is false. A TR

¹Propositions are necessary for proving properties of the state machine. In the case of transitions, they can be proved to be either true or false.

definition using configurations is shown below.

TR input

CFG

```
input::inputList  
CURRENT_STATE  
outList
```

CFG

```
inputList  
(NS input CURRENT_STATE)  
(NOut input CURRENT_STATE )::output
```

CFG denotes that what follows are the configuration components. The first configuration is before transition and the second configuration is after. The second configuration has the next-state, the next input (in the input stream) and the next-output as components.

The input stream n the first configuration is a list. The double colons (::) separate the first element of the list (head) from the remainder of the list. Using an input stream, the next configuration can be defined using the remainder of the input stream (which could be the empty list) `inputList` as its input.

The state in the first configuration is just the current state. The next state in the final configuration is the result of the next-state function *NS input CURRENT_STATE*.

The output in the first configuration is the `outputList`. It's head should be the output that corresponds to the current state. The output in the final configuration is the result of the next-output function cons'd onto the front of the `outputList`.

TR relations with configurations form the basis for the HOL representation of state machines, which is a precursor to the secure state machines implemented in this master

thesis.

6.2 Secure State Machines

Secure state machines add a level of security to the state machine model. In particular, the secure state machine implements access control by way of complete mediation.

6.2.1 State Machine Versus Secure State Machine

State machines define states, inputs, outputs, next-state functions, and next-output functions. Through these means, the state machine defines its behavior. Secure state machines add the additional concept of complete mediation to the state machine model by including checks on authentication and authorization. These checks are performed by a monitor. A diagram showing the relationship between state machine and secure state machine with a monitor is shown in figure 6.2.

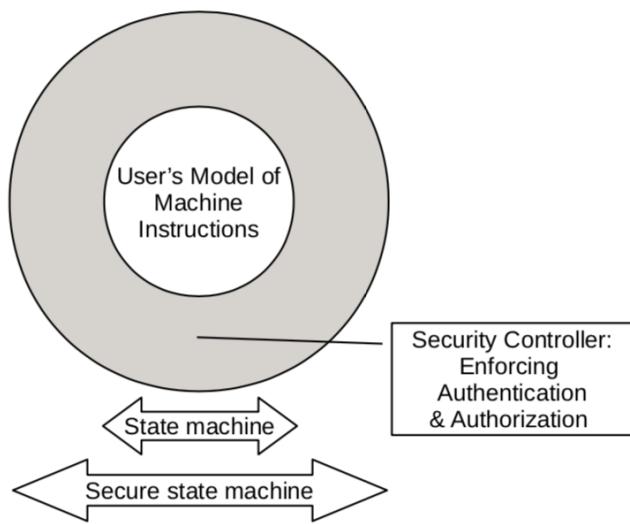


Figure 6.2: State machine versus secure state machine with a monitor. Image taken from Certified Security by Design Using Higher Order Logic [4].

For this master thesis, the "Machine Instructions" represent the state machine model of the patrol base operations.

6.2.2 Monitors

It is the duty of the monitor to control the behavior of the SSM with regards to complete mediation. The monitor is essentially a guard that checks for the proper authentication and authorization for all SSM transition requests. In a real world analogy, the monitor is the sentry at the gate who checks IDs and determines who is granted access to the base. In the SSM, the monitor controls access to state machine transitions in the same manner.

6.2.3 Transition Types

The monitor assigns a transition type to each command (or SSM transition request). This assignment is based on the security policy and the rules for authentication. There are three transition types: execute (*exec*), trap, (*trap*), and discard (*discard*)².

6.2.3.1 *exec*

The *exec* transition type indicates that a command should be executed. For example, if the Platoon Leader issues the command (request) *crossLD*, then the monitor must first check the authentication and authorization for that request. If the monitor authenticates the Platoon Leader and authorizes her on that request, then the monitor can justify executing that request. In the SSM model, this means that the transition from the PLAN_PB state to the MOVE_TO_ORP state (indicated by the command *crossLD*) should be allowed.

²The names are derived from their use in virtual machines. Commands in virtual machines are either executed, trapped, or discarded. Each has a different behavior in the machine.

6.2.3.2 *trap*

The *trap* transition type indicates that a command should NOT be executed. The *trap* transition type indicates that the principal is authenticated, but NOT authorized on that command. For example, if the Platoon Leader issues the command *initiateMovement*, then the monitor must first check the authentication and authorization of that request. In this case, the monitor authenticates the Platoon Leader. But, the monitor does not authorize the Platoon Leader on the command *initiateMovement* because only the Platoon Sergeant is authorized on this command. In the SSM model, this means that the transition from the WARNO state to the REPORT1 state (partially indicated by *initiateMovement*) should NOT be allowed. It should be trapped.

6.2.3.3 *discard*

Like the *trap* transition type, the *discard* transition type indicates that the command should NOT be executed. The *discard* transition type indicates that the principal is neither authenticated nor authorized. It may also indicate that the command is not of the correct form. For example, if SomeGuy issues *anyCommand*, the monitor must first check the authentication and authorization for that request. If the monitor does not authenticate SomeGuy, then SomeGuy's command is discarded.

It is useful to differentiate between the *trap* and *discard* transition types. This allows for the behavior of the SSM to handle authenticated but unauthorized and un-authenticated and unauthorized commands differently. For example, a sentry may choose to deny access to someone who is authenticated but unauthorized. On the other hand, a sentry may choose to detain an unauthenticated person.

6.2.4 Commands

Commands in the SSM are handled differently than in the state machine. Principals issue commands (make requests). The monitor inspects the command (request) for proper authentication and authorization and assigns a transition type to the command (request). This combination of transition type and command (request) is then passed to the next-state and next-output functions. These functions define how the SSM responds to each transition type and command (request) pair. This differs from the state machine in that the state machine only defines next-state and next-output functions for commands. The state machines is not concerned with authentication and authorization (i.e., access-control).

For example, the SSM for the top level is shown in figure 7.1 (this is the same as figure 5.4 in section 5.5.5).

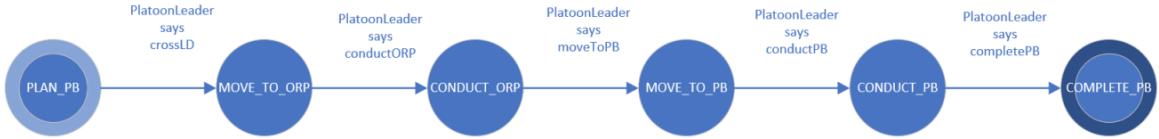


Figure 6.3: Top level diagram.

In the state machine, the transition from PLAN_PB to MOVE_TO_ORP only requires the command *crossLD*. But, in the secure state machine, transitions require some form of access-control. Transitions are indicated by principals making requests of the form *SomePrincipal says someCommand*. In figure 7.1, the Platoon Leader makes a request (issues a command) to transition from the PLAN_PB state to the MOVE_TO_ORP by stating

PlatoonLeader says crossLD.

The monitor then checks the authentication and authorization of the principal and returns the command with a transition type. For example, if the Platoon Leader is both authenticated and authorized on the command *crossLD*, then the monitor returns the transition type and command pair

exec crossLD

This is passed to the next-state and next-output functions. The next-state function is then justified in executing the transition from the PLAN_PB state to the MOVE_TO_ORP state.

6.2.5 Authentication

Authentication in the SSM refers to verification of identity. Authentication is a very broad topic and details are beyond the scope of this master thesis.

In this master thesis, authentication is dealt with by simple visual confirmation of a principal's identity³. This was justified by assuming that most soldiers in the platoon should recognize their leaders.

Authentication is verified by the monitor on each request⁴. This means that each request must be of the form *SomePrincipal says someCommand*, even if the principal was previously authenticated.

³Although, other methods are discussed in the chapter 8, particularly with regards to accountability systems.

⁴In the implementation of the SSM, this may or may not be the case. For example, access to a secured facility may require proper authentication upon entering the facility. This may be indicated with a badge worn by the individual while in the facility. Within the facility, only badges are checked. But, the process of verifying identity only occurs at the entry.

6.2.6 Authorization

Authentication is a way of controlling who has access to what by means of a security policy. The security policy implements the principal of complete mediation (and other security policies).

Authentication in the SSM is typically one or more functions that define which principals have control over which transitions. A simple authorization in a SSM is *SomePrincipal controls someCommand*. Using the access-control logic (ACL) described in Chapter 4 and a request of the form *SomePrincipal says someCommand*, *someCommand* is justified. Of course, the monitor must first check the authentication of SomePrincipal. Once that is verified, the monitor returns the transition type and command pair *exec someCommand*.

Authorization is often more complicated. For example, there are two security policies for the top level SSM shown in figure 7.1. The security policy for the Omni principal is

Omni controls omniCommand.

This means that the Omni principal is authorized on any *omniCommand*. For example, *ssmPlanPBComplete* is one *omniCommand* in the top level SSM in figure 7.1. The security policy for the Platoon Leader depends on the current state because the Platoon Leader is only authorized to transition to the next state if the current state is complete. (Omni indicates when the current state is complete.) Thus, the security policy for the Platoon Leader on the transition from the *PLAN_PB* state to the *MOVE_TO_ORP* state in figure 7.1 has the form

ssmPlanPBComplete impf PlatoonLeader controls crossLE

where *impf* is the ACL operator for implication.

6.2.7 Next-state And Next-output Functions

The next-state and next-output functions define the behavior for the SSM. Whereas the behavior of the state machine is defined only for states and commands, the behavior in the SSM includes the behavior for each transition type in combination with each command. This means that for each command, the next-state and next-output functions define three separate behaviors: *exec someCommand*, *trap someCommand*, and *discard someCommand*.

6.2.8 Configurations

The configuration in the secure state machine adds three additional components, for a total of six: (1) an authentication function, (2) a state interpretation function (a state-dependent security policy), (3), a security context, (4) an input list (a command list), (5) a state, and (6) an output list. The next-state and next-output functions are not part of the configuration because they define the permanent structure of the SSM.

6.2.9 TR Relations

Transition relations in the secure state machine are similar to those for the state machine.

TR (trTpye input)

CFG

authenticationTest

stateInterpretation

securityContext

input::inputList

```

CURRENT_STATE
outList
CFG
authenticationTest
stateInterpretation
securityContext
inputList
(NS (tyType input) CURRENT_STATE)
(NOut (trType input) CURRENT_STATE )::output

```

In addition to the three additional components, the transition type (trType) is added to the front of the input. Because the next-state and next-output functions are defined in terms of the transition type and command pair, the trType is also used in these functions.

The authenticationTest, stateInterpretation, and securityContext functions are the same in both the initial and final configurations.

6.2.10 Configuration Interpretation

The TR relation is a proposition that returns true or false based on whether the second configuration follows from the first. But, this is not wholly sufficient to prove complete mediation. Complete mediation requires that the configuration be interpreted in the context of authentication and authorization.

ConfigurationInterpretation

(M,Oi,Os)

CFG

AuthenticationTest inputOK

```

StateInterpretation
  SecurityContext
    input::inputList
    CURRENT_STATE
    outList
  ⇐⇒
  M,Oi,Os) satisfies (securityContext, input, and stateInterpretation)

```

In the broadest sense, this function says that the configuration interpretation is true if and only if the securityContext, input, and stateInterpretation satisfy the Kripke structure (M, O_i, O_s) .

This configuration interpretation function takes a Kripke structure and the initial configuration as parameters. The Kripke structure is necessary to prove properties of soundness. But, Kripke structures are not used in this master thesis. Nevertheless, soundness is true for ALL Kripke structures. Thus it is sufficient to pass any undefined Kripke structure⁵.

In practice, the last line of the configurationInterpretation function is reduced for *exec* and *trap* transition types.

```

(M,Oi,Os) satisfies someCommand
(M,Oi,Os) satisfies noCommand

```

where *someCommand* means that execution of the input command is justified and *noCommand* means that the command is trapped. Details depend on the implication and use the Option datatype, described in the section 6.3.3 below. The *discard* transition type does not require configurationInterpretation because the command does not pass the authentication phase.

⁵Remember that the Kripke structure in the HOL implementation of the CSBD has a Kripke structure is of the form (M, O_i, O_s) .

6.3 Secure State Machines in HOL

The secure state machines are implemented in HOL. This section describes the components of the parametrizable secure state machine (denoted "ssm" with lower-case letters).

6.3.1 Parameterizable Secure State Machine

ssm is implemented as a parametrizable secure state machine. Parametrization allows for re-use of common definitions and theorems in the SSM model. The parametrizable components are

- Next-state function
- Next-output function
- Input & Input stream
- Output & Output stream
- States
- Principals
- Security context
- State interpretation-based security context
- Authentication test function

To use the ssm, the rule ISPECL HOL is used to specialize the parametrizable theorems for a specific secure state machine.

6.3.2 Input Stream

The input to the secure state machines is in the form of a list of inputs (an input stream). Elements in the list are of the form $P \text{ says } prop \text{ (SOME cmd)}$. It is necessary to extract particular components from the list and list elements. Several functions are defined to do this. These are essentially helper functions.

extractCommand `extractCommand` takes one input of the form $P \text{ says } prop \text{ (SOME cmd)}$ and extracts the cmd part.

[`extractCommand_def`]

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

commandList `commandList` takes an input list consisting of list elements of the form $P \text{ says } prop \text{ (SOME cmd)}$. It returns a list of all the cmd elements.

```
⊢ ∀ x.
```

```
commandList x = MAP extractCommand x
```

extractPropCommand `extractPropCommand` takes one input of the form $P \text{ says } prop \text{ (SOME cmd)}$ and extracts the $prop \text{ (SOME cmd)}$ part.

[`extractPropCommand_def`]

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

propCommand `propCommand` takes an input list consisting of list elements of the form $P \text{ says } prop \text{ (SOME cmd)}$. It returns a list of all the $prop \text{ (SOME cmd)}$ elements.

$\vdash \forall x.$

```
propCommandList x = MAP extractPropCommand x
```

extractInput `extractInput` takes one input of the form P says $prop\ x$ and extracts the x part. Note that x can have two forms: *NONE* or *SOME cmd*.

[`extractInput_def`]

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

inputList `inputList` takes an input list consisting of list elements of the form P says $prop\ x$. It returns a list of all the x elements.

$\vdash \forall xs.$

```
inputList xs = MAP extractInput xs
```

6.3.3 Commands

Option Type The option type allows for the return of "no value." In functional programming, this is an important concept because functions must always return a value. Consider the following example. An association list consists of tuples of the form $\{('a',1), ('b',2), \dots ('a',26)\}$. A search for 1 returns the integer value 'a'. But, what if the target of the search is the number 27? The search must return something. But, there is nothing to return. This is where the option type comes in handy. Instead of returning 'a' for 1 and 'b' for 2 and so on, the search returns SOME 'a' and SOME 'b' and so on. If the search does not find a match, it returns NONE.

The definition for the option datatype is

```
option = NONE | SOME 'a
```

In the datatype definition above, 'a is replaced with some other datatype.

A Closer Look at Commands To see what this looks like in HOL, it is necessary to define some other datatypes. The OMNILevel folder in OMNITypesScript.sml contains definitions that are used in all patrol base operations SSMs. One definition is the *command* datatype definition.

```
command = ESCC escCommand | SLC 'slCommand
```

The *command* datatype consists of two additional datatypes: ESCC escCommand and SLC 'slCommand. Note that the first part of each of these are the datatype constructors⁶: ESCC and SLC. The second part is the name of the datatype variable⁷ or datatype.

The first datatype refers to the escape commands. They are defined as escCommand in the same file as *command*.

```
escCommand = returnToBase  
| changeMission  
| resupply | reactToContact
```

This datatype definition defines three commands (or datatype values) which represent escape conditions in the patrol base operations.

The second datatype variable 'slCommand refers to the state-level commands. These are defined further in each SSM.

⁶see the background section 2.5

⁷Both of these are datatype variables because they define other datatypes.

Notice that there is a tick mark (apostrophe) before '`slCommand`' and not before '`escCommand`'. In general, the tick mark in HOL represents an undefined datatype. In this case, '`slCommand`' is not yet defined (because it is defined elsewhere), whereas the definition for '`escCommand`' is defined in the same file and above the definition for *command*.

An example of a definition for '`slCommand`' can be found in the top level SSM. It is defined in the folder `topLevel` and in the file `PBIntegratedTypeScript.sml` file.

```
slCommand = PL plCommand | OMNI omniCommand
```

This is defined similarly to *command*. There are two datatypes that make-up the datatype *slCommand*. None of these have tick marks, which means both of these are defined. In particular, they are both defined in the same file as *slCommand*.

plCommand refers to the Platoon Leader commands. These are commands that the Platoon Leader is authorized to make.

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

omniCommand refers to commands that the OMNI level principal⁸ is authorized to make.

```
omniCommand = ssmPlanPBComplete
```

⁸See section 5.5.3 for a discussion of the OMNI level principal.

```

| ssmMoveToORPComplete
| ssmConductORPComplete
| ssmMoveToPBComplete
| ssmConductPBComplete
| invalidOmniCommand

```

Option Type with Commands With these definitions, it is possible to see how the options types are used with commands (datatypes). What follows is a list of examples using the option types and commands (datatypes) described above. The type signatures are also included because it will help the reader recognize them in the HOL code.

SOME (SLc (ESCc returnToBase))

The type for this is (escCommand command)Option.

SOME (SLc (PL moveToORP))

The type for this is ((plCommand slCommand) command)Option.

SOME (SLc (OMNI ssmMoveToORPComplete))

The type for this is ((omniCommand slCommand) command)Option.

Note that the constructors are necessary for each command. Also, note that in the HOL code for the patrol base operations, the reader will typically see (slCommand command)Option. This is because the definitions require a type *slCommand*, which includes *plCommand* and *omniCommand*.

6.3.3.1 Transition Types

Transition datatypes indicate how a command is handled by the monitor. The three transition datatypes are described below.

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

The 'cmdlist refers to a list of commands of the form discussed in the section above. For example, to execute the transition from the PLAN_PB state to the MOVE_TO_ORP state, the monitor must return the transition type and command pair with the later of the pair in the form of a list

$$\text{exec } [\text{SOME } (\text{SLc } (\text{PL crossLD}))]$$

where *SOME* (*SLc* (*PL crossLD*)) is the single item in the cmdlist. The transition type with the command list is then passed to the next-state and next-output functions.

6.3.4 Authentication

Authorization is context dependent. But, a parametrizable authentication is defined in the parametrizable SSM.

$$\begin{aligned} &\vdash \forall \text{elementTest } x. \\ &\quad \text{authenticationTest elementTest } x \iff \\ &\quad \text{FOLDR } (\lambda p\ q. p \wedge q) \top (\text{MAP elementTest } x) \end{aligned}$$

This function takes an elementTest function and an input list as parameters. The elementTest function is named inputOK and it is defined separately for each SSM. elementTest takes a single input of the form *SomePrincipal says someCommand*. It returns TT (the ACL representation of True) if the input is authenticated and FF otherwise.

The authenticationTest function FOLDERS the elementTest function over the input list x

with the conjunction function and True as the accumulator⁹ Thus, if all the input elements in the input list x pass the elementTest, the authenticationTest function returns true, otherwise it returns false.

6.3.5 Authorization

Authorization is the security context and it is SSM-dependent. This means that each SSM defines its own security context. The parametrizable SSM allows for two ways to define the security context and pass them as parameters. The first is the state interpretation function. This function takes a state and an input list as parameters. It defines the security context based on state. All state-dependent behavior may also be defined in this function.

The second function is the security context function. It takes only the input list as a parameter. Its definition applies to all states.

6.3.6 Next-state And Next-output Functions

The next-state and next-output functions are parameters to the parametrizable SSM. They are defined separately in each SSM. In the parametrizable SSM they are denoted by NS and Out, respectively.

⁹This means that the elementTest function is applied to each element in the input list x , resulting in a TT or FF value for each element in the input list x . Then, in essence, the value of the conjunction of all these values is returned.

6.3.7 Configurations

Configurations in the SSMs have six components. Each SSM must define all six components to use the parametrizable ssm. These components are:

1. authentication test function.

The type signature is

$$((\text{'command option}, \text{'principal}, \text{'d}, \text{'e}) \text{ Form} \rightarrow \text{bool})$$

2. state interpretation function.

The type signature is

$$(\text{'state} \rightarrow ((\text{'command option}, \text{'principal}, \text{'d}, \text{'e}) \text{ Form list} \rightarrow (\text{'command option}, \text{'principal}, \text{'d}, \text{'e}) \text{ Form list}))$$

3. security context function.

The type signature is

$$((\text{'command option}, \text{'principal}, \text{'d}, \text{'e}) \text{ Form list} \rightarrow (\text{'command option}, \text{'principal}, \text{'d}, \text{'e}) \text{ Form list})$$

4. input list stream.

The type signature is

$$((\text{'command option}, \text{'principal}, \text{'d}, \text{'e}) \text{ Form list list})$$

5. state.

The type signature is

$$\text{'state}$$

6. output stream.

The type signature is

$$(\text{'output list})$$

Note that the authentication test function is defined in the parametrizable ssm as authenticationTest. This function takes one input (named elementTest in this ssm and inputOK in the SSMs).

These components comprise the six components of the configuration datatype (with CFG as the datatype constructor)

```
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)
```

6.3.8 Configuration Interpretation

The monitor must interpret the configuration. The CFGInterpret function take as input a Kripke structure and a configuration. It returns a conjunction of three things: a satList of the security context, a satList of the input stream, and a satList of the stateInterpretation function.

satList is a list of elements that satisfy the property of soundness as discussed in chapter 4 section 4.2.5.4¹⁰. satList is defined in satListTheory.

¹⁰Note that in the ACL implementation of the "satisfies" and "soundness" properties, "satisfies" serves as "soundness" when it is generalized for all Kripke structures.

$$\begin{aligned}
&\vdash \forall M \ Oi \ Os \ formList. \\
&(M, Oi, Os) \ satList \ formList \iff \\
&\text{FOLDR } (\lambda x \ y. \ x \wedge y) \ T \ (\text{MAP } (\lambda f. \ (M, Oi, Os) \ sat f) \ formList)
\end{aligned}$$

satList MAPs the *sat* operator onto each element in the *formList*. It then FOLDRs the *formList* elements with the conjunction function and accumulator True. This means that *satList* applied to *formList* returns true if each element in the list satisfies the *sat* property.

Other properties of *satList* can be found in appendix A.

With *satList* defined, the meaning of *CFGInterpret* should follow.

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

6.3.9 TR Rules

The transition rules in HOL are defined in figure 6.4.

There are three rules, one for each transition type. Above each line are the hypotheses and below are the conclusions. The symbol $Config \xrightarrow{\text{exec } (inputList \ x)} Config_e$ represents the TR relation for the input *exec inputList x*. Similarly, the symbols $Config \xrightarrow{\text{trap } (inputList \ x)} Config_t$ and $Config \xrightarrow{\text{discard } (inputList \ x)} Config_d$ represent the TR relations for *trap inputList x* and *discard inputList x*, respectively.

The *Execute* rule takes the *authenticationTest* function (with two parameters) and the

SSM behavior is defined inductively by three rules.

$$\text{Execute} \quad \frac{(\text{authenticationTest } \text{elementTest } x) \quad (\text{CFGInterpret } (M, O_i, O_s) \text{ Config})}{\text{Config} \xrightarrow{\text{exec (inputList } x)} \text{Config}_e}$$

$$\text{Trap} \quad \frac{(\text{authenticationTest } \text{elementTest } x) \quad (\text{CFGInterpret } (M, O_i, O_s) \text{ Config})}{\text{Config} \xrightarrow{\text{trap (inputList } x)} \text{Config}_t}$$

$$\text{Discard} \quad \frac{\neg(\text{authenticationTest } \text{elementTest } x)}{\text{Config} \xrightarrow{\text{discard (inputList } x)} \text{Config}_d}$$

where,

$$\text{Config} = \text{CFG elementTest stateInterp context } (x :: \text{ins}) s \text{ outs}$$

$$\text{Config}_e = \text{CFG elementTest stateInterp context ins}$$

$$(NS s (\text{exec (inputList } x))) (Out s (\text{exec (inputList } x)) :: \text{outs})$$

$$\text{Config}_t = \text{CFG elementTest stateInterp context ins}$$

$$(NS s (\text{trap (inputList } x))) (Out s (\text{trap (inputList } x)) :: \text{outs})$$

$$\text{Config}_d = \text{CFG elementTest stateInterp context ins}$$

$$(NS s (\text{discard (inputList } x))) (Out s (\text{discard (inputList } x)) :: \text{outs})$$

Figure 6.4: Transition rules in HOL. Image taken from Certified Security by Design Using Higher Order Logic [4]

CFGInterpret function (with two parameters). If these are true, then *Execute* concludes the TR relation $\text{Config} \xrightarrow{\text{exec (inputList } x)} \text{Config}_e$. *Trap* is similar to *Execute*.

The *Discard* rule takes only the authenticationTest function (with two parameters). If this is false, then the hypothesis is true. Thus, the $\text{Config} \xrightarrow{\text{discard (inputList } x)} \text{Config}_d$ follows.

The definitions for the *Configs* are defined below the rules.

These definitions are defined in HOL as rule0, rule1, and rule2

(TR_discard_cmd_rule). Notice that these are biconditionals. The hypotheses and conclusions are reversed in the definitions below.

rule0 [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)
```

rule1 [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)
```

rule2 [rule2... (same as) 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
```

Complete mediation and TR Relations It remains to prove that complete mediation justifies execution, trapping, or discarding of commands.

TR_exec_cmd_rule The following function demonstrates the property of complete mediation as a condition for execution a command.

[TR_exec_cmd_rule]

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

This is similar to rule0. It adds the following as a premise for concluding rule0.

$$\begin{aligned}
 & \vdash \forall \text{elementTest } \text{context } \text{stateInterp } x \text{ ins } s \text{ outs} . \\
 & (\forall M \text{ } Oi \text{ } Os . \\
 & \quad \text{CFGInterpret } (M, Oi, Os) \\
 & \quad (\text{CFG elementTest stateInterp context } (x::\text{ins}) \text{ } s \text{ outs})
 \end{aligned}$$

$\Rightarrow(M, Oi, Os) \text{ satList propCommandList } x)$

This part requires that the propCommandList applied to the inputList satisfies the Kripke structure.

TR_trap_cmd_rule TR_trap_cmd_rule is similar to TR_exec_cmd_rule.

[TR_trap_cmd_rule]

$$\vdash \forall \text{elementTest context stateInterp } x \text{ ins } s \text{ outs} .$$

$$(\forall M \text{ } Oi \text{ } Os .$$

$$\text{CFGInterpret } (M, Oi, Os)$$

$$(\text{CFG elementTest stateInterp context } (x :: \text{ins}) \text{ } s$$

$$\text{outs}) \Rightarrow$$

$$(M, Oi, Os) \text{ sat prop NONE} \Rightarrow$$

$$\forall NS \text{ Out } M \text{ } Oi \text{ } Os .$$

$$\text{TR } (M, Oi, Os) \text{ (trap (inputList } x))$$

$$(\text{CFG elementTest stateInterp context } (x :: \text{ins}) \text{ } s \text{ outs})$$

$$(\text{CFG elementTest stateInterp context } \text{ins}$$

$$(NS \text{ } s \text{ (trap (inputList } x)))$$

$$(\text{Out } s \text{ (trap (inputList } x)) :: \text{outs})) \iff$$

$$\text{authenticationTest elementTest } x \wedge$$

$$\text{CFGInterpret } (M, Oi, Os)$$

$$(\text{CFG elementTest stateInterp context } (x :: \text{ins}) \text{ } s \text{ outs}) \wedge$$

$$(M, Oi, Os) \text{ sat prop NONE}$$

The difference is that instead of requiring propCommandList to satisfy the Kripke structure, it must satisfy NONE.

$$\vdash \forall \text{elementTest context stateInterp } x \text{ ins } s \text{ outs} .$$

$$(\forall M \text{ } Oi \text{ } Os .$$

$$\text{CFGInterpret } (M, Oi, Os)$$

$(CFG \ elementTest \ stateInterp \ context \ (x::ins) \ s \ outs)$
 $\Rightarrow (M, Oi, Os) \ sat \ prop \ NONE$

TR _ discard _ cmd _ rule This is the same as rule2 because the *discard* transition type does not require configuration interpretation.

The next chapter demonstrates the parametrizable ssm applied to specific patrol base operations SSMs.

Chapter 7

Patrol Base Operations as Secure State Machines

The hierarchy of secure state machines (SSMs) consists of many SSMs. Ten of these SSM are verified in HOL. Many of these SSMs are similar. Thus, a sampling of representative SSMs is described in this section. All SSMs can be found in the appendices starting with appendix C. They are also contained in the MasterThesis/HOL/ folders.

7.1 ssmPB: A Typical Example from the Hierarchy

ssmPB is the top level SSM. It is an example of a typical one-principal SSM. A diagram of ssmPB is shown in figure 7.1.

ssmPB runs sequentially from the initial state PLAN_PB to the final state COMPLETE_PB. State transitions require notification from the Omni principal that the state is complete and a request from the Platoon Leader to change states. Thus, a

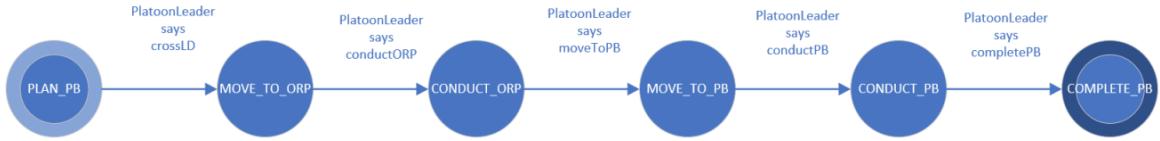


Figure 7.1: Top level diagram.

transition request has the form

Omni says stateComplete;

PlatoonLeader says moveToNextState

The security context reflects this two input requirement. It authorizes the Platoon Leader on *moveToTheNextState* if the current state is complete. A clause in the security context has the form

stateComplete impf

PlatoonLeader controls moveToNextState

This clause is state dependent because *stateComplete* is specific for the current state. Thus, there are five statements in the state-dependent security context.

There is also a state-independent security context. This has one clause authorizing the Omni principal on all omniCommands. It has the form

Omni controls stateComplete

Both the Omni and PlatoonLeader principals are authenticated on all slCommands. This includes omniCommands and plCommands. However, authorization does distinguish between omniCommand and plCommand. The authentication function has

two statements of the form

Omni says slCommand

PlatoonLeader says slCommand

The HOL implementation of the datatypes, functions, and theorems are included below.

7.1.1 Principals

The principal datatype is defined in OMNITypeScript.sml. It has a constructor and one datatype variable.

principal = SR 'stateRole

SR is the type constructor and 'stateRole is the type variable. Each SSM defines its own principal as a stateRole. In ssmPB, the stateRole datatype has two principals.

stateRole = PlatoonLeader | Omni

7.1.2 States

States are also defined in OMNITyprpeScript.sml.

state = ESCs escState | SLs 'slState

'slState is a state-dependent state which is further defined in each SSM. ssmPB defines six states.

slState = PLAN_PB

```
| MOVE_TO_ORP  
| CONDUCT_ORP  
| MOVE_TO_PB  
| CONDUCT_PB  
| COMPLETE_PB
```

7.1.3 Outputs

Outputs are defined in OMNITyrpeScript.sml.

output = ESCo escOutput | SLo 'slOutput

SLo 'slOutput is the state-dependent output. It is defined further in each SSM. ssmPB defines seven outputs.

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

The unAuthorized and unAuthenticated pertain to *trap* and *discard* transition types, respectively.

7.1.4 Commands

OMNITYrpeScript.sml defines datatypes for all the SSMs.

command = ESCc escCommand | SLC 'slCommand

SLC 'slCommand is further defined in each SSM. ssmPB defines two datatypes for the slCommand.

slCommand = PL plCommand | OMNI omniCommand

The omniCommand and plCommand are further defined in ssmPB.

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

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

7.1.5 Next-State Function

Each SSM defines its own next-state function.

[PBNS_def]

```

 $\vdash (\text{PBNS PLAN\_PB } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{crossLD} \text{ then MOVE\_TO\_ORP else PLAN\_PB}) \wedge$ 
 $\quad (\text{PBNS MOVE\_TO\_ORP } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{conductORP} \text{ then CONDUCT\_ORP}$ 
 $\quad \text{else MOVE\_TO\_ORP}) \wedge$ 
 $\quad (\text{PBNS CONDUCT\_ORP } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{moveToPB} \text{ then MOVE\_TO\_PB else CONDUCT\_ORP}) \wedge$ 
 $\quad (\text{PBNS MOVE\_TO\_PB } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{conductPB} \text{ then CONDUCT\_PB else MOVE\_TO\_PB}) \wedge$ 
 $\quad (\text{PBNS CONDUCT\_PB } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{completePB} \text{ then COMPLETE\_PB}$ 
 $\quad \text{else CONDUCT\_PB}) \wedge (\text{PBNS } s \text{ (trap } v_0) = s) \wedge$ 
 $\quad (\text{PBNS } s \text{ (discard } v_1) = s)$ 

```

The next-state function for ssmPB uses pattern matching and if-then-else statements.

The function *getPlCom* extracts the plCommand from the command list. It compares the extracted command to the command to transition. If they are equal, then the transition occurs. Otherwise, no transition occurs.

For the *trap* and *discard* transition types, no transition occurs regardless of the command. Therefore, the command is not checked.

PBNS takes a state and a transition (*exec*, *trap*, or *discard*) and command list (*x*) pair.

7.1.6 Next-Output Function

Each SSM defines its own next-state function.

[PBOut_def]

```
⊤ (PBOut PLAN_PB (exec x) =  
    if getPlCom x = crossID then MoveToORP else PlanPB) ∧  
    (PBOut MOVE_TO_ORP (exec x) =  
    if getPlCom x = conductORP then ConductORP else MoveToORP) ∧  
    (PBOut CONDUCT_ORP (exec x) =  
    if getPlCom x = moveToPB then MoveToORP else ConductORP) ∧  
    (PBOut MOVE_TO_PB (exec x) =  
    if getPlCom x = conductPB then ConductPB else MoveToPB) ∧  
    (PBOut CONDUCT_PB (exec x) =  
    if getPlCom x = completePB then CompletePB else ConductPB) ∧  
    (PBOut s (trap v0) = unAuthorized) ∧  
    (PBOut s (discard v1) = unAuthenticated)
```

The next-output function behaves similarly to the next-state function. But, instead of returning the next state, it returns the next function. The *trap* and *discard* transition types return *unAuthenticated* and *unAuthorized*, respectively.

7.1.7 Authentication

The *authenticationTest* function is defined in the parametrizable ssm. It takes a function as an input and that function takes an input list as an input. The first function is named *elementTest* in ssm. It is named *inputOK* in the SSMs.

In HOL, *inputOK* uses the wild card denoted by an underscore "_". The underscore

causes additional output from HOL. For this reason, the HOL definition is shown first. It is followed by the HOL code generated by the function.

HOL Definition for inputOK The HOL definition for inputOK uses pattern matching.

```
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)`
```

The first call to *inputOK* matches the input to $((\text{Name PlatoonLeader}) \text{ says } \text{prop} (\text{cmd} : ((\text{slCommand command}) \text{ option})))$. If it matches, the function returns T for true. The second call to *inputOK* matches the input to $((\text{Name Omni}) \text{ says } \text{prop} (\text{cmd} : ((\text{slCommand command}) \text{ option})))$. If it matches, it returns true. The last call to *inputOK* uses the wild card. This returns false for any other input.

The first two calls to *inputOK* authenticate the PlatoonLeader and Omni principals on any slCommand.

HOL Generated Output for inputOK [inputOK_def]

```

 $\vdash (\text{inputOK} (\text{Name PlatoonLeader} \text{ says } \text{prop} \text{ } \text{cmd}) \iff \text{T}) \wedge$ 
 $(\text{inputOK} (\text{Name Omni} \text{ says } \text{prop} \text{ } \text{cmd}) \iff \text{T}) \wedge$ 
 $(\text{inputOK} \text{ TT} \iff \text{F}) \wedge (\text{inputOK} \text{ FF} \iff \text{F}) \wedge$ 
 $(\text{inputOK} (\text{prop} \text{ } v) \iff \text{F}) \wedge (\text{inputOK} (\text{notf} \text{ } v_1) \iff \text{F}) \wedge$ 
 $(\text{inputOK} (\text{v}_2 \text{ andf} \text{ } v_3) \iff \text{F}) \wedge (\text{inputOK} (\text{v}_4 \text{ orf} \text{ } v_5) \iff \text{F}) \wedge$ 
 $(\text{inputOK} (\text{v}_6 \text{ impf} \text{ } v_7) \iff \text{F}) \wedge (\text{inputOK} (\text{v}_8 \text{ eqf} \text{ } v_9) \iff \text{F}) \wedge$ 
 $(\text{inputOK} (\text{v}_{10} \text{ says } \text{TT}) \iff \text{F}) \wedge (\text{inputOK} (\text{v}_{10} \text{ says } \text{FF}) \iff \text{F}) \wedge$ 
 $(\text{inputOK} (\text{v}_{133} \text{ meet } \text{v}_{134} \text{ says } \text{prop} \text{ } v_{66}) \iff \text{F}) \wedge$ 
```

```

(inputOK (v135 quoting v136 says prop v66)  $\iff$  F)  $\wedge$ 
\iff F)  $\wedge$  (\iff F)  $\wedge$ 
\iff F)  $\wedge$  (\iff F)

```

It is straight forward to prove that any command that is not issued by a principal is rejected. This should follow directly from the definition of *inputOK*.

[`inputOK_cmd_reject_lemma`]

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

7.1.8 Authorization

There are two functions for authentication in the parametrizable ssm. One is state-dependent, the other is not.

State-dependent Authorization In ssmPB, the state-dependent authentication function is named *secContext*. *secContext* uses both pattern matching and if-then-else statements. It takes a state and an input list as parameters.

[*secContext_def*]

$$\vdash (\forall xs.$$

```

secContext PLAN_PB xs =
  if getOmniCommand xs = ssmPlanPBComplete then
    [prop (SOME (SLc (OMNI ssmPlanPBComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL crossLD)))]
  else [prop NONE]) ∧

(\forall xs.
  secContext MOVE_TO_ORP xs =
  if getOmniCommand xs = ssmMoveToORPComplete then
    [prop (SOME (SLc (OMNI ssmMoveToORPComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL conductORP)))]
  else [prop NONE]) ∧

(\forall xs.
  secContext CONDUCT_ORP xs =
  if getOmniCommand xs = ssmConductORPComplete then
```

```

[prop (SOME (SLc (OMNI ssmConductORPComplete))) impf
  Name PlatoonLeader controls
  prop (SOME (SLc (PL moveToPB)))]

else [prop NONE] ) ∧

(∀xs .
  secContext MOVE_TO_PB xs =
  if getOmniCommand xs = ssmConductORPComplete then
    [prop (SOME (SLc (OMNI ssmMoveToPBComplete))) impf
      Name PlatoonLeader controls
      prop (SOME (SLc (PL conductPB)))]

  else [prop NONE] ) ∧

  ∀xs .
  secContext CONDUCT_PB xs =
  if getOmniCommand xs = ssmConductPBComplete then
    [prop (SOME (SLc (OMNI ssmConductPBComplete))) impf
      Name PlatoonLeader controls
      prop (SOME (SLc (PL completePB)))]

  else [prop NONE]

```

secContext uses a helper function named *getOmniCommand* to extract the *omniCommand* from the input list. It compares this command to the expected command. If it matches, it returns an implication of the form described at the top of this section.

HOL Definition for *getOmniCommand* *getOmniCommand* also uses an underscore. For brevity, only the HOL definition is presented.

```

val getOmniCommand_def =
Define `

(getOmniCommand ([]:( slCommand command) option , stateRole , 'd,'e)Form list )
  = invalidOmniCommand:omniCommand) /\

(getOmniCommand (((Name Omni) says prop (SOME (SLc (OMNI cmd))))::xs)
  = (cmd:omniCommand)) /\

```

```
(getOmniCommand ((x:((slCommand command) option , stateRole , 'd , 'e)Form)::xs)
= (getOmniCommand xs))`
```

getOmniCommand uses pattern matching and recursion. The first pattern matches the input against the empty list. It returns *invalidOmniCommand*. The second pattern matches against a statement of the form *Omni says cmd*. It returns *cmd*. The final pattern is a recursive call to *getOmniCommand*.

State-independent Authorization The state-independent authorization function takes an input list as a parameter. It uses a helper function called *secHelper*. *secHelper* calls *getOmniCommand*. It returns a authorization statement of the form

Omni controls cmd.

[[secAuthorization_def](#)]

$\vdash \forall xs. \text{secAuthorization } xs = \text{secHelper} (\text{getOmniCommand } xs)$

[[secHelper_def](#)]

$\vdash \forall cmd.$
 $\text{secHelper } cmd =$
 $[\text{Name Omni controls prop (SOME (SLc (OMNI } cmd)))]$

7.1.9 Proved Theorems

PlatoonLeader_PLAN_PB_exec_justified_thm This theorem proves that transition from the PLAN_PB state to the MOVE_TO_ORP state is justified for the following assumptions:

- The current state is PLAN_PB
- Omni says ssmPlanPBComplete
- PlatoonLeader says crossLD

Most proofs are similar to this. Therefore, this one will be discussed in detail and used as a reference.

PlatoonLeader_PLAN_PB_exec_justified_thm begins by specializing *TR_exec_cmd_rule* described in section 6.3.9 of chapter 6. That rule is repeated here for reference.

[*TR_exec_cmd_rule*]

```

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

```

Specializing requires use of the ISPECL rule. This rule takes a list of parameters (lists are enclosed in square brackets) and a theorem (the theorem to be specialized).

```
val thPlanPB =
  ISPECL
  [inputOK:((slCommand command)option , stateRole , 'd,'e)Form -> bool``,
   secAuthorization :((slCommand command)option , stateRole , 'd,'e)Form list ->
    ((slCommand command)option , stateRole , 'd,'e)Form list ``,
   secContext: (slState) ->
    ((slCommand command)option , stateRole , 'd,'e)Form list ->
    ((slCommand command)option , stateRole , 'd,'e)Form list ``,
   [(Name Omni) says (prop (SOME (SLc (OMNI ssmPlanPBComplete))))
    :((slCommand command)option , stateRole , 'd,'e)Form;
   (Name PlatoonLeader) says (prop (SOME (SLc (PL crossLD))))
    :((slCommand command)option , stateRole , 'd,'e)Form] ``,
   ins:((slCommand command)option , stateRole , 'd,'e)Form list list ``,
   (PLAN_PB) ``,
   outs:slOutput output list trType list ``] TR_exec_cmd_rule
```

The `TthPlanPB` is a temporary function used to store an intermediate value. In it, the following values are substituted into the `TR_exec_cmd_rule`.

- `inputOK`: substitutes for `elementTest`
- `secAuthorization`: substitutes for `context`
- `secContext`: substitutes for `stateInterp`
- an input list: substitutes for `s`
- the current state: substitutes for `s`
- an output list: substitutes for `out`

The `TR_exec_cmd_rule` is already proved in the parametrizable `ssm`. It is an implication with a hypothesis and a conclusion. But, with the specialization, it is necessary to prove that the theorem is valid after the authentication and authorization are applied to the input list. To do this, the `thPlanPB` is destructed into a hypothesis

and conclusion. Each of these form the basis for a lemma. The lemmas are then used to prove the overall theorem.

The first lemma begins with the hypothesis of *thPlanPB*. It extracts the hypothesis using the *dest_imp* to destroy the implication and the *fst* function to retain the first part of the result.

```
fst (dest_imp (concl thPlanPB)))
```

The result is a HOL term which is used in a tactical proof (backwards proof).

```
val PlatoonLeader_PLAN_PB_exec_lemma =
TAC_PROOF(
[] , fst (dest_imp (concl thPlanPB))) ,
REWRITE_TAC[CFGInterpret_def, secContext_def, secAuthorization_def, secHelper_def,
propCommandList_def, extractPropCommand_def, inputList_def,
getOmniCommand_def,
MAP, extractInput_def, satList_CONS, satList_nil, GSYM satList_conj] THEN
PROVE_TAC[Controls, Modus_Ponens])
```

The proof consists of a call to the *REWRITE_TAC* with several definitions passed as parameters. The reader should recognize most of these commands which are described elsewhere in this master thesis¹. The last line of the proof uses *PROVE_TAC*, an automatic prover, with the Controls and Modus_Ponens rules. These rules are described in the HOL representation of the ACL.

This lemma is saved as *PlatoonLeader_PLAN_PB_exec_lemma*. The HOL generated output is shown below.

[PlatoonLeader_PLAN_PB_exec_lemma]

```
|- ! M Oi Os .
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
```

¹Those that aren't can be found at HOL's extensive documentation <https://hol-theorem-prover.org/kananaskis-11-helpdocs/help/HOLindex.html>

```

([Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))) ] :: ins) PLAN_PB
outs) ⇒
(M, Oi, Os) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))]

```

The next lemma works on the conclusion of *thPlanPB*. It extracts the conclusion using the *dest_imp* and *snd* rules.

```
snd(dest_imp(concl thPlanPB))
```

It is also a tactical proof that uses *PROVE_TAC*. *PROVE_TAC* takes two parameters. The first is the previous lemma and the second is *TR_exec_cmd_rule*. It is named

```

val PlatoonLeader_PLAN_PB_exec_justified_lemma =
TAC_PROOF(
[], snd(dest_imp(concl thPlanPB)),
PROVE_TAC[PlatoonLeader_PLAN_PB_exec_lemma, TR_exec_cmd_rule])

```

This lemma is saved as *PlatoonLeader_PLAN_PB_exec_justified_lemma*. The HOL generated output is shown below.

```
[PlatoonLeader_PLAN_PB_exec_justified_lemma]
```

```

⊢ ∀ NS Out M Oi Os.
  TR (M, Oi, Os)
  (exec
    (inputList
      [Name Omni says

```

```

prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))])
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))])::ins) PLAN_PB outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(exec
(inputList
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))]))
(Out PLAN_PB
(exec
(inputList
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))])::outs)) ⇔
authenticationTest inputOK
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))] ∧
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says

```

```

prop (SOME (SLc (OMNI ssmPlanPBComplete))) ;
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))) ] :: ins) PLAN_PB
outs) ∧
(M, Oi, Os) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete))) ;
Name PlatoonLeader says prop (SOME (SLc (PL crossLD))) ]

```

With this last lemma, the main theorem is proved using a simple forward proof. The proof uses the *REWRITE_RULE* with several theorems including the previous lemma.

```

val PlatoonLeader_PLAN_PB_exec_justified_thm =
REWRITE_RULE[ inputList_def , extractInput_def , MAP, propCommandList_def ,
extractPropCommand_def , PlatoonLeader_PLAN_PB_exec_lemma]
PlatoonLeader_PLAN_PB_exec_justified_lemma

```

The HOL generated output is shown below.

```
[PlatoonLeader_PLAN_PB_exec_justified_thm]
```

```

⊢ ∀ NS Out M Oi Os .
TR (M, Oi, Os)
(exec
[SOME (SLc (OMNI ssmPlanPBComplete));
 SOME (SLc (PL crossLD))])
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete))) ;
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))) ] :: ins) PLAN_PB outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB

```

```

(exec
  [SOME (SLc (OMNI ssmPlanPBComplete));
   SOME (SLc (PL crossLD)) ]))

(Out PLAN_PB
(exec
  [SOME (SLc (OMNI ssmPlanPBComplete));
   SOME (SLc (PL crossLD)) ] )::outs) )  $\iff$ 
authenticationTest inputOK

[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))) ]  $\wedge$ 
CFGInterpret ( $M, O_i, O_s$ )
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))) ] )::ins) PLAN_PB
outs)  $\wedge$ 
( $M, O_i, O_s$ ) satList
[prop (SOME (SLc (OMNI ssmPlanPBComplete)));
prop (SOME (SLc (PL crossLD))) ]

```

With few exceptions, the remaining proofs in all SSMs follow the same format.

PlatoonLeader_PLAN_PB_trap_justified_thm

PlatoonLeader_PLAN_PB_trap_justified_thm proves the *trap* transition is justified in the following case

- current state is PLAN_PB

- Omni says someOmniCommand and someOmniCommand \neq ssmPlanPBComplete
- PlatoonLeader says crossLD

This is a *trap* because both Omni and PlatoonLeader are authenticated. But, PlatoonLeader is not authorized to transition to the next state unless Omni says *ssmPlanPBComplete*.

PlatoonLeader_PLAN_PB_trap_justified_thm follows the same pattern of implications destruction and lemma proofs. The difference is that the *TR_trap_cmd_rule* is specialized instead of the *TR_exec_cmd_rule*. The two lemmas and theorem are shown below.

[*PlatoonLeader_PLAN_PB_trap_lemma*]

```

 $\vdash \text{omniCommand} \neq \text{ssmPlanPBComplete} \Rightarrow$ 
 $(s = \text{PLAN\_PB}) \Rightarrow$ 
 $\forall M \ Oi \ Os.$ 
 $\text{CFGInterpret } (M, Oi, Os)$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand))});$ 
 $\text{Name PlatoonLeader says}$ 
 $\text{prop (SOME (SLc (PL crossLD)))}] :: ins) \text{ PLAN\_PB}$ 
 $outs) \Rightarrow$ 
 $(M, Oi, Os) \text{ sat prop NONE}$ 

```

[*PlatoonLeader_PLAN_PB_trap_justified_lemma*]

```

 $\vdash \text{omniCommand} \neq \text{ssmPlanPBComplete} \Rightarrow$ 
 $(s = \text{PLAN\_PB}) \Rightarrow$ 
 $\forall NS \ Out \ M \ Oi \ Os.$ 

```

```

TR (M, Oi, Os)

(trap

  (inputList
    [Name Omni says
      prop (SOME (SLc (OMNI omniCommand))) ;
      Name PlatoonLeader says
      prop (SOME (SLc (PL crossLD))))])))

(CFG inputOK secContext secAuthorization
  ([Name Omni says prop (SOME (SLc (OMNI omniCommand))) ;
  Name PlatoonLeader says
  prop (SOME (SLc (PL crossLD)))]) :: ins) PLAN_PB outs)
  (CFG inputOK secContext secAuthorization ins
    (NS PLAN_PB

      (trap

        (inputList
          [Name Omni says
            prop (SOME (SLc (OMNI omniCommand))) ;
            Name PlatoonLeader says
            prop (SOME (SLc (PL crossLD))))]))))

      (Out PLAN_PB

        (trap

          (inputList
            [Name Omni says
              prop (SOME (SLc (OMNI omniCommand))) ;
              Name PlatoonLeader says
              prop (SOME (SLc (PL crossLD))))]) :: outs) ) ⇔
authenticationTest inputOK

  [Name Omni says prop (SOME (SLc (OMNI omniCommand))) ;
  Name PlatoonLeader says
  prop (SOME (SLc (PL crossLD)))] ∧

CFGInterpret (M, Oi, Os)

```

```

(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (SLc (OMNI omniCommand))) ;
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))]) ::ins) PLAN_PB
outs)  $\wedge$  (M, Oi, Os) sat prop NONE

```

[PlatoonLeader_PLAN_PB_trap_justified_thm]

```

 $\vdash omniCommand \neq \text{ssmPlanPBComplete} \Rightarrow$ 
(s = PLAN_PB)  $\Rightarrow$ 
 $\forall NS\ Out\ M\ Oi\ Os.$ 
TR (M, Oi, Os)
(trap
[SOME (SLc (OMNI omniCommand));
SOME (SLc (PL crossLD))])
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (SLc (OMNI omniCommand))) ;
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))]) ::ins) PLAN_PB outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(trap
[SOME (SLc (OMNI omniCommand));
SOME (SLc (PL crossLD))]))
(Out PLAN_PB
(trap
[SOME (SLc (OMNI omniCommand));
SOME (SLc (PL crossLD))]) ::outs) )  $\iff$ 
authenticationTest inputOK
[Name Omni says prop (SOME (SLc (OMNI omniCommand))) ;
Name PlatoonLeader says

```

```

prop (SOME (SLc (PL crossLD))) ]  $\wedge$ 
CFGInterpret ( $M, Oi, Os$ )
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (SLc (OMNI omniCommand)) );
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))) ] ::ins) PLAN_PB
outs)  $\wedge$  ( $M, Oi, Os$ ) sat prop NONE

```

PlatoonLeader_Omni_notDiscard_s1Command_thm This theorem proves that if the PlatoonLeader issues an omniCommand and Omni issues a plCommand, the command is not discarded. The reason for this is that plCommand and omniCommand are both s1Command and inputOK authenticates both principals for s1Command.

This proof specializes the *TR_discard_cmd_rule*. This follows a different pattern with no lemmas. The HOL generated output is shown below.

[PlatoonLeader_Omni_notDiscard_s1Command_thm]

```

 $\vdash \forall NS\ Out\ M\ Oi\ Os.$ 
 $\neg\text{TR}\ (M, Oi, Os)$ 
(discard
[SOME (SLc (PL plCommand)) ;
SOME (SLc (OMNI omniCommand)) ])
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (SLc (PL plCommand)) );
Name PlatoonLeader says
prop (SOME (SLc (OMNI omniCommand)) ] ::ins) PLAN_PB
outs)
(CFG inputOK secContext secAuthorization ins
( $NS$  PLAN_PB
(discard

```

```

[ SOME (SLC (PL plCommand) ) ;
  SOME (SLC (OMNI omniCommand) ) ] )
(Out PLAN_PB
(discard
[ SOME (SLC (PL plCommand) ) ;
  SOME (SLC (OMNI omniCommand) ) ] ) ::outs) )

```

Theorems for each transition are possible. For the most part, they are a cut-n-paste of the theorems above with a few keywords changed. But, they become repetitive and no new information is gained by generating them.

7.2 ssmConductORP: Multiple Principals

ssmConductORP is an example of a SSM with more than one principal authorized to execute transitions among states. The diagram is shown in figure 7.2.

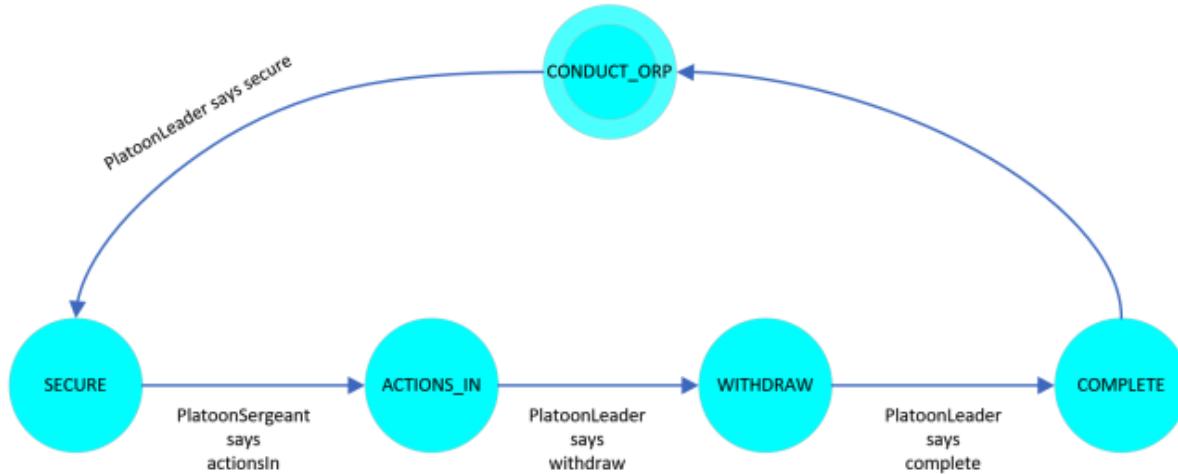


Figure 7.2: Horizontal slice: ConductORP diagram.

Other than the number of principals, ssmConductToORP follows the same structure as ssmPB described in the previous section.

7.2.1 Principals

The principals are defined in the datatype stateRole. There are three principals including Omni.

```
stateRole = PlatoonLeader | PlatoonSergeant | Omni
```

7.2.2 States

States are defined as slState. The names differ only slightly from the diagram but are straight forward.

```
slState = CONDUCT_ORP  
| SECURE  
| ACTIONS_IN  
| WITHDRAW  
| COMPLETE
```

7.2.3 Outputs

Outputs are named similarly to ssmPB outputs.

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

```
| unAuthorized
```

7.2.4 Commands

The slCommand datatype defines three datatype variables, one for each principal. Each are further defined.

```
slCommand =PL plCommand  
| PSG psgCommand  
| OMNI omniCommand
```

plCommand defines the PlatoonLeader commands.

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

psgCommand defines the PlatoonSergeant commands. *psgCommand* = actionsIn |
psgIncomplete

omniCommand defines the Omni commands.

```
omniCommand = ssmSecureComplete  
| ssmActionsIncomplete  
| ssmWithdrawComplete  
| invalidOmniCommand
```

7.2.5 Next-State Function

The next-state function follows the same pattern as for ssmPB. The only difference is that one of the transitions requires a psgCommand instead of a plCommand.

[conductORPNS_def]

```

 $\vdash (\text{conductORPNS CONDUCT\_ORP } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{secure} \text{ then SECURE} \text{ else CONDUCT\_ORP}) \wedge$ 
 $\quad (\text{conductORPNS SECURE } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPsgCom } x = \text{actionsIn} \text{ then ACTIONS\_IN} \text{ else SECURE}) \wedge$ 
 $\quad (\text{conductORPNS ACTIONS\_IN } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{withdraw} \text{ then WITHDRAW} \text{ else ACTIONS\_IN}) \wedge$ 
 $\quad (\text{conductORPNS WITHDRAW } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{complete} \text{ then COMPLETE} \text{ else WITHDRAW}) \wedge$ 
 $\quad (\text{conductORPNS } s \text{ (trap } x) = s) \wedge$ 
 $\quad (\text{conductORPNS } s \text{ (discard } x) = s)$ 
```

The next-state function requires two helper functions just as the next-state function for ssmPB. These are *getPlCom* and *getPsgCom*. The former extracts PlatoonLeader commands and the later extracts PlatoonSergeant commands. They both use the underscore as a wild card. The HOL definition bot both functions is shown below.

```

val getPlCom_def =
Define `

  (getPlCom ([]:((slCommand command)option)list)
    = plIncomplete:plCommand) /\

  (getPlCom (SOME (SLc (PL cmd)): (slCommand command)option :: xs)
    = cmd:plCommand) /\

  (getPlCom (_::(xs :(slCommand command)option list))
    = (getPlCom xs))`


val getPsgCom_def =
Define `

  (getPsgCom ([]:((slCommand command)option)list)
```

```

= psgIncomplete : psgCommand) /\

(getPsgCom (SOME (SLc (PSG cmd)) : (slCommand command) option :: xs)
= cmd : psgCommand) /\

(getPsgCom (_ :: (xs : (slCommand command) option list))
= (getPsgCom xs)) `
```

7.2.6 Next-Output Function

The next-output function follows the same pattern as the next-state function. It returns outputs instead of states. [conductORPOut_def]

```

 $\vdash$  (conductORPOut CONDUCT_ORP (exec x) =
  if getPlCom x = secure then Secure else ConductORP)  $\wedge$ 
  (conductORPOut SECURE (exec x) =
    if getPsgCom x = actionsIn then ActionsIn else Secure)  $\wedge$ 
  (conductORPOut ACTIONS_IN (exec x) =
    if getPlCom x = withdraw then Withdraw else ActionsIn)  $\wedge$ 
  (conductORPOut WITHDRAW (exec x) =
    if getPlCom x = complete then Complete else Withdraw)  $\wedge$ 
  (conductORPOut s (trap x) = unAuthorized)  $\wedge$ 
  (conductORPOut s (discard x) = unAuthenticated)
```

7.2.7 Authentication

Authentication uses the *inputOK* function. It is the same function as the ssmPB *inputOK* except that it adds and additional pattern matching to authenticate the PlatoonSergeant.

```

val inputOK_def =
Define
`(inputOK
  ((Name PlatoonLeader) says (prop (cmd : (slCommand command) option)))
```

```

:(( slCommand command)option ,stateRole , 'd , 'e)Form) = T) /\ 
(inputOK
((Name PlatoonSergeant) 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)`
```

It is straight forward to prove that any command that is not issued by a principal is rejected. This should follow directly from the definition of *inputOK*.

[*inputOK_cmd_reject_lemma*]

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

7.2.8 Authorization

As in ssmPB and all SSMs, there is a state-dependent and state-independent authorization function.

State-dependent Authorization Like ssmPB, the state-dependent authorization function is named *secContext*. It takes a state and an input list as parameters. It returns an authorization statement. It follows the same logic as *secContext* in ssmPB.

[*secContext_def*]

$\vdash (\forall xs.$
 $\text{secContext CONDUCT_ORP } xs =$
 $[\text{Name PlatoonLeader controls}$
 $\text{prop} (\text{SOME} (\text{SLc} (\text{PL secure}))))]) \wedge$
 $(\forall xs.$

```

secContext SECURE xs =
if getOmniCommand xs = ssmSecureComplete then
  [prop (SOME (SLc (OMNI ssmSecureComplete))) impf
   Name PlatoonSergeant controls
   prop (SOME (SLc (PSG actionsIn)))]
else [prop NONE]) ∧
(∀xs.

secContext ACTIONS_IN xs =
if getOmniCommand xs = ssmActionsIncomplete then
  [prop (SOME (SLc (OMNI ssmActionsIncomplete))) impf
   Name PlatoonLeader controls
   prop (SOME (SLc (PL withdraw)))]
else [prop NONE]) ∧
∀xs.

secContext WITHDRAW xs =
if getOmniCommand xs = ssmWithdrawComplete then
  [prop (SOME (SLc (OMNI ssmWithdrawComplete))) impf
   Name PlatoonLeader controls
   prop (SOME (SLc (PL complete)))]
else [prop NONE]

```

secContext in *ssmConductORP* uses the exact same *getOmniCommand* function. It is repeated here as a reference.

```

val getOmniCommand_def =
Define `

(getOmniCommand ([]:( (slCommand command) option , stateRole , 'd,'e)Form list ))
  = invalidOmniCommand:omniCommand) /\

(getOmniCommand (((Name Omni) says prop (SOME (SLc (OMNI cmd))))::xs)
  = (cmd:omniCommand)) /\

(getOmniCommand ((x:((slCommand command) option , stateRole , 'd,'e)Form)::xs)
  = (getOmniCommand xs))`
```

State-independent Authorization Like ssmPB, the state-independent authorization function is named *secAuthorization*. It takes only an input list as a parameter. This is the exact same *secAuthorization* as in ssmPB. It also uses the exact same *secHelper* function as ssmPB. There are repeated here as a reference.

[*secAuthorization_def*]

$\vdash \forall xs. \text{ secAuthorization } xs = \text{secHelper} (\text{getOmniCommand } xs)$

[*secHelper_def*]

$\vdash \forall cmd.$

```
secHelper cmd =
[Name Omni controls prop (SOME (SLC (OMNI cmd)))]
```

7.2.9 Proved Theorems

These theorems follow those in section 7.1.9 for ssmPB. There are few changes and thus detailed explanation with provide no new or insightful information. They are presented here for completeness of this section.

PlatoonSergeant _ SECURE _ exec _ justified _ thm This theorem justifies transition from the SECURE state to the ACTIONS_IN state. The authenticated principal is the PlatoonSergeant. This theorem specializes the *TR_exec_cmd_rule*. The two lemmas and theorem are shown below.

[*PlatoonSergeant_SECURE_exec_lemma*]

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

```
CFGInterpret (M, Oi, Os)
```

```

(CFG inputOK secContext secAuthorization
 ([Name Omni says
    prop (SOME (SLc (OMNI ssmSecureComplete)));
    Name PlatoonSergeant says
    prop (SOME (SLc (PSG actionsIn)))]) :: ins) SECURE
outs) ⇒
(M, Oi, Os) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))]

```

[PlatoonSergeant_SECURE_exec_justified_lemma]

```

⊤ ∀ NS Out M Oi Os.
TR (M, Oi, Os)
(exec
(inputList
[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))))

(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))]) :: ins) SECURE
outs)

(CFG inputOK secContext secAuthorization ins
(NS SECURE

```

```

(exec
  (inputList
    [Name Omni says
      prop (SOME (SLc (OMNI ssmSecureComplete)));
      Name PlatoonSergeant says
      prop (SOME (SLc (PSG actionsIn))))])))

(Out SECURE

(exec
  (inputList
    [Name Omni says
      prop (SOME (SLc (OMNI ssmSecureComplete)));
      Name PlatoonSergeant says
      prop (SOME (SLc (PSG actionsIn))))]):::
  outs) )  ⇔

authenticationTest inputOK

[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn))) ] ∧
CFGInterpret (M, Oi, Os)

(CFG inputOK secContext secAuthorization

([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn))))]::ins) SECURE
outs) ∧

(M, Oi, Os) satList

propCommandList

[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says

```

```
prop (SOME (SLc (PSG actionsIn))) ]
```

[PlatoonSergeant_SECURE_exec_justified_thm]

```
⊢ ∀ NS Out M Oi Os.
  TR (M, Oi, Os)
  (exec
    [SOME (SLc (OMNI ssmSecureComplete));
     SOME (SLc (PSG actionsIn))])
  (CFG inputOK secContext secAuthorization
    ([Name Omni says
       prop (SOME (SLc (OMNI ssmSecureComplete)));
       Name PlatoonSergeant says
       prop (SOME (SLc (PSG actionsIn))))] :: ins) SECURE
    outs)
  (CFG inputOK secContext secAuthorization ins
    (NS SECURE
      (exec
        [SOME (SLc (OMNI ssmSecureComplete));
         SOME (SLc (PSG actionsIn))]))
    (Out SECURE
      (exec
        [SOME (SLc (OMNI ssmSecureComplete));
         SOME (SLc (PSG actionsIn))]) :: outs)) ⇔
  authenticationTest inputOK
  [Name Omni says
   prop (SOME (SLc (OMNI ssmSecureComplete)));
   Name PlatoonSergeant says
   prop (SOME (SLc (PSG actionsIn)))] ∧
  CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secAuthorization
```

```

([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn))) ]::ins) SECURE
outs) ∧
(M, Oi, Os) satList
[prop (SOME (SLc (OMNI ssmSecureComplete)));
prop (SOME (SLc (PSG actionsIn)))]

```

PlatoonLeader_ACTIONS_IN_exec_justified_thm This theorem justifies transition from the ACTIONS_IN state to the WITHDRAW state. The authenticated principal is the PlatoonLeader. This theorem specializes the *TR_exec_cmd_rule*. The two lemmas and theorem are shown below.

[PlatoonLeader_ACTIONS_IN_exec_lemma]

```

⊢ ∀ M Oi Os .
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmActionsInIncomplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))) ]::ins) ACTIONS_IN
outs) ⇒
(M, Oi, Os) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmActionsInIncomplete)));
Name PlatoonLeader says prop (SOME (SLc (PL withdraw)))]
```

[PlatoonLeader_ACTIONS_IN_exec_justified_lemma]

```

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

TR (M, Oi, Os)

(exec

  (inputList
    [Name Omni says
      prop (SOME (SLc (OMNI ssmActionsIncomplete)));
      Name PlatoonLeader says
      prop (SOME (SLc (PL withdraw))))])

  (CFG inputOK secContext secAuthorization
    ([Name Omni says
      prop (SOME (SLc (OMNI ssmActionsIncomplete)));
      Name PlatoonLeader says
      prop (SOME (SLc (PL withdraw))))] :: ins) ACTIONS_IN
    outs)

  (CFG inputOK secContext secAuthorization ins
    (NS ACTIONS_IN

      (exec
        (inputList
          [Name Omni says
            prop
            (SOME (SLc (OMNI ssmActionsIncomplete)));
            Name PlatoonLeader says
            prop (SOME (SLc (PL withdraw))))])

      (Out ACTIONS_IN

        (exec
          (inputList
            [Name Omni says
              prop
              (SOME (SLc (OMNI ssmActionsIncomplete)));
              Name PlatoonLeader says
              prop (SOME (SLc (PL withdraw))))] ::
```

```

 $outs) \Leftrightarrow$ 
authenticationTest inputOK
[Name Omni says
prop (SOME (SLc (OMNI ssmActionsIncomplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw)))] \wedge
CFGInterpret ( $M, Oi, Os$ )
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmActionsIncomplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw)))]) ::ins) ACTIONS_IN
 $outs) \wedge$ 
( $M, Oi, Os$ ) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmActionsIncomplete)));
Name PlatoonLeader says prop (SOME (SLc (PL withdraw)))]
```

[PlatoonLeader_ACTIONS_IN_exec_justified_thm]

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

TR (M, Oi, Os)

(exec

[SOME (SLc (OMNI ssmActionsIncomplete));
SOME (SLc (PL withdraw))])

(CFG inputOK secContext secAuthorization

([Name Omni says

prop (SOME (SLc (OMNI ssmActionsIncomplete)));
Name PlatoonLeader says

prop (SOME (SLc (PL withdraw)))]) ::ins) ACTIONS_IN

```

 $outs)$ 

 $(\text{CFG } \text{inputOK } \text{secContext } \text{secAuthorization } ins$ 
 $(NS \text{ ACTIONS\_IN}$ 
 $\text{(exec}$ 
 $\text{[SOME (SLc (OMNI ssmActionsIncomplete)) ;}$ 
 $\text{SOME (SLc (PL withdraw))]))$ 
 $(Out \text{ ACTIONS\_IN}$ 
 $\text{(exec}$ 
 $\text{[SOME (SLc (OMNI ssmActionsIncomplete)) ;}$ 
 $\text{SOME (SLc (PL withdraw))]) :: outs) ) \iff$ 
 $\text{authenticationTest inputOK}$ 
 $\text{[Name Omni says}$ 
 $\text{prop (SOME (SLc (OMNI ssmActionsIncomplete))) ;}$ 
 $\text{Name PlatoonLeader says}$ 
 $\text{prop (SOME (SLc (PL withdraw)))] \wedge}$ 
 $\text{CFGInterpret } (M, Oi, Os)$ 
 $(\text{CFG } \text{inputOK } \text{secContext } \text{secAuthorization}$ 
 $\text{([Name Omni says}$ 
 $\text{prop (SOME (SLc (OMNI ssmActionsIncomplete))) ;}$ 
 $\text{Name PlatoonLeader says}$ 
 $\text{prop (SOME (SLc (PL withdraw)))] :: ins) ACTIONS\_IN}$ 
 $outs) \wedge$ 
 $(M, Oi, Os) \text{ satList}$ 
 $\text{[prop (SOME (SLc (OMNI ssmActionsIncomplete))) ;}$ 
 $\text{prop (SOME (SLc (PL withdraw)))]}$ 

```

PlatoonLeader_ACTIONS_IN_trap_justified_thm This theorem justifies trapping the transition from the ACTIONS_IN state to the WITHDRAW state. The authenticated principal is the PlatoonLeader. But, in this case, Omni does not issue the command *ssmActionsIncomplete*. Therefore, the transition is not authorized. This

theorem specializes the *TR_exec_cmd_rule*. The two lemmas and theorem are shown below.

[PlatoonLeader_ACTIONS_IN_trap_lemma]

```

 $\vdash \text{omniCommand} \neq \text{ssmActionsInComplete} \Rightarrow$ 
 $(s = \text{ACTIONS\_IN}) \Rightarrow$ 
 $\forall M \ Oi \ Os.$ 
 $\text{CFGInterpret } (M, Oi, Os)$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand))});$ 
 $\text{Name PlatoonLeader says}$ 
 $\text{prop (SOME (SLc (PL withdraw)))}] :: ins) \text{ ACTIONS\_IN}$ 
 $outs) \Rightarrow$ 
 $(M, Oi, Os) \text{ sat prop NONE}$ 

```

[PlatoonLeader_ACTIONS_IN_trap_justified_lemma]

```

 $\vdash \text{omniCommand} \neq \text{ssmActionsInComplete} \Rightarrow$ 
 $(s = \text{ACTIONS\_IN}) \Rightarrow$ 
 $\forall NS \ Out \ M \ Oi \ Os.$ 
 $\text{TR } (M, Oi, Os)$ 
 $(\text{trap}$ 
 $(\text{inputList}$ 
 $([\text{Name Omni says}$ 
 $\text{prop (SOME (SLc (OMNI omniCommand))});$ 
 $\text{Name PlatoonLeader says}$ 
 $\text{prop (SOME (SLc (PL withdraw)))}))$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $([\text{Name Omni says prop (SOME (SLc (OMNI omniCommand))});$ 
 $\text{Name PlatoonLeader says}$ 

```

```

prop (SOME (SLc (PL withdraw))) ] ::ins) ACTIONS_IN
outs)

(CFG inputOK secContext secAuthorization ins
(NS ACTIONS_IN
(trap
(inputList
[Name Omni says
prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))))))

(Out ACTIONS_IN
(trap
(inputList
[Name Omni says
prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))))]) ::

outs) ) ⇔
authenticationTest inputOK
[Name Omni says prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw)))] ∧
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw)))] ::ins) ACTIONS_IN
outs) ∧ (M, Oi, Os) sat prop NONE

```

[PlatoonLeader_ACTIONS_IN_trap_justified_thm]

```

 $\vdash \text{omniCommand} \neq \text{ssmActionsIncomplete} \Rightarrow$ 
 $(s = \text{ACTIONS\_IN}) \Rightarrow$ 
 $\forall NS \text{ Out } M \text{ Oi } Os.$ 
 $\text{TR } (M, Oi, Os)$ 
 $(\text{trap}$ 
 $\quad [\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand})) ;$ 
 $\quad \text{SOME } (\text{SLc } (\text{PL withdraw}))])$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $\quad ([\text{Name Omni says prop } (\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand}))) ;$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL withdraw})))]) :: ins) \text{ ACTIONS\_IN}$ 
 $\quad outs)$ 
 $(\text{CFG inputOK secContext secAuthorization } ins$ 
 $\quad (NS \text{ ACTIONS\_IN}$ 
 $\quad (\text{trap}$ 
 $\quad [\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand})) ;$ 
 $\quad \text{SOME } (\text{SLc } (\text{PL withdraw})))])$ 
 $(Out \text{ ACTIONS\_IN}$ 
 $\quad (\text{trap}$ 
 $\quad [\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand})) ;$ 
 $\quad \text{SOME } (\text{SLc } (\text{PL withdraw})))]) :: outs) ) \iff$ 
 $\text{authenticationTest inputOK}$ 
 $\quad [\text{Name Omni says prop } (\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand}))) ;$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL withdraw})))] \wedge$ 
 $\text{CFGInterpret } (M, Oi, Os)$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $\quad ([\text{Name Omni says prop } (\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand}))) ;$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL withdraw})))]) :: ins) \text{ ACTIONS\_IN}$ 
 $\quad outs) \wedge (M, Oi, Os) \text{ sat prop NONE}$ 

```

PlatoonSergeant _ SECURE _ exec _ justified _ thm This theorem justifies transition from the SECURE state to the ACTIONS_IN state. The authenticated principal is the PlatoonSergeant. This theorem specializes the *TR_exec_cmd_rule*. The two lemmas and theorem are shown below.

As with ssmPlanPB, other theorems could be proved. But, no new information would be gained.

7.3 ssmPlanPB: Non-sequential Transitions

The ssmPlanPB SSM is one of the first SSMs². It is the only SSM that is not integrated with the Omni principal. But, it is also the only SSM that uses a non-sequential progression through states. Figure 7.3 is a diagram of this SSM.

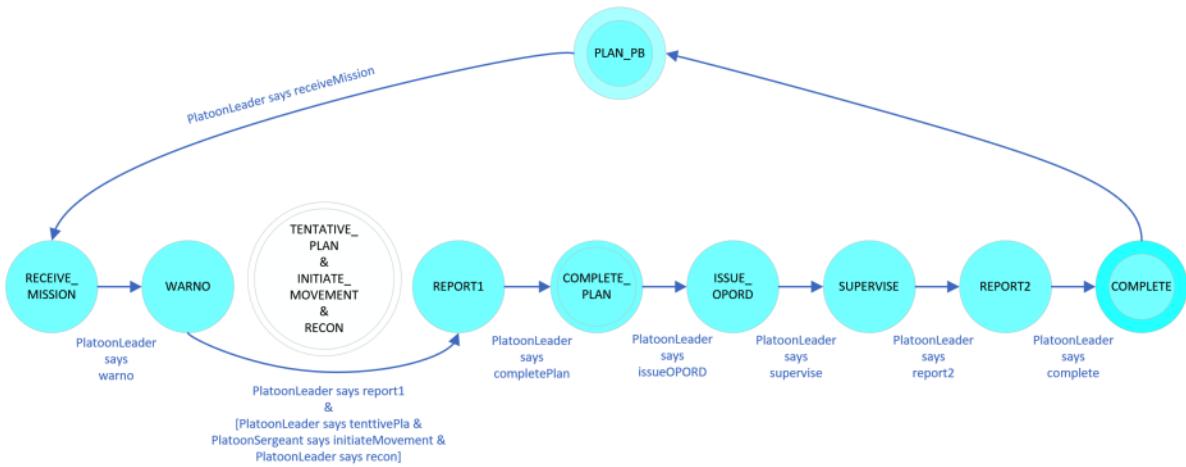


Figure 7.3: Horizontal slice: PlanPB diagram.

The three non-sequential states are hidden in the white circle in the diagram. They are TENTATIVE_PLAN, INITIATE_MOVEMENT, and RECON. These three states may be completed in any order, but all three must be completed before progressing to the

²This is only partially true. The first SSM was ssmPB. But, upon working with the SSM in this section, the parametrizable ssm needed to be updated to accommodate multiple input statements. ssmPlanPB was the first SSM used with this new parametrizable ssm. All other SSMs were redone with the updated parametrizable ssm.

next state REPORT1.

To handle this, the three non-sequential states are combined into a "virtual state." This state does not exist. But, completion of these states must precede transition from the WARNO state to the REPORT1 state. Completion is indicated by the following three ACL statements

PlatoonLeader says tentativePlan

PlatoonSergeant says initiateMovement

PlatoonLeader says recon

when combined with a request from the PlatoonLeader to transition to the REPORT1 state, the input for this transition is an input list with the four statements

PlatoonLeader says tentativePlan

PlatoonSergeant says initiateMovement

PlatoonLeader says recon

PlatoonLeader says report1

The security policy handles this with the following implication

tentativePlan andf

initiateMovement andf

recon impf

PlatoonLeader controls recon

The remaining details of this implementation follow.

7.3.1 Principals

The planning phase SSM has two principals: PlatoonLeader and PlatoonSergeant.

These are defined in the *stateRole* datatype.

```
stateRole = PlatoonLeader | PlatoonSergeant
```

7.3.2 States

There are 12 states. But, TENTATIVE_PLAN, INITIATE_MOVEMENT, and RECON are virtual states and not used in the next-state and next-output functions.

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

7.3.3 Commands

The slCommand datatype for this SSM is defined below.

```
slCommand = PL plCommand | PSG psgCommand
```

The two datatypes for plCommand and psgCommand represent the PlatoonLeader and PlatoonSergeant commands, respectively. These are defined below.

```
plCommand = receiveMission
```

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

```
psgCommand = initiateMovement
```

```
| psgIncomplete  
| invalidPsgCommand
```

Providing each principal with her own set of commands simplifies the authentication and authorization functions.

7.3.4 Output

There are 14 outputs. But, PlanPB, TentativePlan, InitiateMovement, and Recon are not used in the next-output function. The unAuthorized output is returned for trapped commands. The unAuthenticated output is returned for discarded commands.

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

7.3.5 Next-State Function

The next-state function takes the current state and a transition type and inputList pair. It returns the next-state. For the *exec* transition type, an if-then-else decision statement checks the state and input. If the input indicates a transition, then transition occurs, else no state change occurs. The WARNO state is an exception. It requires four inputs as described at the beginning of this section. To do this, four functions extract the

appropriate command from the input list. If all four commands are not present, then the else state returns WARNO (i.e., no state change occurs).

For the *trap* and *discard* transition types (the last two statements on the last line), the current state is returned for any command.

[planPBNS_def]

```

 $\vdash (\text{planPBNS} \text{ WARNO } (\text{exec } x) =$ 
 $\quad \text{if}$ 
 $\quad (\text{getRecon } x = [\text{SOME } (\text{SLc } (\text{PL recon}))]) \wedge$ 
 $\quad (\text{getTenativePlan } x = [\text{SOME } (\text{SLc } (\text{PL tentativePlan}))]) \wedge$ 
 $\quad (\text{getReport } x = [\text{SOME } (\text{SLc } (\text{PL report1}))]) \wedge$ 
 $\quad (\text{getInitMove } x = [\text{SOME } (\text{SLc } (\text{PSG initiateMovement}))])$ 
 $\quad \text{then}$ 
 $\quad \text{REPORT1}$ 
 $\quad \text{else } \text{WARNO}) \wedge$ 
 $(\text{planPBNS PLAN\_PB } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{receiveMission } \text{then } \text{RECEIVE\_MISSION}$ 
 $\quad \text{else } \text{PLAN\_PB}) \wedge$ 
 $(\text{planPBNS RECEIVE\_MISSION } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{warno } \text{then } \text{WARNO } \text{else } \text{RECEIVE\_MISSION}) \wedge$ 
 $(\text{planPBNS REPORT1 } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{completePlan } \text{then } \text{COMPLETE\_PLAN}$ 
 $\quad \text{else } \text{REPORT1}) \wedge$ 
 $(\text{planPBNS COMPLETE\_PLAN } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{ooid } \text{then } \text{OOID } \text{else } \text{COMPLETE\_PLAN}) \wedge$ 
 $(\text{planPBNS OOID } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{supervise } \text{then } \text{SUPERVISE } \text{else } \text{OOID}) \wedge$ 
 $(\text{planPBNS SUPERVISE } (\text{exec } x) =$ 
 $\quad \text{if } \text{getPlCom } x = \text{report2 } \text{then } \text{REPORT2 } \text{else } \text{SUPERVISE}) \wedge$ 
```

```

(planPBNS REPORT2 (exec x) =
  if getPlCom x = complete then COMPLETE else REPORT2) ∧
(planPBNS s (trap v0) = s) ∧ (planPBNS s (discard v1) = s)

```

Five functions are defined (see below) to extract the command from the input list.

These are getRecon, getTentativePlan, getReport, getInitMove, and getPlCom. Each of these functions takes an input list as a parameter and returns a command with its option type. For the sake of brevity, only getRecon is shown below. The other functions are similar.

HOL Definition for getRecon ...

```

val getRecon_def = Define `

  (getRecon ([]:(slCommand command)option , stateRole , 'd,'e)Form list) =
    [NONE] ) /\

  (getRecon ((Name PlatoonLeader) says (prop (SOME (SLc (PL recon)))))

    :((slCommand command)option , stateRole , 'd,'e)Form::xs)
    = [SOME (SLc (PL recon)):(slCommand command)option] ) /\

  (getRecon (_::xs) = getRecon xs)`
```

getRecon is defined recursively and uses pattern matching. The first line beginning with getRecon pattern matches against an empty list of type *((slCommand command)option, stateRole, 'd,'e)Form list* as input. It returns the option type NONE. The next getRecon pattern matches the head of the list with *((Name PlatoonLeader) says (prop (SOME (SLc (PL recon))))* of type *((slCommand command)option, stateRole, 'd,'e)Form list*). This returns *[SOME (SLc (PL recon)):(slCommand command)option]*. The final getRecon pattern matches the wild card "_" to any other input. It recursively calls getRecon on the remainder of the list, while throwing out the head.

HOL Generated Output for getRecon ...

The HOL generated code converts the wild card into all allowable cases for head of the list. It is shown below.

[getRecon_def]

```

 $\vdash (\text{getRecon} [] = [\text{NONE}]) \wedge$ 
 $(\forall xs.$ 
 $\quad \text{getRecon}$ 
 $\quad (\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))}) ::$ 
 $\quad xs) =$ 
 $\quad [\text{SOME (SLc (PL recon))}] \wedge$ 
 $\quad (\forall xs. \text{getRecon (TT::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs. \text{getRecon (FF::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_2. \text{getRecon (prop }v_2::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_3. \text{getRecon (notf }v_3::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_5 v_4. \text{getRecon (v}_4 \text{ andf } v_5::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_7 v_6. \text{getRecon (v}_6 \text{ orf } v_7::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_9 v_8. \text{getRecon (v}_8 \text{ impf } v_9::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_{10} v_{11}. \text{getRecon (v}_{10} \text{ eqf } v_{11}::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_{12}. \text{getRecon (v}_{12} \text{ says TT::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_{12}. \text{getRecon (v}_{12} \text{ says FF::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_{134}.$ 
 $\quad \quad \text{getRecon (Name }v_{134} \text{ says prop NONE::}xs) = \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs v_{146}.$ 
 $\quad \quad \text{getRecon}$ 
 $\quad \quad (\text{Name PlatoonLeader says prop (SOME (ESCC }v_{146})) ::}xs) =$ 
 $\quad \quad \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs.$ 
 $\quad \quad \text{getRecon}$ 
 $\quad \quad (\text{Name PlatoonLeader says}$ 
 $\quad \quad \quad \text{prop (SOME (SLc (PL receiveMission))) ::}xs) =$ 
 $\quad \quad \quad \text{getRecon } xs) \wedge$ 
 $\quad (\forall xs.$ 
 $\quad \quad \text{getRecon}$ 

```

```

        (Name PlatoonLeader says prop (SOME (SLC (PL warno)))::  

         xs) =  

        getRecon xs)  $\wedge$   

        ( $\forall$  xs.  

         getRecon  

         (Name PlatoonLeader says  

          prop (SOME (SLC (PL tentativePlan)))::xs) =  

          getRecon xs)  $\wedge$   

        ( $\forall$  xs.  

         getRecon  

         (Name PlatoonLeader says  

          prop (SOME (SLC (PL report1)))::xs) =  

          getRecon xs)  $\wedge$   

        ( $\forall$  xs.  

         getRecon  

         (Name PlatoonLeader says  

          prop (SOME (SLC (PL completePlan)))::xs) =  

          getRecon xs)  $\wedge$   

        ( $\forall$  xs.  

         getRecon  

         (Name PlatoonLeader says prop (SOME (SLC (PL opoid)))::  

          xs) =  

          getRecon xs)  $\wedge$   

        ( $\forall$  xs.  

         getRecon  

         (Name PlatoonLeader says  

          prop (SOME (SLC (PL supervise)))::xs) =  

          getRecon xs)  $\wedge$   

        ( $\forall$  xs.  

         getRecon  

         (Name PlatoonLeader says

```

```

prop (SOME (SLC (PL report2)))::xs) =
getRecon xs)  $\wedge$ 
( $\forall$  xs .
getRecon
(Name PlatoonLeader says
prop (SOME (SLC (PL complete)))::xs) =
getRecon xs)  $\wedge$ 
( $\forall$  xs .
getRecon
(Name PlatoonLeader says
prop (SOME (SLC (PL plIncomplete)))::xs) =
getRecon xs)  $\wedge$ 
( $\forall$  xs .
getRecon
(Name PlatoonLeader says
prop (SOME (SLC (PL invalidPlCommand)))::xs) =
getRecon xs)  $\wedge$ 
( $\forall$  xs v151 .
getRecon
(Name PlatoonLeader says prop (SOME (SLC (PSG v151)))::xs) =
getRecon xs)  $\wedge$ 
( $\forall$  xs v144 .
getRecon
(Name PlatoonSergeant says prop (SOME v144)::xs) =
getRecon xs)  $\wedge$ 
( $\forall$  xs v68 v136 v135 .
getRecon (v135 meet v136 says prop v68::xs) =
getRecon xs)  $\wedge$ 
( $\forall$  xs v68 v138 v137 .
getRecon (v137 quoting v138 says prop v68::xs) =

```

```

getRecon xs) ∧
(∀xs v69 v12.
  getRecon (v12 says notf v69::xs) = getRecon xs) ∧
(∀xs v71 v70 v12.
  getRecon (v12 says (v70 andf v71)::xs) = getRecon xs) ∧
(∀xs v73 v72 v12.
  getRecon (v12 says (v72 orf v73)::xs) = getRecon xs) ∧
(∀xs v75 v74 v12.
  getRecon (v12 says (v74 impf v75)::xs) = getRecon xs) ∧
(∀xs v77 v76 v12.
  getRecon (v12 says (v76 eqf v77)::xs) = getRecon xs) ∧
(∀xs v79 v78 v12.
  getRecon (v12 says v78 says v79::xs) = getRecon xs) ∧
(∀xs v81 v80 v12.
  getRecon (v12 says v80 speaks_for v81::xs) =
  getRecon xs) ∧
(∀xs v83 v82 v12.
  getRecon (v12 says v82 controls v83::xs) = getRecon xs) ∧
(∀xs v86 v85 v84 v12.
  getRecon (v12 says reps v84 v85 v86::xs) = getRecon xs) ∧
(∀xs v88 v87 v12.
  getRecon (v12 says v87 domi v88::xs) = getRecon xs) ∧
(∀xs v90 v89 v12.
  getRecon (v12 says v89 eqi v90::xs) = getRecon xs) ∧
(∀xs v92 v91 v12.
  getRecon (v12 says v91 doms v92::xs) = getRecon xs) ∧
(∀xs v94 v93 v12.
  getRecon (v12 says v93 eqs v94::xs) = getRecon xs) ∧
(∀xs v96 v95 v12.
  getRecon (v12 says v95 eqn v96::xs) = getRecon xs) ∧
(∀xs v98 v97 v12.
  getRecon (v12 says v97 eqn v98::xs) = getRecon xs)

```

```

getRecon (v12 says v97 lte v98::xs) = getRecon xs) ∧
(∀xs v99 v12 v100.
  getRecon (v12 says v99 lt v100::xs) = getRecon xs) ∧
(∀xs v15 v14.
  getRecon (v14 speaks_for v15::xs) = getRecon xs) ∧
(∀xs v17 v16.
  getRecon (v16 controls v17::xs) = getRecon xs) ∧
(∀xs v20 v19 v18.
  getRecon (reps v18 v19 v20::xs) = getRecon xs) ∧
(∀xs v22 v21. getRecon (v21 domi v22::xs) = getRecon xs) ∧
(∀xs v24 v23. getRecon (v23 eqi v24::xs) = getRecon xs) ∧
(∀xs v26 v25. getRecon (v25 doms v26::xs) = getRecon xs) ∧
(∀xs v28 v27. getRecon (v27 eqs v28::xs) = getRecon xs) ∧
(∀xs v30 v29. getRecon (v29 eqn v30::xs) = getRecon xs) ∧
(∀xs v32 v31. getRecon (v31 lte v32::xs) = getRecon xs) ∧
∀xs v34 v33. getRecon (v33 lt v34::xs) = getRecon xs

```

7.3.6 Next-Output Function

The next-output differs from the next-state function in that it returns an output rather than a state. It follows the same logic. The *trap* and *discard* transition types return the outputs unAuthorized and unAuthenticated, respectively. [planPBOut_def]

```

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

```

```

Report1

else unAuthorized) ∧

(planPBOut PLAN_PB (exec x) =
if getPlCom x = receiveMission then ReceiveMission
else unAuthorized) ∧

(planPBOut RECEIVE_MISSION (exec x) =
if getPlCom x = warno then Warno else unAuthorized) ∧

(planPBOut REPORT1 (exec x) =
if getPlCom x = completePlan then CompletePlan
else unAuthorized) ∧

(planPBOut COMPLETE_PLAN (exec x) =
if getPlCom x = opoid then Opoid else unAuthorized) ∧

(planPBOut OPOID (exec x) =
if getPlCom x = supervise then Supervise
else unAuthorized) ∧

(planPBOut SUPERVISE (exec x) =
if getPlCom x = report2 then Report2 else unAuthorized) ∧

(planPBOut REPORT2 (exec x) =
if getPlCom x = complete then Complete else unAuthorized) ∧

(planPBOut s (trap v0) = unAuthorized) ∧

(planPBOut s (discard v1) = unAuthenticated)

```

7.3.7 Authentication

The authenticationTest function takes an elementTest function and an input list as parameters. authenticationTest is defined in the parametrizable ssm. elementTest function is defined as a parameter in each SSM. In the SSMs, elementTest is named inputOK.

HOL Definition for inputOK The HOL definition for inputOK uses the wild card "`_`". Thus, the HOL definition differs from the HOL generated output. The HOL definition is shown below.

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

inputOK uses pattern matching, but not recursion. The first inputOK pattern matches the input against *((Name PlatoonLeader) says (prop (cmd:((slCommand command)option)))* which is of type *((slCommand command)option, stateRole, 'd, 'e)Form*). This is defined as T (True). This authenticates the PlatoonLeader on any slCommand. The next inputOK pattern matches the input against *((Name PlatoonSergeant) says (prop (cmd:((slCommand command)option)))* which is of the same type as the first pattern. This is defined as T (True). This authenticates the PlatoonSergeant on any slCommand. The last inputOK pattern uses the wild card to match ANY OTHER input. ANY OTHER input is defined as F (false).

HOL Generated Output for inputOK The HOL generated output matches the wild card to all other possible values for the input.

[`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 v1) ⇔ F) ∧
  (inputOK (v2 andf v3) ⇔ F) ∧ (inputOK (v4 orf v5) ⇔ F) ∧
```

```

(inputOK (v6 impf v7)  $\iff$  F)  $\wedge$  (inputOK (v8 eqf v9)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says TT)  $\iff$  F)  $\wedge$  (inputOK (v10 says FF)  $\iff$  F)  $\wedge$ 
(inputOK (v133 meet v134 says prop v66)  $\iff$  F)  $\wedge$ 
(inputOK (v135 quoting v136 says prop v66)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says notf v67)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says (v68 andf v69))  $\iff$  F)  $\wedge$ 
(inputOK (v10 says (v70 orf v71))  $\iff$  F)  $\wedge$ 
(inputOK (v10 says (v72 impf v73))  $\iff$  F)  $\wedge$ 
(inputOK (v10 says (v74 eqf v75))  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v76 says v77)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v78 speaks_for v79)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v80 controls v81)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says reps v82 v83 v84)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v85 domi v86)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v87 eqi v88)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v89 doms v90)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v91 eqs v92)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v93 eqn v94)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v95 lte v96)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v97 lt v98)  $\iff$  F)  $\wedge$ 
(inputOK (v12 speaks_for v13)  $\iff$  F)  $\wedge$ 
(inputOK (v14 controls v15)  $\iff$  F)  $\wedge$ 
(inputOK (reps v16 v17 v18)  $\iff$  F)  $\wedge$ 
(inputOK (v19 domi v20)  $\iff$  F)  $\wedge$ 
(inputOK (v21 eqi v22)  $\iff$  F)  $\wedge$ 
(inputOK (v23 doms v24)  $\iff$  F)  $\wedge$ 
(inputOK (v25 eqs v26)  $\iff$  F)  $\wedge$  (inputOK (v27 eqn v28)  $\iff$  F)  $\wedge$ 
(inputOK (v29 lte v30)  $\iff$  F)  $\wedge$  (inputOK (v31 lt v32)  $\iff$  F)

```

The theorem `inputOK_cmd_reject_lemma` proves that any command that is not requested by a principal is false.

```

val inputOK_cmd_reject_lemma =
TAC_PROOF(
  ([] ,
   ``!cmd. ~ (inputOK
    (( prop (SOME cmd)) : (( slCommand command) option , stateRole , 'd , 'e ) Form)) ``),
PROVE_TAC[inputOK_def])

```

7.3.8 Authorization

Authorization is defined in two ways. Authorization is state-dependent because of the added requirement for the transition from the WARNO state to the REPORT1 state. This is equivalent to the stateInterp function in ssm. In ssmPlanPB, this function is called secContext.

```

val secContext_def = Define `

secContext (s:slState) (x:((slCommand command) option , stateRole , 'd , 'e ) Form list) =
  if (s = WARNO) then
    (if (getRecon           x = [SOME (SLc (PL recon))
                                    :(slCommand command) option] ) /\ \
     (getTenativePlan   x = [SOME (SLc (PL tentativePlan))
                            :(slCommand command) option]) /\ \
     (getReport         x = [SOME (SLc (PL report1))
                            :(slCommand command) option]) /\ \
     (getInitMove       x = [SOME (SLc (PSG initiateMovement))
                            :(slCommand command) option])
    then [
      PL_WARNO_Auth
      :(( slCommand command) option , stateRole , 'd , 'e ) Form;
      (Name PlatoonLeader) controls prop (SOME (SLc (PL recon)));
      (Name PlatoonLeader) controls prop (SOME (SLc (PL tentativePlan)));
      (Name PlatoonSergeant) controls prop (SOME (SLc (PSG initiateMovement)))
    ]
   else [(prop NONE):(( slCommand command) option , stateRole , 'd , 'e ) Form]
  else if ((getPlCom x) = invalidPlCommand)
    then [(prop NONE):(( slCommand command) option , stateRole , 'd , 'e ) Form]
   else [PL_notWARNO_Auth (getPlCom x)]`
```

secContext uses pattern matching, but not recursion. It is similar to the definition for the next state function. If the $s = WARNO$, then secContext looks for all four

statements using the get* functions. If these are all present, then secContext returns *PL_WARNO_Auth*. *PL_WARNO_Auth* contains the implication that authorizes the Platoon Leader on the transition to REPORT1. If all four statements are not present, then *[(prop NONE)]* is returned.

```
val PL_WARNO_Auth_def = Define `

PL_WARNO_Auth =
^(impfTermList
[``(prop (SOME (SLc (PL recon))))
 :((slCommand command)option , stateRole , 'd , 'e)Form`` ,
``(prop (SOME (SLc (PL tentativePlan))))
 :((slCommand command)option , stateRole , 'd , 'e)Form`` ,
``(prop (SOME (SLc (PSG initiateMovement))))
 :((slCommand command)option , stateRole , 'd , 'e)Form`` ,
``(Name PlatoonLeader) controls prop (SOME (SLc (PL report1)))
 :((slCommand command)option , stateRole , 'd , 'e)Form`` ]
``:((slCommand command)option , stateRole , 'd , 'e)Form```
```

The remainder of secContext applies to all other states. If the getPlCom function returns *invalidPlComand*, then secContext returns *[(prop NONE)]* is returned. Otherwise, it returns *PL_notWARNO_Auth*. *PL_notWARNO_Auth* authorizes the PlatoonLeader on any plCommand except *report1*. If the command is *report1* is found, then it returns *[(prop NONE)]*.

```
val PL_notWARNO_Auth_def = Define `

PL_notWARNO_Auth (cmd:plCommand) =
if (cmd = report1) (* report1 exits WARNO state *)
then
  (prop NONE):((slCommand command)option , stateRole , 'd , 'e)Form
else
  ((Name PlatoonLeader) says (prop (SOME (SLc (PL cmd))))
   :((slCommand command)option , stateRole , 'd , 'e)Form) impf
   (((Name PlatoonLeader) controls prop (SOME (SLc (PL cmd)))))
   :((slCommand command)option , stateRole , 'd , 'e)Form)`
```

7.3.9 Proved Theorems

As with ssmConductORP, theorems for ssmPlanPB are similar. Although, this SSM contains no Omni principal, it contains conditional transitions. These theorems differ slightly in their set-up, but then follow the same proof process as

PlatoonLeader_ACTIONS_IN_trap_lemma.

PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm This theorem proves that if the state is not WARNO and the plCommand is not *report1* then execution of any plCommand is justified. It uses two theorems and follow the same pattern as the other three-part proofs. The HOL generated output is shown below.

[`PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma`]

```

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

```

[`PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma`]

```

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

```

```

(M, Oi, Os) satList
propCommandList
  [Name PlatoonLeader says
    prop (SOME (SLc (PL plCommand)))]

```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]

```

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

```

PlatoonLeader_psgCommand_notDiscard_thm This next theorem proves that if the PlatoonLeader issues a psgCommand then that command is not discarded. The reason for this is that the psgCommand is an slCommand. In inputOK, the

PlatoonLeader is authenticated on all psgCommand. Therefore, this command should be trapped an not discarded. It follows the same proof as the ssmPlanPB theorem *PlatoonLeader_Omni_notDiscard_slCommand_thm*. There are no lemmas.

[PlatoonLeader_psgCommand_notDiscard_thm]

$$\vdash \forall NS\ Out\ M\ Oi\ Os.\ \neg \text{TR}\ (M, Oi, Os) \ (\text{discard} [\text{SOME} \ (\text{SLC} \ (\text{PSG} \ psgCommand))]) \\ (\text{CFG inputOK secContext secContextNull} \\ ([\text{Name PlatoonLeader says} \\ \text{prop} \ (\text{SOME} \ (\text{SLC} \ (\text{PSG} \ psgCommand)))] :: ins) \ s \ outs) \\ (\text{CFG inputOK secContext secContextNull} \ ins \\ (NS \ s \ (\text{discard} [\text{SOME} \ (\text{SLC} \ (\text{PSG} \ psgCommand))]))) \\ (Out \ s \ (\text{discard} [\text{SOME} \ (\text{SLC} \ (\text{PSG} \ psgCommand))])) :: \\ outs))$$

PlatoonSergeant_trap_plCommand_justified_thm This next theorem proves that if the PlatoonLeader issues a psgCommand then that command is trapped. It specializes the *TR_trap_cmd_rule* with two lemmas to help out.

[PlatoonLeader_trap_psgCommand_lemma]

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

[PlatoonLeader_trap_psgCommand_justified_lemma]

```

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

```

[PlatoonSergeant_trap_plCommand_justified_thm]

```

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

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

```

PlatoonLeader_WARNO_exec_report1_justified_thm This theorem proves that transition from the WARNO to the REPORT1 is justified if the following four statements are supplied.

- PlatoonLeader says tentativePlan
- PlatoonSergeant says initiateMovement
- PlatoonLeader says recon
- PlatoonLeader says report1

It specializes the *TR_exex_cmd_rule* and requires the standard two lemmas. They are shown below.

[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)))])

```

[PlatoonLeader_WARNO_exec_report1_justified_lemma]

```

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

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

```

```
Name PlatoonLeader says prop (SOME (SLc (PL report1))) ]
```

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

```

PlatoonSergeant_trap_plCommand_justified_thm This final theorem proves that the PlatoonSergeant is trapped on all plCommands. The reason should be obvious.

[PlatoonSergeant_trap_plCommand_lemma]

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

```

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

```

[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)
            ) ) ⇔
        authenticationTest inputOK
        [Name PlatoonSergeant says
      )
    )
  )

```

```

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

```

[PlatoonSergeant_trap_plCommand_justified_thm]

```

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

```

Chapter 8

Non-implemented modules

The previous chapter discusses the implementation of several patrol base operations implementations in HOL. But, there are numerous ways to describe the patrol base operations. This chapter briefly discusses ideas that are not implemented in this project.

8.1 Authentication & Roles

This master thesis discussed authentication as a visual confirmation among the soldiers. For example, most soldiers in a platoon know who their platoon leader is and do not require a password. But, there are scenarios where this is not true. In some cases, more stringent authentication is warranted. Such authentication may include the use of identification cards, password, biometric, implanted computer chips. The parametrizable secure state machine model used in this master thesis is also implemented as a crypto version. This version enforces the access-control logic using more stringent checks on authentication.

This master thesis also focuses on authentication and verification of roles. For example,

the PlatoonLeader and PlatoonSergeant issue commands. But, there is an alternative representation that considers soldiers acting in roles. For example, G.I. Jane acting in the role of Platoon Leader becomes the authenticated and authorized principal. Just as there is a parametrizable secure state machine model for crypto, there is one for principals acting in roles.

Both crypto and role versions of the parametrizable secure state machine add another level of complexity and flexibility to the model of the patrol base operations. They are not implemented in this master thesis because the used model seemed most relevant. Additionally, there were time constraints on this project. However, these versions of the ssms have been demonstrated in the "Assured Foundations" course at Syracuse University.

8.2 Soldier, Squad, and Platoon Theories

The patrol base operations model discussed in the previous chapters described transitions among phases of the operations. Additional models consider modules of soldiers, squads, and platoons. These are not implemented because of time constraints.

The soldier module would describe a soldier datatype. This would have various values such as name, ID, weapons, health, water, battleReadiness, assignment, etc. This could then prove various properties of a soldier. For example, *Soldier GI Jane says battleReady*, then the policy may state that

soldierHealthy andf

weaponsReady andf

waterReady andf

SoldierGIJane controls battleReady

From this, the ACL could prove that the soldier is battle ready if and only if the soldier is healthy and has her weapons and water. Similar modules could describe properties of a squad and platoon. These could be described with a secure state machine. However, they are simple enough to be described without it.

Chapter 9

Discussion

The previous chapters describe and discuss the work of this master thesis. This chapter summarizes the previous chapters and adds a few insights discovered along the way. It also discusses the scope of this work and any limitations or issues.

9.1 Summary

This master thesis begins with the systems security engineering framework not only because it presents the context for this work but also because it is helpful in guiding the work flow. This work begins with no assumptions regarding how to apply CSBD to the patrol base operations. The systems security engineering framework guides us to focus on unacceptable losses and security-sensitive objects. The former would become the escape level secure state machine. The later would become the phases of the operations.

The next chapter describes the Certified Security by Design with the access-control logic and its implementation in HOL. CSBD focuses the work on modeling the patrol base operations such that complete mediation can be verified and documented.

Verification is built into the secure state machine model. This model is taught in the "Assurance Foundations" course at Syracuse University. It has already been tested and implemented in HOL. To capitalize on this, the patrol base operations are modeled as secure state machines.

The next chapter describes the model of the patrol base operations. The patrol base operations are large. Thus, the model begins with a single, abstract, six-phase secure state machine that sequentially runs from the initial planning phase through to the completion phase of the operations. This becomes the top level of a hierarchy of secure state machines. Each level of the hierarchy descends into less abstract description of phases in the level above it. This approach has a couple of benefits. First, it is easier to model a large system if it is abstracted into levels and modularized. Second, it is more efficient to delegate work. Once the initial secure state machine is modeled, the CSBD expert focus on verification and documentation while the subject matter expert designs the next level of the model.

The next chapter, describes the secure state machine model and it's implementation in HOL. It discusses the concept of a monitor in terms of access-control. It also describes the parametrizable secure state machine (ssm) HOL implementation. All patrol base operations secure state machines use this ssm to verify and document properties of complete mediation.

The next chapter is the culmination of all the previous chapters. It describes several implementations of the patrol base operations in HOL. These implementation are examples that demonstrate the patterns of verification and documentation by way of formal proofs. These patterns are repeated throughout all the patrol base operations implementations.

The next chapter discusses ideas and models that are considered during the project but not implemented in HOL. These are worthy of implementation, but not implemented

because of the size of the system and finite amount of time.

The chapter following this one, discusses ideas on future work.

9.2 Challenges And Limitations of The Approach

The goal of this project is to determine if non-automated, human-centered systems are a limitation to the applicability of the CSBD approach. No limitations are found in this respect.

One challenge is to model a large system. This is solved using levels of abstraction and modularization described above. Once this is tackled, everything is straight forward ¹.

The main difficulty of CSBD is the learning curve in the mathematical application of formal methods to systems engineering. But, this is a general problem and not specific to this approach. In all areas of systems engineering, theorem proving remains a mostly interactive and not automated endeavor. While most mathematicians will appreciate the job security, most computer scientists would prefer to let the computer do the grunt work. HOL, as with most theorem provers, can be difficult to work with. However, it is a highly reliable theorem prover. Thus, when reliability is required, HOL shows promise.

The learning curve problem is partially solved by the use of pre-implemented and parametrizable models such as the secure state machine model. This combined with ample examples, eases the burden of reinventing HOL code. It is also likely, as theorem proving with ACL becomes more popular, more reusable and parametrizable code will be available to further ease the burden of proof.

¹The only exception was that the parametrizable secure state machine (ssm) needed to be remodeled half-way through the project. This meant that all the secure state machines had to be redone with the new model.

Chapter 10

Future Work

The previous chapters discussed the application of CSBD to a patrol base operations, This chapter touches briefly an other hon-automated human-centered systems where this work is applicable. It also touches briefly on the applicability of CSBD to a accountability systems.

10.1 Applicability

There are numerous non-automated, human-centered systems that require some form of security. For example, a building evacuation plan requires clear delineation of who controls a return to the building. Can anyone say "its safe to return to the building?" Or, must it be a fire fighter or police officer? On a larger scale, national disaster emergency management requires planning for leadership, acquisition of resources, and communication with the media. If a police officer calls the team leader and says he needs four more fire engines, should that request be granted? Does the police officer have the authority? These are issues that should be worked out during the disaster response planning phase.

The work in this master thesis demonstrates that a large non-automated, human-centered system can be designed with access-control in mind. But, this means that it can also be applied to smaller systems such as active-shooter responses. The use of a hierarchy of secure state machines need not be implemented for every phase of the system, but this design strategy allows the engineer to see the system in more detail. This detail can help highlight areas where access control is needed. With or without the formal verification and documentation, this approach is useful to designing more trustworthy systems with regards to access control.

10.1.1 Accountability Systems

The applicability of CSBD has already been demonstrated with automated systems. With this master thesis, it has been demonstrated on two extremes of the range of man-made systems. It is not too much of a leap to apply CSBD to a mixture of the two.

An idea that came up often during this work is the use of CSBD to designing accountability systems. For example, soldiers could have an application where they scan their equipment, enter their state of health, and provide other information. The application would then send a message to the platoon computer (or head quarters or the pentagon) that the soldier is battle ready. This requires access-control. Essentially, the application speaks for the soldier. The applications says *Application "quoting" SoldierGIJane says battleReady*. A policy would then include information describing the conditions whereby *SoldierGIJane controls battleReady*.

Furthermore, the application could track phases of the patrol base operations. This information could track personnel and equipment. Or, the state of the operations could be fed into a machine learning program that calculates strategies in combination with information from other operations- (a real-time mission analysis). This could be done using a variety of signals from the patrol base operations to indicate operational phase.

For example, radio confirmation indicating the mission is received would place the operations in the planning phase. The platoon leader could tap a link on his phone to indicate movement to the objective rally point. And so on. Such an accountability system will require some form of access control to verify that the right messages are coming from the right people. The method discussed in this master thesis is an effective way to do this.

Appendices

Appendix A

Access Control Logic Theories: Pretty-Printed Theories

Contents

1 aclfoundation Theory	3
1.1 Datatypes	3
1.2 Definitions	3
1.3 Theorems	4
2 aclsemantics Theory	6
2.1 Definitions	6
2.2 Theorems	8
3 aclrules Theory	10
3.1 Definitions	11
3.2 Theorems	11
4 aclDrules Theory	17
4.1 Theorems	17

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]

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

[intpKS_def]

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

[jKS_def]

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

[01_def]

$\vdash 01 = \text{PO one_weakorder}$

[one_weakorder_def]

$\vdash \forall x\ y.\ \text{one_weakorder } x\ y \iff \text{T}$

[po_TY_DEF]

$\vdash \exists rep.\ \text{TYPE_DEFINITION WeakOrder } rep$

[po_tybij]

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

[prod_PO_def]

$\vdash \forall PO_1\ PO_2.\ prod_{\text{PO}}\ PO_1\ PO_2 = \text{PO } (\text{RPROD } (\text{repPO } PO_1)\ (\text{repPO } PO_2))$

[smapKS_def]

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

[Subset_PO_def]

$\vdash \text{Subset_PO} = \text{PO } (\subseteq)$

1.3 Theorems

[abs_po11]

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

[absPO_fn_onto]

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

[antisym_prod_antisym]

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

[EQ_WeakOrder]

$\vdash \text{WeakOrder } (=)$

[KS_bij]

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

[one_weakorder_WO]

$\vdash \text{WeakOrder one_weakorder}$

[onto_po]

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

[po_bij]

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

[PO_repPO]

$\vdash \forall a. \text{PO } (\text{repPO } a) = a$

[refl_prod_refl]

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

[repPO_iPO_partial_order]

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

[repPO_01]

$\vdash \text{repPO } 01 = \text{one_weakorder}$

[repPO_prod_PO]

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

[repPO_Subset_PO]

$\vdash \text{repPO } \text{Subset_PO} = (\subseteq)$

[RPROD_THM]

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

[SUBSET_WO]

$\vdash \text{WeakOrder } (\subseteq)$

[trans_prod_trans]

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

[WeakOrder_Exists]

$\vdash \exists R. \text{WeakOrder } R$

[WO_prod_WO]

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

[WO_repPO]

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

2 aclsemantics Theory

Built: 25 February 2018

Parent Theories: aclfoundation

2.1 Definitions

[Efn_def]

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

```

if Jext (jKS M) Q RSUBSET Jext (jKS M) P then U(:'v)
else {}  $\wedge$ 
( $\forall Oi Os M P f.$ 
 Efn Oi Os M (P controls f) =
 U(:'v) DIFF {w | 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) =
 U(:'v) DIFF
 {w | Jext (jKS M) (P quoting Q) w  $\subseteq$  Efn Oi Os M f}  $\cup$ 
 {w | Jext (jKS M) Q w  $\subseteq$  Efn Oi Os M f})  $\wedge$ 
( $\forall Oi Os M intl_1 intl_2.$ 
 Efn Oi Os M (intl_1 domi intl_2) =
 if repPO Oi (Lifn M intl_2) (Lifn M intl_1) then 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 U(:'v)
 else {})  $\cap$ 
 if repPO Oi (Lifn M intl_1) (Lifn M intl_2) then 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 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 U(:'v)
 else {})  $\cap$ 
 if repPO Os (Lsfn M secl_1) (Lsfn M secl_2) then 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 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 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 U(:'v) else {}

```

[Jext_def]

```

 $\vdash (\forall J s. \text{Jext } J (\text{Name } s) = J s) \wedge$ 
 $(\forall J P_1 P_2.$ 
 $\text{Jext } J (P_1 \text{ meet } P_2) = \text{Jext } J P_1 \text{ RUNION } \text{Jext } J P_2) \wedge$ 
 $\forall J P_1 P_2. \text{Jext } J (P_1 \text{ quoting } P_2) = \text{Jext } J P_2 \text{ O } \text{Jext } J P_1$ 

```

[Lifn_def]

```

 $\vdash (\forall M l. \text{Lifn } M (\text{iLab } l) = l) \wedge$ 
 $\forall M name. \text{Lifn } M (\text{il } name) = \text{imapKS } M name$ 

```

[**Lsfn_def**]

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

2.2 Theorems

[**andf_def**]

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

[**controls_def**]

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

[**controls_says**]

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

[**domi_def**]

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

[**doms_def**]

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

[**eqf_def**]

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

[**eqf_impf**]

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

[eqi_def]

$\vdash \forall Oi Os M \ intL_2 \ intL_1 .$
 $\text{Efn } Oi Os M (\intL_2 \text{ eqi } \intL_1) =$
 $(\text{if repP0 } Oi (\text{Lifn } M \ intL_2) (\text{Lifn } M \ intL_1) \text{ then } \mathcal{U}(:'v)$
 $\text{else } \{ \}) \cap$
 $\text{if repP0 } Oi (\text{Lifn } M \ intL_1) (\text{Lifn } M \ intL_2) \text{ then } \mathcal{U}(:'v)$
 $\text{else } \{ \}$

[eqi_domi]

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

[eqn_def]

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

[eqs_def]

$\vdash \forall Oi Os M \ secl_2 \ secl_1 .$
 $\text{Efn } Oi Os M (\secl_2 \text{ eqs } \secl_1) =$
 $(\text{if repP0 } Os (\text{Lsfn } M \ secl_2) (\text{Lsfn } M \ secl_1) \text{ then } \mathcal{U}(:'v)$
 $\text{else } \{ \}) \cap$
 $\text{if repP0 } Os (\text{Lsfn } M \ secl_1) (\text{Lsfn } M \ secl_2) \text{ then } \mathcal{U}(:'v)$
 $\text{else } \{ \}$

[eqs_doms]

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

[FF_def]

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

[impf_def]

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

[lt_def]

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

[lte_def]

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

[meet_def]

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

[name_def]

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

[notf_def]

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

[orf_def]

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

[prop_def]

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

[quoting_def]

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

[reps_def]

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

[says_def]

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

[speaks_for_def]

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

[TT_def]

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

3 aclrules Theory

Built: 25 February 2018

Parent Theories: aclsemantics

3.1 Definitions

[sat_def]

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

3.2 Theorems

[And_Says]

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

[And_Says_Eq]

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

[and_says_lemma]

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

[Controls_Eq]

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

[DIFF_UNIV_SUBSET]

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

[domi_antisymmetric]

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

[domi_reflexive]

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

[domi_transitive]

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

[doms_antisymmetric]

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

[doms_reflexive]

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

[doms_transitive]

$$\begin{aligned} &\vdash \forall M \text{ } Oi \text{ } Os \text{ } l_1 \text{ } l_2 \text{ } l_3. \\ &\quad (M, Oi, Os) \text{ sat } l_1 \text{ doms } l_2 \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } l_2 \text{ doms } l_3 \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } l_1 \text{ doms } l_3 \end{aligned}$$

[eqf_and_impf]

$$\begin{aligned} &\vdash \forall M \text{ } Oi \text{ } Os \text{ } f_1 \text{ } f_2. \\ &\quad (M, Oi, Os) \text{ sat } f_1 \text{ eqf } f_2 \iff \\ &\quad (M, Oi, Os) \text{ sat } (f_1 \text{ impf } f_2) \text{ andf } (f_2 \text{ impf } f_1) \end{aligned}$$

[eqf_andf1]

$$\begin{aligned} &\vdash \forall M \text{ } Oi \text{ } Os \text{ } f \text{ } f' \text{ } g. \\ &\quad (M, Oi, Os) \text{ sat } f \text{ eqf } f' \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } f \text{ andf } g \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } f' \text{ andf } g \end{aligned}$$

[eqf_andf2]

$$\begin{aligned} &\vdash \forall M \text{ } Oi \text{ } Os \text{ } f \text{ } f' \text{ } g. \\ &\quad (M, Oi, Os) \text{ sat } f \text{ eqf } f' \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } g \text{ andf } f \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } g \text{ andf } f' \end{aligned}$$

[eqf_controls]

$$\begin{aligned} &\vdash \forall M \text{ } Oi \text{ } Os \text{ } P \text{ } f \text{ } f'. \\ &\quad (M, Oi, Os) \text{ sat } f \text{ eqf } f' \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } P \text{ controls } f \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } P \text{ controls } f' \end{aligned}$$

[eqf_eq]

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

[eqf_eqf1]

$$\begin{aligned} &\vdash \forall M \text{ } Oi \text{ } Os \text{ } f \text{ } f' \text{ } g. \\ &\quad (M, Oi, Os) \text{ sat } f \text{ eqf } f' \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } f \text{ eqf } g \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } f' \text{ eqf } g \end{aligned}$$

[eqf_eqf2]

$$\begin{aligned} &\vdash \forall M \text{ } Oi \text{ } Os \text{ } f \text{ } f' \text{ } g. \\ &\quad (M, Oi, Os) \text{ sat } f \text{ eqf } f' \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } g \text{ eqf } f \Rightarrow \\ &\quad (M, Oi, Os) \text{ sat } g \text{ eqf } f' \end{aligned}$$

[eqf_impf1]

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

[eqf_impf2]

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

[eqf_notf]

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

[eqf_orf1]

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

[eqf_orf2]

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

[eqf_reps]

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

[eqf_sat]

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

[eqf_says]

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

[eqi_Eq]

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

[eqs_Eq]

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

[Idemp_Speaks_For]

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

[Image_cmp]

$$\vdash \forall R_1 \ R_2 \ R_3 \ u. \ (R_1 \ 0 \ R_2) \ u \subseteq R_3 \iff R_2 \ u \subseteq \{y \mid R_1 \ y \subseteq R_3\}$$

[Image_SUBSET]

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

[Image_UNION]

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

[INTER_EQ_UNIV]

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

[Modus_Ponens]

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

[Mono_speaks_for]

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

[MP_Says]

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

[Quoting]

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

[Quoting_Eq]

$$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$$

$$(M, Oi, Os) \text{ sat } P \text{ quoting } Q \text{ says } f \iff$$

$$(M, Oi, Os) \text{ sat } P \text{ says } Q \text{ says } f$$

[reps_def_lemma]

$$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$$

$$\text{Efn } Oi \ Os \ M \ (\text{reps } P \ Q \ f) =$$

$$\text{Efn } Oi \ Os \ M \ (P \text{ quoting } Q \text{ says } f \text{ impf } Q \text{ says } f)$$

[Reps_Eq]

$$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$$

$$(M, Oi, Os) \text{ sat } \text{reps } P \ Q \ f \iff$$

$$(M, Oi, Os) \text{ sat } P \text{ quoting } Q \text{ says } f \text{ impf } Q \text{ says } f$$

[sat_allworld]

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

[sat_andf_eq_and_sat]

$$\vdash (M, Oi, Os) \text{ sat } f_1 \text{ andf } f_2 \iff$$

$$(M, Oi, Os) \text{ sat } f_1 \wedge (M, Oi, Os) \text{ sat } f_2$$

[sat_TT]

$$\vdash (M, Oi, Os) \text{ sat } \text{TT}$$

[Says]

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

[says_and_lemma]

$$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$$

$$(M, Oi, Os) \text{ sat } P \text{ says } f \text{ andf } Q \text{ says } f \text{ impf } P \text{ meet } Q \text{ says } f$$

[Speaks_For]

$$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$$

$$(M, Oi, Os) \text{ sat } P \text{ speaks_for } Q \text{ impf } P \text{ says } f \text{ impf } Q \text{ says } f$$

[speaks_for_SUBSET]

$$\vdash \forall R_3 \ R_2 \ R_1.$$

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

[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]

$$\vdash \forall M \ Oi \ Os \ P \ Q \ R. \\ (M, Oi, Os) \text{ sat } P \text{ speaks_for } Q \Rightarrow \\ (M, Oi, Os) \text{ sat } Q \text{ speaks_for } R \Rightarrow \\ (M, Oi, Os) \text{ sat } P \text{ speaks_for } R$$

[UNIV_DIFF_SUBSET]

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

[world_and]

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

[world_eq]

$$\vdash \forall M \ Oi \ Os \ f_1 \ f_2 \ w. \\ w \in \text{Efn } Oi \ Os \ M \ (f_1 \text{ eqf } f_2) \iff \\ (w \in \text{Efn } Oi \ Os \ M \ f_1 \iff w \in \text{Efn } Oi \ Os \ M \ f_2)$$

[world_eqn]

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

[world_F]

$$\vdash \forall M \ Oi \ Os \ w. \ w \notin \text{Efn } Oi \ Os \ M \text{ FF}$$

[world_imp]

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

[world_lt]

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

[world_lte]

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

[world_not]

$$\vdash \forall M \ Oi \ Os \ f \ w. \ w \in \text{Efn } Oi \ Os \ M \ (\text{notf } f) \iff w \notin \text{Efn } Oi \ Os \ M \ f$$

[world_or]

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

[world_says]

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

[world_T]

$$\vdash \forall M \ Oi \ Os \ w. \ w \in \text{Efn } Oi \ Os \ M \text{ TT}$$

4 aclDrules Theory

Built: 25 February 2018

Parent Theories: aclrules

4.1 Theorems

[Conjunction]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ f_1 \ f_2. \\ (M, Oi, Os) \text{ sat } f_1 \Rightarrow \\ (M, Oi, Os) \text{ sat } f_2 \Rightarrow \\ (M, Oi, Os) \text{ sat } f_1 \text{ andf } f_2 \end{aligned}$$

[Controls]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ P \ f. \\ (M, Oi, Os) \text{ sat } P \text{ says } f \Rightarrow \\ (M, Oi, Os) \text{ sat } P \text{ controls } f \Rightarrow \\ (M, Oi, Os) \text{ sat } f \end{aligned}$$

[Derived_Controls]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ P \ Q \ f. \\ (M, Oi, Os) \text{ sat } P \text{ speaks_for } Q \Rightarrow \\ (M, Oi, Os) \text{ sat } Q \text{ controls } f \Rightarrow \\ (M, Oi, Os) \text{ sat } P \text{ controls } f \end{aligned}$$

[Derived_Speaks_For]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ P \ Q \ f. \\ (M, Oi, Os) \text{ sat } P \text{ speaks_for } Q \Rightarrow \\ (M, Oi, Os) \text{ sat } P \text{ says } f \Rightarrow \\ (M, Oi, Os) \text{ sat } Q \text{ says } f \end{aligned}$$

[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]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ f_1 \ f_2. \\ (M, Oi, Os) \text{ sat } f_1 \text{ orf } f_2 \Rightarrow \\ (M, Oi, Os) \text{ sat } \text{notf } f_1 \Rightarrow \\ (M, Oi, Os) \text{ sat } f_2 \end{aligned}$$

[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]

$$\begin{aligned} \vdash (M, Oi, Os) \text{ sat } c_1 \text{ eqn } n_1 &\Rightarrow \\ (M, Oi, Os) \text{ sat } c_2 \text{ eqn } n_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } n_1 \text{ eqn } n_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } c_1 \text{ eqn } c_2 & \end{aligned}$$

[eqn_lt]

$$\begin{aligned} \vdash (M, Oi, Os) \text{ sat } c_1 \text{ eqn } n_1 &\Rightarrow \\ (M, Oi, Os) \text{ sat } c_2 \text{ eqn } n_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } n_1 \text{ lt } n_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } c_1 \text{ lt } c_2 & \end{aligned}$$

[eqn_lte]

$$\begin{aligned} \vdash (M, Oi, Os) \text{ sat } c_1 \text{ eqn } n_1 &\Rightarrow \\ (M, Oi, Os) \text{ sat } c_2 \text{ eqn } n_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } n_1 \text{ lte } n_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } c_1 \text{ lte } c_2 & \end{aligned}$$

[Hypothetical_Syllogism]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ f_1 \ f_2 \ f_3. \\ (M, Oi, Os) \text{ sat } f_1 \text{ impf } f_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } f_2 \text{ impf } f_3 &\Rightarrow \\ (M, Oi, Os) \text{ sat } f_1 \text{ impf } f_3 & \end{aligned}$$

[il_domi]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ P \ Q \ l_1 \ l_2. \\ (M, Oi, Os) \text{ sat } il \ P \ eqi \ l_1 &\Rightarrow \\ (M, Oi, Os) \text{ sat } il \ Q \ eqi \ l_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } l_2 \ domi \ l_1 &\Rightarrow \\ (M, Oi, Os) \text{ sat } il \ Q \ domi \ il \ P & \end{aligned}$$

[INTER_EQ_UNIV]

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

[Modus_Tollens]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ f_1 \ f_2. \\ (M, Oi, Os) \text{ sat } f_1 \text{ impf } f_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } notf \ f_2 &\Rightarrow \\ (M, Oi, Os) \text{ sat } notf \ f_1 & \end{aligned}$$

[Rep_Controls_Eq]

$$\begin{aligned} \vdash \forall M \ Oi \ Os \ A \ B \ f. \\ (M, Oi, Os) \text{ sat } reps \ A \ B \ f &\iff \\ (M, Oi, Os) \text{ sat } A \text{ controls } B \text{ says } f & \end{aligned}$$

[Rep_Says]

$$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$$

$$(M, Oi, Os) \text{ sat } \text{reps } P \ Q \ f \Rightarrow$$

$$(M, Oi, Os) \text{ sat } P \text{ quoting } Q \text{ says } f \Rightarrow$$

$$(M, Oi, Os) \text{ sat } Q \text{ says } f$$

[Reps]

$$\vdash \forall M \ Oi \ Os \ P \ Q \ f.$$

$$(M, Oi, Os) \text{ sat } \text{reps } P \ Q \ f \Rightarrow$$

$$(M, Oi, Os) \text{ sat } P \text{ quoting } Q \text{ says } f \Rightarrow$$

$$(M, Oi, Os) \text{ sat } Q \text{ controls } f \Rightarrow$$

$$(M, Oi, Os) \text{ sat } f$$

[Says_Simplification1]

$$\vdash \forall M \ Oi \ Os \ P \ f_1 \ f_2.$$

$$(M, Oi, Os) \text{ sat } P \text{ says } (f_1 \text{ andf } f_2) \Rightarrow (M, Oi, Os) \text{ sat } P \text{ says } f_1$$

[Says_Simplification2]

$$\vdash \forall M \ Oi \ Os \ P \ f_1 \ f_2.$$

$$(M, Oi, Os) \text{ sat } P \text{ says } (f_1 \text{ andf } f_2) \Rightarrow (M, Oi, Os) \text{ sat } P \text{ says } f_2$$

[Simplification1]

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

[Simplification2]

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

[sl_doms]

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

$$(M, Oi, Os) \text{ sat } \text{sl } P \text{ eqs } l_1 \Rightarrow$$

$$(M, Oi, Os) \text{ sat } \text{sl } Q \text{ eqs } l_2 \Rightarrow$$

$$(M, Oi, Os) \text{ sat } l_2 \text{ doms } l_1 \Rightarrow$$

$$(M, Oi, Os) \text{ sat } \text{sl } Q \text{ doms sl } P$$

Index

aclDrules Theory, 17
Theorems, 17
Conjunction, 17
Controls, 17
Derived_Controls, 17
Derived_Speaks_For, 17
Disjunction1, 17
Disjunction2, 17
Disjunctive_Syllogism, 17
Double_Negation, 17
eqn_eqn, 18
eqn_lt, 18
eqn_lte, 18
Hypothetical_Syllogism, 18
il_domi, 18
INTER_EQ_UNIV, 18
Modus_Tollens, 18
Rep_Controls_Eq, 18
Rep_Says, 19
Reps, 19
Says_Simplification1, 19
Says_Simplification2, 19
Simplification1, 19
Simplification2, 19
sl_doms, 19

aclfoundation Theory, 3
Datatypes, 3
Definitions, 3
imapKS_def, 4
intpKS_def, 4
jKS_def, 4
O1_def, 4
one_weakorder_def, 4
po_TY_DEF, 4
po_tybij, 4
prod.PO_def, 4
smapKS_def, 4
Subset.PO_def, 4
Theorems, 4
abs_po11, 4

absPO_fn_onto, 4
antisym_prod_antisym, 5
EQ_WeakOrder, 5
KS_bij, 5
one_weakorder_WO, 5
onto_po, 5
po_bij, 5
PO_repPO, 5
refl_prod_refl, 5
repPO_iPO_partial_order, 5
repPO_O1, 5
repPO_prod_PO, 5
repPO_Subset_PO, 5
RPROD_THM, 5
SUBSET_WO, 6
trans_prod_trans, 6
WeakOrder_Exists, 6
WO_prod_WO, 6
WO_repPO, 6

aclrules Theory, 10
Definitions, 11
sat_def, 11
Theorems, 11
And_Says, 11
And_Says_Eq, 11
and_says_lemma, 11
Controls_Eq, 11
DIFF_UNIV_SUBSET, 11
domi_antisymmetric, 11
domi_reflexive, 11
domi_transitive, 11
doms_antisymmetric, 11
doms_reflexive, 12
doms_transitive, 12
eqf_and_impf, 12
eqf_andf1, 12
eqf_andf2, 12
eqf_controls, 12
eqf_eq, 12
eqf_eqf1, 12

eqf_eqf2, 12
 eqf_impf1, 13
 eqf_impf2, 13
 eqf_notf, 13
 eqf_orf1, 13
 eqf_orf2, 13
 eqf_reps, 13
 eqf_sat, 13
 eqf_says, 13
 equi_Eq, 14
 eqs_Eq, 14
 Idemp_Speaks_For, 14
 Image_cmp, 14
 Image_SUBSET, 14
 Image_UNION, 14
 INTER_EQ_UNIV, 14
 Modus_Ponens, 14
 Mono_speaks_for, 14
 MP_Says, 14
 Quoting, 14
 Quoting_Eq, 15
 reps_def_lemma, 15
 Reps_Eq, 15
 sat_allworld, 15
 sat_andf_eq_and_sat, 15
 sat_TT, 15
 Says, 15
 says_and_lemma, 15
 Speaks_For, 15
 speaks_for_SUBSET, 15
 SUBSET_Image_SUBSET, 15
 Trans_Speaks_For, 16
 UNIV_DIFF_SUBSET, 16
 world_and, 16
 world_eq, 16
 world_eqn, 16
 world_F, 16
 world_imp, 16
 world_lt, 16
 world_lte, 16
 world_not, 16
 world_or, 16
 world_says, 16
 world_T, 16
aclsemantics Theory, 6
 Definitions, 6
 Efn_def, 6
 Jext_def, 7
 Lifn_def, 7
 Lsfn_def, 8
 Theorems, 8
 andf_def, 8
 controls_def, 8
 controls_says, 8
 domi_def, 8
 doms_def, 8
 eqf_def, 8
 eqf_impf, 8
 equi_def, 9
 equi_domi, 9
 eqn_def, 9
 eqs_def, 9
 eqs_doms, 9
 FF_def, 9
 impf_def, 9
 lt_def, 9
 lte_def, 9
 meet_def, 10
 name_def, 10
 notf_def, 10
 orf_def, 10
 prop_def, 10
 quoting_def, 10
 reps_def, 10
 says_def, 10
 speaks_for_def, 10
 TT_def, 10

Appendix B

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

Contents

1 OMNIType Theory	3
1.1 Datatypes	3
1.2 Theorems	3
2 ssm11 Theory	4
2.1 Datatypes	4
2.2 Definitions	4
2.3 Theorems	5
3 ssm Theory	11
3.1 Datatypes	11
3.2 Definitions	12
3.3 Theorems	13
4 satList Theory	21
4.1 Definitions	21
4.2 Theorems	21
5 PBTypeIntegrated Theory	21
5.1 Datatypes	21
5.2 Theorems	22
6 PBIntegratedDef Theory	23
6.1 Definitions	23
6.2 Theorems	24
7 ssmPBIntegrated Theory	28
7.1 Theorems	28
8 ssmConductORP Theory	35
8.1 Theorems	35
9 ConductORPType Theory	44
9.1 Datatypes	44
9.2 Theorems	45
10 ConductORPDef Theory	46
10.1 Definitions	46
10.2 Theorems	47
11 ssmConductPB Theory	51
11.1 Definitions	51
11.2 Theorems	52

12 ConductPBType Theory	57
12.1 Datatypes	57
12.2 Theorems	57
13 ssmMoveToORP Theory	58
13.1 Definitions	58
13.2 Theorems	58
14 MoveToORPType Theory	62
14.1 Datatypes	62
14.2 Theorems	63
15 ssmMoveToPB Theory	63
15.1 Definitions	63
15.2 Theorems	64
16 MoveToPBType Theory	68
16.1 Datatypes	68
16.2 Theorems	68
17 ssmPlanPB Theory	69
17.1 Theorems	69
18 PlanPBType Theory	79
18.1 Datatypes	79
18.2 Theorems	79
19 PlanPBDef Theory	82
19.1 Definitions	82
19.2 Theorems	83

1 OMNIType Theory

Built: 10 June 2018

Parent Theories: indexedLists, patternMatches

1.1 Datatypes

```
command = ESCc escCommand | SLc 'slCommand

escCommand = returnToBase | changeMission | resupply
| reactToContact

escOutput = ReturnToBase | ChangeMission | Resupply
| ReactToContact

escState = RTB | CM | RESUPPLY | RTC

output = ESCo escOutput | SLo 'slOutput

principal = SR 'stateRole

state = ESCs escState | SLs 'slState
```

1.2 Theorems

[`command_distinct_clauses`]

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

[`command_one_one`]

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

[`escCommand_distinct_clauses`]

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

[`escOutput_distinct_clauses`]

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

[`escState_distinct_clauses`]

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

[output_distinct_clauses]

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

[output_one_one]

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

[principal_one_one]

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

[state_distinct_clauses]

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

[state_one_one]

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

2 ssm11 Theory

Built: 10 June 2018

Parent Theories: satList

2.1 Datatypes

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

order = SOME 'command | NONE

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

2.2 Definitions

[TR_def]

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

```

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

```

2.3 Theorems

[CFGInterpret_def]

```

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

```

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

[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]

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

[order_one_one]

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

[TR_cases]

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

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

[TR_discard_cmd_rule]

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

[TR_EQ_rules_thm]

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

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

[TR_exec_cmd_rule]

$$\vdash \forall \text{authenticationTest} \text{ securityContext} \text{ stateInterp } P \text{ cmd ins } s \text{ outs.}$$

$$(\forall M \text{ Oi Os.}$$

$$\text{CFGInterpret } (M, Oi, Os)$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$(P \text{ says prop (SOME cmd)::ins}) \ s \text{ outs}) \Rightarrow$$

$$(M, Oi, Os) \text{ sat prop (SOME cmd)} \Rightarrow$$

$$\forall NS \text{ Out } M \text{ Oi Os.}$$

$$\text{TR } (M, Oi, Os) \text{ (exec cmd)}$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$(P \text{ says prop (SOME cmd)::ins}) \ s \text{ outs})$$

$$(\text{CFG authenticationTest stateInterp securityContext ins}$$

$$(NS\ s\ (\text{exec cmd}))\ (\text{Out}\ s\ (\text{exec cmd})::outs) \iff \text{authenticationTest}\ (P \text{ says prop (SOME cmd)}) \wedge$$

$$\text{CFGInterpret } (M, Oi, Os)$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$(P \text{ says prop (SOME cmd)::ins}) \ s \text{ outs}) \wedge$$

$$(M, Oi, Os) \text{ sat prop (SOME cmd)}$$

[TR_ind]

$$\vdash \forall TR'.$$

$$(\forall \text{authenticationTest } P \text{ NS } M \text{ Oi Os Out } s \text{ securityContext}$$

$$\text{stateInterp cmd ins outs.}$$

$$\text{authenticationTest } (P \text{ says prop (SOME cmd)}) \wedge$$

$$\text{CFGInterpret } (M, Oi, Os)$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$(P \text{ says prop (SOME cmd)::ins}) \ s \text{ outs}) \Rightarrow$$

$$TR' \text{ (M, Oi, Os) (exec cmd)}$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$(P \text{ says prop (SOME cmd)::ins}) \ s \text{ outs})$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$\text{ins } (NS\ s\ (\text{exec cmd}))\ (\text{Out}\ s\ (\text{exec cmd})::outs))) \wedge$$

$$(\forall \text{authenticationTest } P \text{ NS } M \text{ Oi Os Out } s \text{ securityContext}$$

$$\text{stateInterp cmd ins outs.}$$

$$\text{authenticationTest } (P \text{ says prop (SOME cmd)}) \wedge$$

$$\text{CFGInterpret } (M, Oi, Os)$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$(P \text{ says prop (SOME cmd)::ins}) \ s \text{ outs}) \Rightarrow$$

$$TR' \text{ (M, Oi, Os) (trap cmd)}$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$(P \text{ says prop (SOME cmd)::ins}) \ s \text{ outs})$$

$$(\text{CFG authenticationTest stateInterp securityContext}$$

$$\text{ins } (NS\ s\ (\text{trap cmd}))\ (\text{Out}\ s\ (\text{trap cmd})::outs))) \wedge$$

$$(\forall \text{authenticationTest } NS \text{ M } Oi \text{ Os Out } s \text{ securityContext}$$

$$\text{stateInterp cmd } x \text{ ins outs.}$$

```

¬authenticationTest x ⇒
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))) ⇒
∀ a0 a1 a2 a3. TR a0 a1 a2 a3 ⇒ TR' a0 a1 a2 a3

```

[TR_rules]

```

⊢ ( ∀ authenticationTest P NS M Oi Os Out s securityContext
    stateInterp cmd ins outs .
    authenticationTest (P says prop (SOME cmd)) ∧
    CFGInterpret (M, Oi, Os)
    (CFG authenticationTest stateInterp securityContext
     (P says prop (SOME cmd)::ins) s outs) ⇒
    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))) ∧
( ∀ authenticationTest P NS M Oi Os Out s securityContext
    stateInterp cmd ins outs .
    authenticationTest (P says prop (SOME cmd)) ∧
    CFGInterpret (M, Oi, Os)
    (CFG authenticationTest stateInterp securityContext
     (P says prop (SOME cmd)::ins) s outs) ⇒
    TR (M, Oi, Os) (trap cmd)
    (CFG authenticationTest stateInterp securityContext
     (P says prop (SOME cmd)::ins) s outs)
    (CFG authenticationTest stateInterp securityContext ins
     (NS s (trap cmd)) (Out s (trap cmd)::outs))) ∧
 ∀ authenticationTest NS M Oi Os Out s securityContext
    stateInterp cmd x ins outs .
    ¬authenticationTest x ⇒
    TR (M, Oi, Os) (discard cmd)
    (CFG authenticationTest stateInterp securityContext
     (x::ins) s outs)
    (CFG authenticationTest stateInterp securityContext ins
     (NS s (discard cmd)) (Out s (discard cmd)::outs)))

```

[TR_strongind]

```

⊢ ∀ TR'.
( ∀ authenticationTest P NS M Oi Os Out s securityContext
    stateInterp cmd ins outs .
    authenticationTest (P says prop (SOME cmd)) ∧
    CFGInterpret (M, Oi, Os)
    (CFG authenticationTest stateInterp securityContext
     (P says prop (SOME cmd)::ins) s outs) ⇒

```

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

[TR_trap_cmd_rule]

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

[TRrule0]

$\vdash \text{TR (M, Oi, Os) (exec cmd)}$
 $(\text{CFG authenticationTest stateInterp securityContext}$

```

(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: 10 June 2018

Parent Theories: satList

3.1 Datatypes

```

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

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

```

3.2 Definitions

[authenticationTest_def]

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

[commandList_def]

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

[inputList_def]

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

[propCommandList_def]

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

[TR_def]

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

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

3.3 Theorems

[CFGInterpret_def]

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

[CFGInterpret_ind]

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

[configuration_one_one]

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

[extractCommand_def]

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

[extractCommand_ind]

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

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

[`extractInput_def`]

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

[`extractInput_ind`]

$\vdash \forall P'.$

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

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

[extractPropCommand_def]

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

[extractPropCommand_ind]

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

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

[TR_cases]

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

```

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

```

[TR_discard_cmd_rule]

```

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

```

[TR_EQ_rules_thm]

```

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

```

[TR_exec_cmd_rule]

```

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

```

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

[TR_ind]

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

[TR_rules]

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

```

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

```

[TR_strongind]

```

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

```

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

[TR_trap_cmd_rule]

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

[TRrule0]

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

[TRrule1]

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

[trType_distinct_clauses]

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

[trType_one_one]

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

4 satList Theory

Built: 10 June 2018

Parent Theories: aclDrules

4.1 Definitions

[satList_def]

$$\vdash \forall M Oi Os \text{ formList}. \\ (M, Oi, Os) \text{ satList formList} \iff \\ \text{FOLDL} (\lambda x y. x \wedge y) \text{ T } (\text{MAP} (\lambda f. (M, Oi, Os) \text{ sat } f) \text{ formList})$$

4.2 Theorems

[satList_conj]

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

[satList_CONS]

$$\vdash \forall h t M Oi Os. \\ (M, Oi, Os) \text{ satList } (h :: t) \iff \\ (M, Oi, Os) \text{ sat } h \wedge (M, Oi, Os) \text{ satList } t$$

[satList_nil]

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

5 PBTypeIntegrated Theory

Built: 11 June 2018

Parent Theories: OMNIType

5.1 Datatypes

$$\begin{aligned} \text{omniCommand} &= \text{ssmPlanPBComplete} \mid \text{ssmMoveToORPComplete} \\ &\quad \mid \text{ssmConductORPComplete} \mid \text{ssmMoveToPBComplete} \\ &\quad \mid \text{ssmConductPBComplete} \mid \text{invalidOmniCommand} \end{aligned}$$

$$\begin{aligned} \text{plCommand} &= \text{crossLD} \mid \text{conductORP} \mid \text{moveToPB} \mid \text{conductPB} \\ &\quad \mid \text{completePB} \mid \text{incomplete} \end{aligned}$$

```

 $slCommand =$ 
  PL PBTypeIntegrated$plCommand
  | OMNI PBTypeIntegrated$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

```

5.2 Theorems

[omniCommand_distinct_clauses]

```

 $\vdash ssmPlanPBComplete \neq ssmMoveToORPComplete \wedge$ 
 $ssmPlanPBComplete \neq ssmConductORPComplete \wedge$ 
 $ssmPlanPBComplete \neq ssmMoveToPBComplete \wedge$ 
 $ssmPlanPBComplete \neq ssmConductPBComplete \wedge$ 
 $ssmPlanPBComplete \neq invalidOmniCommand \wedge$ 
 $ssmMoveToORPComplete \neq ssmConductORPComplete \wedge$ 
 $ssmMoveToORPComplete \neq ssmMoveToPBComplete \wedge$ 
 $ssmMoveToORPComplete \neq ssmConductPBComplete \wedge$ 
 $ssmMoveToORPComplete \neq invalidOmniCommand \wedge$ 
 $ssmConductORPComplete \neq ssmMoveToPBComplete \wedge$ 
 $ssmConductORPComplete \neq ssmConductPBComplete \wedge$ 
 $ssmConductORPComplete \neq invalidOmniCommand \wedge$ 
 $ssmMoveToPBComplete \neq ssmConductPBComplete \wedge$ 
 $ssmMoveToPBComplete \neq invalidOmniCommand \wedge$ 
 $ssmConductPBComplete \neq invalidOmniCommand$ 

```

[plCommand_distinct_clauses]

```

 $\vdash crossLD \neq conductORP \wedge crossLD \neq moveToPB \wedge$ 
 $crossLD \neq conductPB \wedge crossLD \neq completePB \wedge$ 
 $crossLD \neq incomplete \wedge conductORP \neq moveToPB \wedge$ 
 $conductORP \neq conductPB \wedge conductORP \neq completePB \wedge$ 
 $conductORP \neq incomplete \wedge moveToPB \neq conductPB \wedge$ 
 $moveToPB \neq completePB \wedge moveToPB \neq incomplete \wedge$ 
 $conductPB \neq completePB \wedge conductPB \neq incomplete \wedge$ 
 $completePB \neq incomplete$ 

```

[slCommand_distinct_clauses]

```

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

```

[slCommand_one_one]

```

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

```

[s1Output_distinct_clauses]

```

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

```

[s1State_distinct_clauses]

```

 $\vdash \text{PLAN\_PB} \neq \text{MOVE\_TO\_ORP} \wedge \text{PLAN\_PB} \neq \text{CONDUCT\_ORP} \wedge$ 
 $\text{PLAN\_PB} \neq \text{MOVE\_TO\_PB} \wedge \text{PLAN\_PB} \neq \text{CONDUCT\_PB} \wedge$ 
 $\text{PLAN\_PB} \neq \text{COMPLETE\_PB} \wedge \text{MOVE\_TO\_ORP} \neq \text{CONDUCT\_ORP} \wedge$ 
 $\text{MOVE\_TO\_ORP} \neq \text{MOVE\_TO\_PB} \wedge \text{MOVE\_TO\_ORP} \neq \text{CONDUCT\_PB} \wedge$ 
 $\text{MOVE\_TO\_ORP} \neq \text{COMPLETE\_PB} \wedge \text{CONDUCT\_ORP} \neq \text{MOVE\_TO\_PB} \wedge$ 
 $\text{CONDUCT\_ORP} \neq \text{CONDUCT\_PB} \wedge \text{CONDUCT\_ORP} \neq \text{COMPLETE\_PB} \wedge$ 
 $\text{MOVE\_TO\_PB} \neq \text{CONDUCT\_PB} \wedge \text{MOVE\_TO\_PB} \neq \text{COMPLETE\_PB} \wedge$ 
 $\text{CONDUCT\_PB} \neq \text{COMPLETE\_PB}$ 

```

[stateRole_distinct_clauses]

```

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

```

6 PBIntegratedDef Theory

Built: 11 June 2018

Parent Theories: PBTTypeIntegrated, aclfoundation

6.1 Definitions

[secAuthorization_def]

```

 $\vdash \forall xs. \text{secAuthorization } xs = \text{secHelper} (\text{getOmniCommand } xs)$ 

```

[secContext_def]

```

 $\vdash (\forall xs.$ 
 $\text{secContext PLAN\_PB } xs =$ 
 $\text{if getOmniCommand } xs = \text{ssmPlanPBComplete} \text{ then}$ 
 $\quad [\text{prop (SOME (SLc (OMNI ssmPlanPBComplete))) impf}$ 
 $\quad \text{Name PlatoonLeader controls}$ 
 $\quad \text{prop (SOME (SLc (PL crossLD)))}]$ 

```

```

else [prop NONE])  $\wedge$ 
 $(\forall xs.$ 
  secContext MOVE_TO_ORP  $xs =$ 
  if getOmniCommand  $xs =$  ssmMoveToORPComplete then
    [prop (SOME (SLc (OMNI ssmMoveToORPComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL conductORP)))]
  else [prop NONE])  $\wedge$ 
 $(\forall xs.$ 
  secContext CONDUCT_ORP  $xs =$ 
  if getOmniCommand  $xs =$  ssmConductORPComplete then
    [prop (SOME (SLc (OMNI ssmConductORPComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL moveToPB)))]
  else [prop NONE])  $\wedge$ 
 $(\forall xs.$ 
  secContext MOVE_TO_PB  $xs =$ 
  if getOmniCommand  $xs =$  ssmConductPBComplete then
    [prop (SOME (SLc (OMNI ssmConductPBComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL conductPB)))]
  else [prop NONE])  $\wedge$ 
 $\forall xs.$ 
  secContext CONDUCT_PB  $xs =$ 
  if getOmniCommand  $xs =$  ssmConductPBComplete then
    [prop (SOME (SLc (OMNI ssmConductPBComplete))) impf
     Name PlatoonLeader controls
     prop (SOME (SLc (PL completePB)))]
  else [prop NONE]

[secHelper_def]
 $\vdash \forall cmd.$ 
  secHelper  $cmd =$ 
  [Name Omni controls prop (SOME (SLc (OMNI cmd)))]

```

6.2 Theorems

[getOmniCommand_def]

```

 $\vdash (\text{getOmniCommand} [] = \text{invalidOmniCommand}) \wedge$ 
 $(\forall xs\ cmd.$ 
  getOmniCommand
  (Name Omni says prop (SOME (SLc (OMNI cmd)))):: $xs) =$ 
  cmd)  $\wedge$ 
 $(\forall xs.\ \text{getOmniCommand} (\text{TT}::xs) = \text{getOmniCommand} xs) \wedge$ 
 $(\forall xs.\ \text{getOmniCommand} (\text{FF}::xs) = \text{getOmniCommand} xs) \wedge$ 
 $(\forall xs\ v_2.\ \text{getOmniCommand} (\text{prop } v_2::xs) = \text{getOmniCommand} xs) \wedge$ 
 $(\forall xs\ v_3.\ \text{getOmniCommand} (\text{notf } v_3::xs) = \text{getOmniCommand} xs) \wedge$ 
 $(\forall xs\ v_5\ v_4.$ 

```

```

getOmniCommand (v4 andf v5::xs) = getOmniCommand xs) ∧
(∀ xs v7 v6.
  getOmniCommand (v6 orf v7::xs) = getOmniCommand xs) ∧
(∀ xs v9 v8.
  getOmniCommand (v8 impf v9::xs) = getOmniCommand xs) ∧
(∀ xs v11 v10.
  getOmniCommand (v10 eqf v11::xs) = getOmniCommand xs) ∧
(∀ xs v12.
  getOmniCommand (v12 says TT::xs) = getOmniCommand xs) ∧
(∀ xs v13.
  getOmniCommand (v12 says FF::xs) = getOmniCommand xs) ∧
(∀ xs v134.
  getOmniCommand (Name v134 says prop NONE::xs) =
  getOmniCommand xs) ∧
(∀ xs v144.
  getOmniCommand
    (Name PlatoonLeader says prop (SOME v144)::xs) =
  getOmniCommand xs) ∧
(∀ xs v146.
  getOmniCommand
    (Name Omni says prop (SOME (ESCc v146))::xs) =
  getOmniCommand xs) ∧
(∀ xs v150.
  getOmniCommand
    (Name Omni says prop (SOME (SLc (PL v150)))::xs) =
  getOmniCommand xs) ∧
(∀ xs v68 v136 v135.
  getOmniCommand (v135 meet v136 says prop v68::xs) =
  getOmniCommand xs) ∧
(∀ xs v68 v138 v137.
  getOmniCommand (v137 quoting v138 says prop v68::xs) =
  getOmniCommand xs) ∧
(∀ xs v69 v12.
  getOmniCommand (v12 says notf v69::xs) =
  getOmniCommand xs) ∧
(∀ xs v71 v70 v12.
  getOmniCommand (v12 says (v70 andf v71)::xs) =
  getOmniCommand xs) ∧
(∀ xs v73 v72 v12.
  getOmniCommand (v12 says (v72 orf v73)::xs) =
  getOmniCommand xs) ∧
(∀ xs v75 v74 v12.
  getOmniCommand (v12 says (v74 impf v75)::xs) =
  getOmniCommand xs) ∧
(∀ xs v77 v76 v12.
  getOmniCommand (v12 says (v76 eqf v77)::xs) =
  getOmniCommand xs) ∧
(∀ xs v79 v78 v12.
  getOmniCommand (v12 says v78 says v79::xs) =

```

```

    getOmniCommand xs) ∧
(∀xs v81 v80 v12.
    getOmniCommand (v12 says v80 speaks_for v81::xs) =
    getOmniCommand xs) ∧
(∀xs v83 v82 v12.
    getOmniCommand (v12 says v82 controls v83::xs) =
    getOmniCommand xs) ∧
(∀xs v86 v85 v84 v12.
    getOmniCommand (v12 says reps v84 v85 v86::xs) =
    getOmniCommand xs) ∧
(∀xs v88 v87 v12.
    getOmniCommand (v12 says v87 domi v88::xs) =
    getOmniCommand xs) ∧
(∀xs v90 v89 v12.
    getOmniCommand (v12 says v89 eqi v90::xs) =
    getOmniCommand xs) ∧
(∀xs v92 v91 v12.
    getOmniCommand (v12 says v91 doms v92::xs) =
    getOmniCommand xs) ∧
(∀xs v94 v93 v12.
    getOmniCommand (v12 says v93 eqs v94::xs) =
    getOmniCommand xs) ∧
(∀xs v96 v95 v12.
    getOmniCommand (v12 says v95 eqn v96::xs) =
    getOmniCommand xs) ∧
(∀xs v98 v97 v12.
    getOmniCommand (v12 says v97 lte v98::xs) =
    getOmniCommand xs) ∧
(∀xs v99 v12 v100.
    getOmniCommand (v12 says v99 lt v100::xs) =
    getOmniCommand xs) ∧
(∀xs v15 v14.
    getOmniCommand (v14 speaks_for v15::xs) =
    getOmniCommand xs) ∧
(∀xs v17 v16.
    getOmniCommand (v16 controls v17::xs) =
    getOmniCommand xs) ∧
(∀xs v20 v19 v18.
    getOmniCommand (reps v18 v19 v20::xs) =
    getOmniCommand xs) ∧
(∀xs v22 v21.
    getOmniCommand (v21 domi v22::xs) = getOmniCommand xs) ∧
(∀xs v24 v23.
    getOmniCommand (v23 eqi v24::xs) = getOmniCommand xs) ∧
(∀xs v26 v25.
    getOmniCommand (v25 doms v26::xs) = getOmniCommand xs) ∧
(∀xs v28 v27.
    getOmniCommand (v27 eqs v28::xs) = getOmniCommand xs) ∧
(∀xs v30 v29.
    getOmniCommand (v29 eqn v30::xs) = getOmniCommand xs)

```

```

getOmniCommand (v29 eqn v30::xs) = getOmniCommand xs) ∧
(∀ xs v32 v31.
  getOmniCommand (v31 lte v32::xs) = getOmniCommand xs) ∧
  ∀ xs v34 v33.
    getOmniCommand (v33 lt v34::xs) = getOmniCommand xs

```

[getOmniCommand_ind]

```

⊢ ∀ P.
  P [] ∧
  (∀ cmd xs.
    P (Name Omni says prop (SOME (SLc (OMNI cmd)))::xs)) ∧
    (∀ xs. P xs ⇒ P (TT::xs)) ∧ (∀ xs. P xs ⇒ P (FF::xs)) ∧
    (∀ v2 xs. P xs ⇒ P (prop v2::xs)) ∧
    (∀ v3 xs. P xs ⇒ P (notf v3::xs)) ∧
    (∀ v4 v5 xs. P xs ⇒ P (v4 andf v5::xs)) ∧
    (∀ v6 v7 xs. P xs ⇒ P (v6 orf v7::xs)) ∧
    (∀ v8 v9 xs. P xs ⇒ P (v8 impf v9::xs)) ∧
    (∀ v10 v11 xs. P xs ⇒ P (v10 eqf v11::xs)) ∧
    (∀ v12 xs. P xs ⇒ P (v12 says TT::xs)) ∧
    (∀ v12 xs. P xs ⇒ P (v12 says FF::xs)) ∧
    (∀ v134 xs. P xs ⇒ P (Name v134 says prop NONE::xs)) ∧
    (∀ v144 xs.
      P xs ⇒
      P (Name PlatoonLeader says prop (SOME v144)::xs)) ∧
      (∀ v146 xs.
        P xs ⇒ P (Name Omni says prop (SOME (ESCc v146))::xs)) ∧
        (∀ v150 xs.
          P xs ⇒
          P (Name Omni says prop (SOME (SLc (PL v150)))::xs)) ∧
          (∀ v135 v136 v68 xs.
            P xs ⇒ P (v135 meet v136 says prop v68::xs)) ∧
            (∀ v137 v138 v68 xs.
              P xs ⇒ P (v137 quoting v138 says prop v68::xs)) ∧
              (∀ v12 v69 xs. P xs ⇒ P (v12 says notf v69::xs)) ∧
              (∀ v12 v70 v71 xs. P xs ⇒ P (v12 says (v70 andf v71)::xs)) ∧
              (∀ v12 v72 v73 xs. P xs ⇒ P (v12 says (v72 orf v73)::xs)) ∧
              (∀ v12 v74 v75 xs. P xs ⇒ P (v12 says (v74 impf v75)::xs)) ∧
              (∀ v12 v76 v77 xs. P xs ⇒ P (v12 says (v76 eqf v77)::xs)) ∧
              (∀ v12 v78 v79 xs. P xs ⇒ P (v12 says v78 says v79::xs)) ∧
              (∀ v12 v80 v81 xs.
                P xs ⇒ P (v12 says v80 speaks_for v81::xs)) ∧
                (∀ v12 v82 v83 xs.
                  P xs ⇒ P (v12 says v82 controls v83::xs)) ∧
                  (∀ v12 v84 v85 v86 xs.
                    P xs ⇒ P (v12 says reps v84 v85 v86::xs)) ∧
                    (∀ v12 v87 v88 xs. P xs ⇒ P (v12 says v87 domi v88::xs)) ∧
                    (∀ v12 v89 v90 xs. P xs ⇒ P (v12 says v89 equi v90::xs)) ∧
                    (∀ v12 v91 v92 xs. P xs ⇒ P (v12 says v91 doms v92::xs)) ∧
                    (∀ v12 v93 v94 xs. P xs ⇒ P (v12 says v93 eqs v94::xs)) ∧

```

$$\begin{aligned}
 & (\forall v_{12} v_{95} v_{96} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{95} \text{ eqn } v_{96}::xs)) \wedge \\
 & (\forall v_{12} v_{97} v_{98} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{97} \text{ lte } v_{98}::xs)) \wedge \\
 & (\forall v_{12} v_{99} v_{100} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{99} \text{ lt } v_{100}::xs)) \wedge \\
 & (\forall v_{14} v_{15} xs. P xs \Rightarrow P (v_{14} \text{ speaks_for } v_{15}::xs)) \wedge \\
 & (\forall v_{16} v_{17} xs. P xs \Rightarrow P (v_{16} \text{ controls } v_{17}::xs)) \wedge \\
 & (\forall v_{18} v_{19} v_{20} xs. P xs \Rightarrow P (\text{reps } v_{18} v_{19} v_{20}::xs)) \wedge \\
 & (\forall v_{21} v_{22} xs. P xs \Rightarrow P (v_{21} \text{ domi } v_{22}::xs)) \wedge \\
 & (\forall v_{23} v_{24} xs. P xs \Rightarrow P (v_{23} \text{ eqi } v_{24}::xs)) \wedge \\
 & (\forall v_{25} v_{26} xs. P xs \Rightarrow P (v_{25} \text{ doms } v_{26}::xs)) \wedge \\
 & (\forall v_{27} v_{28} xs. P xs \Rightarrow P (v_{27} \text{ eqs } v_{28}::xs)) \wedge \\
 & (\forall v_{29} v_{30} xs. P xs \Rightarrow P (v_{29} \text{ eqn } v_{30}::xs)) \wedge \\
 & (\forall v_{31} v_{32} xs. P xs \Rightarrow P (v_{31} \text{ lte } v_{32}::xs)) \wedge \\
 & (\forall v_{33} v_{34} xs. P xs \Rightarrow P (v_{33} \text{ lt } v_{34}::xs)) \Rightarrow \\
 & \forall v. P v
 \end{aligned}$$

[getPlCom_def]

$$\begin{aligned}
 \vdash & (\text{getPlCom } [] = \text{incomplete}) \wedge \\
 & (\forall xs cmd. \text{getPlCom } (\text{SOME } (\text{SLc (PL cmd)})::xs) = cmd) \wedge \\
 & (\forall xs. \text{getPlCom } (\text{NONE}::xs) = \text{getPlCom } xs) \wedge \\
 & (\forall xs v_4. \text{getPlCom } (\text{SOME } (\text{ESCc } v_4)::xs) = \text{getPlCom } xs) \wedge \\
 & \forall xs v_9. \text{getPlCom } (\text{SOME } (\text{SLc (OMNI } v_9))::xs) = \text{getPlCom } xs
 \end{aligned}$$

[getPlCom_ind]

$$\begin{aligned}
 \vdash & \forall P. \\
 & P [] \wedge (\forall cmd xs. P (\text{SOME } (\text{SLc (PL cmd)})::xs)) \wedge \\
 & (\forall xs. P xs \Rightarrow P (\text{NONE}::xs)) \wedge \\
 & (\forall v_4 xs. P xs \Rightarrow P (\text{SOME } (\text{ESCc } v_4)::xs)) \wedge \\
 & (\forall v_9 xs. P xs \Rightarrow P (\text{SOME } (\text{SLc (OMNI } v_9))::xs)) \Rightarrow \\
 & \forall v. P v
 \end{aligned}$$

7 ssmPBIntegrated Theory

Built: 11 June 2018

Parent Theories: PBIntegratedDef, ssm

7.1 Theorems

[inputOK_cmd_reject_lemma]

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

[inputOK_def]

$$\begin{aligned}
 \vdash & (\text{inputOK } (\text{Name PlatoonLeader says prop } cmd) \iff \text{T}) \wedge \\
 & (\text{inputOK } (\text{Name Omni says prop } cmd) \iff \text{T}) \wedge \\
 & (\text{inputOK } \text{TT} \iff \text{F}) \wedge (\text{inputOK } \text{FF} \iff \text{F}) \wedge \\
 & (\text{inputOK } (\text{prop } v) \iff \text{F}) \wedge (\text{inputOK } (\text{notf } v_1) \iff \text{F}) \wedge \\
 & (\text{inputOK } (v_2 \text{ andf } v_3) \iff \text{F}) \wedge (\text{inputOK } (v_4 \text{ orf } v_5) \iff \text{F}) \wedge \\
 & (\text{inputOK } (v_6 \text{ impf } v_7) \iff \text{F}) \wedge (\text{inputOK } (v_8 \text{ eqf } v_9) \iff \text{F})
 \end{aligned}$$

```

(inputOK (v10 says TT)  $\iff$  F)  $\wedge$  (inputOK (v10 says FF)  $\iff$  F)  $\wedge$ 
(inputOK (v133 meet v134 says prop v66)  $\iff$  F)  $\wedge$ 
(inputOK (v135 quoting v136 says prop v66)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says notf v67)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says (v68 andf v69))  $\iff$  F)  $\wedge$ 
(inputOK (v10 says (v70 orf v71))  $\iff$  F)  $\wedge$ 
(inputOK (v10 says (v72 impf v73))  $\iff$  F)  $\wedge$ 
(inputOK (v10 says (v74 eqf v75))  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v76 says v77)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v78 speaks_for v79)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v80 controls v81)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says reps v82 v83 v84)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v85 domi v86)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v87 equi v88)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v89 doms v90)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v91 eqs v92)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v93 eqn v94)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v95 lte v96)  $\iff$  F)  $\wedge$ 
(inputOK (v10 says v97 lt v98)  $\iff$  F)  $\wedge$ 
(inputOK (v12 speaks_for v13)  $\iff$  F)  $\wedge$ 
(inputOK (v14 controls v15)  $\iff$  F)  $\wedge$ 
(inputOK (reps v16 v17 v18)  $\iff$  F)  $\wedge$ 
(inputOK (v19 domi v20)  $\iff$  F)  $\wedge$ 
(inputOK (v21 equi v22)  $\iff$  F)  $\wedge$ 
(inputOK (v23 doms v24)  $\iff$  F)  $\wedge$ 
(inputOK (v25 eqs v26)  $\iff$  F)  $\wedge$  (inputOK (v27 eqn v28)  $\iff$  F)  $\wedge$ 
(inputOK (v29 lte v30)  $\iff$  F)  $\wedge$  (inputOK (v31 lt v32)  $\iff$  F)

```

[inputOK_ind]

```

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

```

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

[PBNS_def]

$$\begin{aligned}
\vdash & (\text{PBNS PLAN_PB } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{crossLD} \text{ then MOVE_TO_ORP else PLAN_PB}) \wedge \\
& (\text{PBNS MOVE_TO_ORP } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{conductORP} \text{ then CONDUCT_ORP} \\
& \quad \text{else MOVE_TO_ORP}) \wedge \\
& (\text{PBNS CONDUCT_ORP } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{moveToPB} \text{ then MOVE_TO_PB else CONDUCT_ORP}) \wedge \\
& (\text{PBNS MOVE_TO_PB } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{conductPB} \text{ then CONDUCT_PB else MOVE_TO_PB}) \wedge \\
& (\text{PBNS CONDUCT_PB } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{completePB} \text{ then COMPLETE_PB} \\
& \quad \text{else CONDUCT_PB}) \wedge (\text{PBNS } s \text{ (trap } v_0) = s) \wedge \\
& (\text{PBNS } s \text{ (discard } v_1) = s)
\end{aligned}$$

[PBNS_ind]

$$\begin{aligned}
\vdash & \forall P. \\
& (\forall x. P \text{ PLAN_PB } (\text{exec } x)) \wedge (\forall x. P \text{ MOVE_TO_ORP } (\text{exec } x)) \wedge \\
& (\forall x. P \text{ CONDUCT_ORP } (\text{exec } x)) \wedge \\
& (\forall x. P \text{ MOVE_TO_PB } (\text{exec } x)) \wedge (\forall x. P \text{ CONDUCT_PB } (\text{exec } x)) \wedge \\
& (\forall s v_0. P s \text{ (trap } v_0)) \wedge (\forall s v_1. P s \text{ (discard } v_1)) \wedge \\
& (\forall v_6. P \text{ COMPLETE_PB } (\text{exec } v_6)) \Rightarrow \\
& \forall v v_1. P v v_1
\end{aligned}$$

[PBOOut_def]

$$\begin{aligned}
\vdash & (\text{PBOOut PLAN_PB } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{crossLD} \text{ then MoveToORP else PlanPB}) \wedge \\
& (\text{PBOOut MOVE_TO_ORP } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{conductORP} \text{ then ConductORP else MoveToORP}) \wedge \\
& (\text{PBOOut CONDUCT_ORP } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{moveToPB} \text{ then MoveToORP else ConductORP}) \wedge \\
& (\text{PBOOut MOVE_TO_PB } (\text{exec } x) = \\
& \quad \text{if getPlCom } x = \text{conductPB} \text{ then ConductPB else MoveToPB}) \wedge
\end{aligned}$$

```
(PBOut CONDUCT_PB (exec x) =
  if getPlCom x = completePB then CompletePB else ConductPB)  $\wedge$ 
(PBOut s (trap v0) = unAuthorized)  $\wedge$ 
(PBOut s (discard v1) = unAuthenticated)
```

[PBOut_ind]

```
 $\vdash \forall P.$ 
 $(\forall x. P \text{ PLAN\_PB} (\text{exec } x)) \wedge (\forall x. P \text{ MOVE\_TO\_ORP} (\text{exec } x)) \wedge$ 
 $(\forall x. P \text{ CONDUCT\_ORP} (\text{exec } x)) \wedge$ 
 $(\forall x. P \text{ MOVE\_TO\_PB} (\text{exec } x)) \wedge (\forall x. P \text{ CONDUCT\_PB} (\text{exec } x)) \wedge$ 
 $(\forall s v_0. P s (\text{trap } v_0)) \wedge (\forall s v_1. P s (\text{discard } v_1)) \wedge$ 
 $(\forall v_6. P \text{ COMPLETE\_PB} (\text{exec } v_6)) \Rightarrow$ 
 $\forall v v_1. P v v_1$ 
```

[PlatoonLeader_Omni_notDiscard_s1Command_thm]

```
 $\vdash \forall NS \text{ Out } M \text{ Oi } Os.$ 
 $\neg \text{TR } (M, Oi, Os)$ 
 $(\text{discard}$ 
 $[ \text{SOME } (\text{SLc } (\text{PL } plCommand));$ 
 $\text{SOME } (\text{SLc } (\text{OMNI } omniCommand))])$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $([\text{Name Omni says prop } (\text{SOME } (\text{SLc } (\text{PL } plCommand))) ;$ 
 $\text{Name PlatoonLeader says}$ 
 $\text{prop } (\text{SOME } (\text{SLc } (\text{OMNI } omniCommand))))] :: ins) \text{ PLAN\_PB}$ 
 $outs)$ 
 $(\text{CFG inputOK secContext secAuthorization } ins$ 
 $(NS \text{ PLAN\_PB}$ 
 $(\text{discard}$ 
 $[ \text{SOME } (\text{SLc } (\text{PL } plCommand));$ 
 $\text{SOME } (\text{SLc } (\text{OMNI } omniCommand))])])$ 
 $(Out \text{ PLAN\_PB}$ 
 $(\text{discard}$ 
 $[ \text{SOME } (\text{SLc } (\text{PL } plCommand));$ 
 $\text{SOME } (\text{SLc } (\text{OMNI } omniCommand))]) :: outs))$ 
```

[PlatoonLeader_PLAN_PB_exec_justified_lemma]

```
 $\vdash \forall NS \text{ Out } M \text{ Oi } Os.$ 
 $\text{TR } (M, Oi, Os)$ 
 $(\text{exec}$ 
 $(\text{inputList}$ 
 $[ \text{Name Omni says}$ 
 $\text{prop } (\text{SOME } (\text{SLc } (\text{OMNI } ssmPlanPBComplete)));$ 
 $\text{Name PlatoonLeader says}$ 
 $\text{prop } (\text{SOME } (\text{SLc } (\text{PL } crossLD))))])$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $([\text{Name Omni says}$ 
 $\text{prop } (\text{SOME } (\text{SLc } (\text{OMNI } ssmPlanPBComplete)));$ 
 $\text{Name PlatoonLeader says}$ 
```

```

prop (SOME (SLc (PL crossLD))))::ins) PLAN_PB outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(exec
(inputList
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))))
(Out PLAN_PB
(exec
(inputList
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))]))::outs)) ⇔
authenticationTest inputOK
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))] ∧
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))]::ins) PLAN_PB
outs) ∧
(M, Oi, Os) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))]
```

[PlatoonLeader_PLAN_PB_exec_justified_thm]

```

⊤ ∀ NS Out M Oi Os.
TR (M, Oi, Os)
(exec
[SOME (SLc (OMNI ssmPlanPBComplete));
SOME (SLc (PL crossLD))])
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmPlanPBComplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))]::ins) PLAN_PB outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(exec
[SOME (SLc (OMNI ssmPlanPBComplete));
```

```

        SOME (SLc (PL crossLD))])))
(Out PLAN_PB
  (exec
    [SOME (SLc (OMNI ssmPlanPBComplete));
     SOME (SLc (PL crossLD))]]::outs))  $\iff$ 
authenticationTest inputOK
  [Name Omni says
   prop (SOME (SLc (OMNI ssmPlanPBComplete)));
   Name PlatoonLeader says
   prop (SOME (SLc (PL crossLD))))]  $\wedge$ 
CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secAuthorization
    ([Name Omni says
     prop (SOME (SLc (OMNI ssmPlanPBComplete)));
     Name PlatoonLeader says
     prop (SOME (SLc (PL crossLD)))]::ins) PLAN_PB
     outs)  $\wedge$ 
(M, Oi, Os) satList
  [prop (SOME (SLc (OMNI ssmPlanPBComplete)));
   prop (SOME (SLc (PL crossLD)))]
```

[PlatoonLeader_PLAN_PB_exec_lemma]

```

 $\vdash \forall M\ Oi\ Os.$ 
  CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secAuthorization
    ([Name Omni says
     prop (SOME (SLc (OMNI ssmPlanPBComplete)));
     Name PlatoonLeader says
     prop (SOME (SLc (PL crossLD)))]::ins) PLAN_PB
     outs)  $\Rightarrow$ 
(M, Oi, Os) satList
  propCommandList
    [Name Omni says
     prop (SOME (SLc (OMNI ssmPlanPBComplete)));
     Name PlatoonLeader says prop (SOME (SLc (PL crossLD)))]
```

[PlatoonLeader_PLAN_PB_trap_justified_lemma]

```

 $\vdash omniCommand \neq ssmPlanPBComplete \Rightarrow$ 
(s = PLAN_PB)  $\Rightarrow$ 
 $\forall NS\ Out\ M\ Oi\ Os.$ 
  TR (M, Oi, Os)
  (trap
    (inputList
      [Name Omni says
       prop (SOME (SLc (OMNI omniCommand)));
       Name PlatoonLeader says
       prop (SOME (SLc (PL crossLD))))])
    (CFG inputOK secContext secAuthorization
      ([Name Omni says prop (SOME (SLc (OMNI omniCommand))));
```

```

Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))::ins) PLAN_PB outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(trap
(inputList
[Name Omni says
prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))))
(Out PLAN_PB
(trap
(inputList
[Name Omni says
prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))])))::outs)) ⇔
authenticationTest inputOK
[Name Omni says prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))] ∧
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD))))::ins) PLAN_PB
outs) ∧ (M, Oi, Os) sat prop NONE

```

[PlatoonLeader_PLAN_PB_trap_justified_thm]

```

⊢ omniCommand ≠ ssmPlanPBComplete ⇒
(s = PLAN_PB) ⇒
∀ NS Out M Oi Os.
TR (M, Oi, Os)
(trap
[SOME (SLc (OMNI omniCommand));
SOME (SLc (PL crossLD))])
(CFG inputOK secContext secAuthorization
([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
Name PlatoonLeader says
prop (SOME (SLc (PL crossLD)))]::ins) PLAN_PB outs)
(CFG inputOK secContext secAuthorization ins
(NS PLAN_PB
(trap
[SOME (SLc (OMNI omniCommand));
SOME (SLc (PL crossLD))]))
(Out PLAN_PB
(trap
[SOME (SLc (OMNI omniCommand));
SOME (SLc (PL crossLD))])::outs)) ⇔

```

```

authenticationTest inputOK
  [Name Omni says prop (SOME (SLc (OMNI omniCommand)));
   Name PlatoonLeader says
   prop (SOME (SLc (PL crossLD)))] ∧
CFGInterpret ( $M, O_i, O_s$ )
  (CFG inputOK secContext secAuthorization
    ([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
     Name PlatoonLeader says
     prop (SOME (SLc (PL crossLD))))]::ins) PLAN_PB
    outs) ∧ ( $M, O_i, O_s$ ) sat prop NONE

```

[PlatoonLeader_PLAN_PB_trap_lemma]

```

 $\vdash omniCommand \neq ssmPlanPBComplete \Rightarrow$ 
 $(s = PLAN\_PB) \Rightarrow$ 
 $\forall M \ O_i \ O_s.$ 
CFGInterpret ( $M, O_i, O_s$ )
  (CFG inputOK secContext secAuthorization
    ([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
     Name PlatoonLeader says
     prop (SOME (SLc (PL crossLD))))]::ins) PLAN_PB
    outs) ⇒
 $(M, O_i, O_s)$  sat prop NONE

```

8 ssmConductORP Theory

Built: 11 June 2018

Parent Theories: ConductORPDef

8.1 Theorems

[conductORPNS_def]

```

 $\vdash (conductORPNS CONDUCT\_ORP (exec x) =$ 
 $\quad \text{if getPlCom } x = \text{secure} \text{ then SECURE else CONDUCT\_ORP}) \wedge$ 
 $(conductORPNS SECURE (exec x) =$ 
 $\quad \text{if getPsgCom } x = \text{actionsIn} \text{ then ACTIONS\_IN else SECURE}) \wedge$ 
 $(conductORPNS ACTIONS\_IN (exec x) =$ 
 $\quad \text{if getPlCom } x = \text{withdraw} \text{ then WITHDRAW else ACTIONS\_IN}) \wedge$ 
 $(conductORPNS WITHDRAW (exec x) =$ 
 $\quad \text{if getPlCom } x = \text{complete} \text{ then COMPLETE else WITHDRAW}) \wedge$ 
 $(conductORPNS s (\text{trap } x) = s) \wedge$ 
 $(conductORPNS s (\text{discard } x) = s)$ 

```

[conductORPNS_ind]

```

 $\vdash \forall P.$ 
 $(\forall x. \ P \text{ CONDUCT\_ORP} (\text{exec } x)) \wedge (\forall x. \ P \text{ SECURE} (\text{exec } x)) \wedge$ 
 $(\forall x. \ P \text{ ACTIONS\_IN} (\text{exec } x)) \wedge (\forall x. \ P \text{ WITHDRAW} (\text{exec } x)) \wedge$ 
 $(\forall s \ x. \ P \ s (\text{trap } x)) \wedge (\forall s \ x. \ P \ s (\text{discard } x)) \wedge$ 

```

$$(\forall v_5. P \text{ COMPLETE } (\text{exec } v_5)) \Rightarrow \\ \forall v v_1. P v v_1$$

[conductORPOut_def]

$$\vdash (\text{conductORPOut CONDUCT_ORP } (\text{exec } x) = \\ \text{if getPlCom } x = \text{secure then Secure else ConductORP}) \wedge \\ (\text{conductORPOut SECURE } (\text{exec } x) = \\ \text{if getPsgCom } x = \text{actionsIn then ActionsIn else Secure}) \wedge \\ (\text{conductORPOut ACTIONS_IN } (\text{exec } x) = \\ \text{if getPlCom } x = \text{withdraw then Withdraw else ActionsIn}) \wedge \\ (\text{conductORPOut WITHDRAW } (\text{exec } x) = \\ \text{if getPlCom } x = \text{complete then Complete else Withdraw}) \wedge \\ (\text{conductORPOut } s \text{ (trap } x) = \text{unAuthorized}) \wedge \\ (\text{conductORPOut } s \text{ (discard } x) = \text{unAuthenticated})$$

[conductORPOut_ind]

$$\vdash \forall P. \\ (\forall x. P \text{ CONDUCT_ORP } (\text{exec } x)) \wedge (\forall x. P \text{ SECURE } (\text{exec } x)) \wedge \\ (\forall x. P \text{ ACTIONS_IN } (\text{exec } x)) \wedge (\forall x. P \text{ WITHDRAW } (\text{exec } x)) \wedge \\ (\forall s x. P s \text{ (trap } x)) \wedge (\forall s x. P s \text{ (discard } x)) \wedge \\ (\forall v_5. P \text{ COMPLETE } (\text{exec } v_5)) \Rightarrow \\ \forall v v_1. P v v_1$$

[inputOK_cmd_reject_lemma]

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

[inputOK_def]

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

```

(inputOK (v10 says v89 doms v90)  $\iff$  F)  $\wedge$ 
10 says v91 eqs v92)  $\iff$  F)  $\wedge$ 
10 says v93 eqn v94)  $\iff$  F)  $\wedge$ 
10 says v95 lte v96)  $\iff$  F)  $\wedge$ 
10 says v97 lt v98)  $\iff$  F)  $\wedge$ 
12 speaks_for v13)  $\iff$  F)  $\wedge$ 
14 controls v15)  $\iff$  F)  $\wedge$ 
16 v17 v18)  $\iff$  F)  $\wedge$ 
19 domi v20)  $\iff$  F)  $\wedge$ 
21 eqi v22)  $\iff$  F)  $\wedge$ 
23 doms v24)  $\iff$  F)  $\wedge$ 
25 eqs v26)  $\iff$  F)  $\wedge$  (inputOK (v27 eqn v28)  $\iff$  F)  $\wedge$ 
29 lte v30)  $\iff$  F)  $\wedge$  (inputOK (v31 lt v32)  $\iff$  F)

```

[inputOK_ind]

```

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

```

$\forall v. \ P \ v$

[PlatoonLeader_ACTIONS_IN_exec_justified_lemma]

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

$$\text{TR} (M, Oi, Os)$$

$$(\text{exec}$$

$$(\text{inputList}$$

$$[\text{Name Omni says}$$

$$\text{prop (SOME (SLc (OMNI ssmActionsIncomplete)))};$$

$$\text{Name PlatoonLeader says}$$

$$\text{prop (SOME (SLc (PL withdraw))))}]))$$

$$(\text{CFG inputOK secContext secAuthorization}$$

$$([\text{Name Omni says}$$

$$\text{prop (SOME (SLc (OMNI ssmActionsIncomplete)))};$$

$$\text{Name PlatoonLeader says}$$

$$\text{prop (SOME (SLc (PL withdraw))))}] :: ins) \text{ ACTIONS_IN}$$

$$outs)$$

$$(\text{CFG inputOK secContext secAuthorization} \ ins$$

$$(NS \text{ ACTIONS_IN}$$

$$(\text{exec}$$

$$(\text{inputList}$$

$$[\text{Name Omni says}$$

$$\text{prop}$$

$$(\text{SOME (SLc (OMNI ssmActionsIncomplete)))};$$

$$\text{Name PlatoonLeader says}$$

$$\text{prop (SOME (SLc (PL withdraw))))})))$$

$$(Out \text{ ACTIONS_IN}$$

$$(\text{exec}$$

$$(\text{inputList}$$

$$[\text{Name Omni says}$$

$$\text{prop}$$

$$(\text{SOME (SLc (OMNI ssmActionsIncomplete)))};$$

$$\text{Name PlatoonLeader says}$$

$$\text{prop (SOME (SLc (PL withdraw))))}] ::$$

$$outs)) \iff$$

$$\text{authenticationTest inputOK}$$

$$[\text{Name Omni says}$$

$$\text{prop (SOME (SLc (OMNI ssmActionsIncomplete)))};$$

$$\text{Name PlatoonLeader says}$$

$$\text{prop (SOME (SLc (PL withdraw)))}] \wedge$$

$$\text{CFGInterpret} (M, Oi, Os)$$

$$(\text{CFG inputOK secContext secAuthorization}$$

$$([\text{Name Omni says}$$

$$\text{prop (SOME (SLc (OMNI ssmActionsIncomplete)))};$$

$$\text{Name PlatoonLeader says}$$

$$\text{prop (SOME (SLc (PL withdraw))))}] :: ins) \text{ ACTIONS_IN}$$

$$outs) \wedge$$

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

$$\text{propCommandList}$$

```

[Name Omni says
 prop (SOME (SLc (OMNI ssmActionsIncomplete)));
 Name PlatoonLeader says prop (SOME (SLc (PL withdraw)))]

[PlatoonLeader_ACTIONS_IN_exec_justified_thm]
 $\vdash \forall NS\ Out\ M\ Oi\ Os.$ 
 $\text{TR} (M, Oi, Os)$ 
 $(\text{exec}$ 
 $\quad [\text{SOME} (\text{SLc} (\text{OMNI ssmActionsIncomplete}));$ 
 $\quad \text{SOME} (\text{SLc} (\text{PL withdraw})))])$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $\quad ([\text{Name Omni says}$ 
 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{OMNI ssmActionsIncomplete})));$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{PL withdraw}))))] :: ins) \text{ ACTIONS\_IN}$ 
 $\quad outs)$ 
 $(\text{CFG inputOK secContext secAuthorization} ins$ 
 $\quad (NS \text{ ACTIONS\_IN}$ 
 $\quad (\text{exec}$ 
 $\quad [\text{SOME} (\text{SLc} (\text{OMNI ssmActionsIncomplete}));$ 
 $\quad \text{SOME} (\text{SLc} (\text{PL withdraw})))]) :: outs)) \iff$ 
 $\text{authenticationTest inputOK}$ 
 $\quad [\text{Name Omni says}$ 
 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{OMNI ssmActionsIncomplete})));$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{PL withdraw}))))] \wedge$ 
 $\text{CFGInterpret} (M, Oi, Os)$ 
 $\quad (\text{CFG inputOK secContext secAuthorization}$ 
 $\quad (\text{[Name Omni says}$ 
 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{OMNI ssmActionsIncomplete})));$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{PL withdraw}))))] :: ins) \text{ ACTIONS\_IN}$ 
 $\quad outs) \wedge$ 
 $(M, Oi, Os) \text{ satList}$ 
 $[\text{prop} (\text{SOME} (\text{SLc} (\text{OMNI ssmActionsIncomplete})));$ 
 $\text{prop} (\text{SOME} (\text{SLc} (\text{PL withdraw}))))]$ 

[PlatoonLeader_ACTIONS_IN_exec_lemma]
 $\vdash \forall M\ Oi\ Os.$ 
 $\text{CFGInterpret} (M, Oi, Os)$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $\quad (\text{[Name Omni says}$ 
 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{OMNI ssmActionsIncomplete})));$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{PL withdraw}))))] :: ins) \text{ ACTIONS\_IN}$ 

```

```

    outs) ⇒
  (M , Oi , Os) satList
  propCommandList
  [Name Omni says
    prop (SOME (SLc (OMNI ssmActionsIncomplete)));
    Name PlatoonLeader says prop (SOME (SLc (PL withdraw)))]]

[PlatoonLeader_ACTIONS_IN_trap_justified_lemma]
⊢ omniCommand ≠ ssmActionsIncomplete ⇒
  (s = ACTIONS_IN) ⇒
  ∀ NS Out M Oi Os .
  TR (M , Oi , Os)
  (trap
    (inputList
      [Name Omni says
        prop (SOME (SLc (OMNI omniCommand)));
        Name PlatoonLeader says
        prop (SOME (SLc (PL withdraw))))])
    (CFG inputOK secContext secAuthorization
      ([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
        Name PlatoonLeader says
        prop (SOME (SLc (PL withdraw))))] :: ins) ACTIONS_IN
      outs)
    (CFG inputOK secContext secAuthorization ins
      (NS ACTIONS_IN
        (trap
          (inputList
            [Name Omni says
              prop (SOME (SLc (OMNI omniCommand)));
              Name PlatoonLeader says
              prop (SOME (SLc (PL withdraw))))])
        (Out ACTIONS_IN
          (trap
            (inputList
              [Name Omni says
                prop (SOME (SLc (OMNI omniCommand)));
                Name PlatoonLeader says
                prop (SOME (SLc (PL withdraw))))] :: outs)) ⇔
            authenticationTest inputOK
            [Name Omni says prop (SOME (SLc (OMNI omniCommand)));
              Name PlatoonLeader says
              prop (SOME (SLc (PL withdraw)))] ∧
            CFGInterpret (M , Oi , Os)
            (CFG inputOK secContext secAuthorization
              ([Name Omni says prop (SOME (SLc (OMNI omniCommand)));
                Name PlatoonLeader says
                prop (SOME (SLc (PL withdraw))))] :: ins) ACTIONS_IN
              outs) ∧ (M , Oi , Os) sat prop NONE

```

[PlatoonLeader_ACTIONS_IN_trap_justified_thm]

```

 $\vdash \text{omniCommand} \neq \text{ssmActionsIncomplete} \Rightarrow$ 
 $(s = \text{ACTIONS\_IN}) \Rightarrow$ 
 $\forall NS \text{ Out } M \text{ Oi } Os.$ 
 $\text{TR } (M, Oi, Os)$ 
 $(\text{trap}$ 
 $\quad [\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand}));$ 
 $\quad \text{SOME } (\text{SLc } (\text{PL withdraw}))])$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $\quad ([\text{Name Omni says prop } (\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand})))];$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL withdraw})))]) :: ins) \text{ ACTIONS\_IN}$ 
 $\quad outs)$ 
 $(\text{CFG inputOK secContext secAuthorization } ins$ 
 $\quad (NS \text{ ACTIONS\_IN}$ 
 $\quad (\text{trap}$ 
 $\quad [\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand}));$ 
 $\quad \text{SOME } (\text{SLc } (\text{PL withdraw}))])$ 
 $\quad (Out \text{ ACTIONS\_IN}$ 
 $\quad (\text{trap}$ 
 $\quad [\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand}));$ 
 $\quad \text{SOME } (\text{SLc } (\text{PL withdraw}))]) :: outs) \iff$ 
 $\text{authenticationTest inputOK}$ 
 $\quad [\text{Name Omni says prop } (\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand})))];$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL withdraw})))]) \wedge$ 
 $\text{CFGInterpret } (M, Oi, Os)$ 
 $\quad (\text{CFG inputOK secContext secAuthorization}$ 
 $\quad ([\text{Name Omni says prop } (\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand})))];$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL withdraw})))]) :: ins) \text{ ACTIONS\_IN}$ 
 $\quad outs) \wedge (M, Oi, Os) \text{ sat prop NONE}$ 

```

[PlatoonLeader_ACTIONS_IN_trap_lemma]

```

 $\vdash \text{omniCommand} \neq \text{ssmActionsIncomplete} \Rightarrow$ 
 $(s = \text{ACTIONS\_IN}) \Rightarrow$ 
 $\forall M \text{ Oi } Os.$ 
 $\text{CFGInterpret } (M, Oi, Os)$ 
 $(\text{CFG inputOK secContext secAuthorization}$ 
 $\quad ([\text{Name Omni says prop } (\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand})))];$ 
 $\quad \text{Name PlatoonLeader says}$ 
 $\quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL withdraw})))]) :: ins) \text{ ACTIONS\_IN}$ 
 $\quad outs) \Rightarrow$ 
 $(M, Oi, Os) \text{ sat prop NONE}$ 

```

[PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm]

```

 $\vdash \forall NS \text{ Out } M \text{ Oi } Os.$ 
 $\text{TR } (M, Oi, Os) \text{ (exec } [\text{SOME } (\text{SLc } (\text{PL secure}))])$ 

```

```

(CFG inputOK secContext secAuthorization
  ([Name PlatoonLeader says
    prop (SOME (SLc (PL secure))))::ins) CONDUCT_ORP
  outs)
(CFG inputOK secContext secAuthorization ins
  (NS CONDUCT_ORP (exec [SOME (SLc (PL secure))]))
  (Out CONDUCT_ORP (exec [SOME (SLc (PL secure))])::outs))  $\iff$ 
authenticationTest inputOK
  ([Name PlatoonLeader says prop (SOME (SLc (PL secure)))]  $\wedge$ 
CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secAuthorization
    ([Name PlatoonLeader says
      prop (SOME (SLc (PL secure))))::ins) CONDUCT_ORP
    outs)  $\wedge$ 
  (M, Oi, Os) satList [prop (SOME (SLc (PL secure)))]]

[PlatoonLeader_CONDUCT_ORP_exec_secure_lemma]
 $\vdash \forall M\ Oi\ Os.$ 
  CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secAuthorization
    ([Name PlatoonLeader says
      prop (SOME (SLc (PL secure))))::ins) CONDUCT_ORP
    outs)  $\Rightarrow$ 
  (M, Oi, Os) satList
  propCommandList
    ([Name PlatoonLeader says prop (SOME (SLc (PL secure)))])

[PlatoonSergeant_SECURE_exec_justified_lemma]
 $\vdash \forall NS\ Out\ M\ Oi\ Os.$ 
  TR (M, Oi, Os)
  (exec
    (inputList
      [Name Omni says
        prop (SOME (SLc (OMNI ssmSecureComplete)));
        Name PlatoonSergeant says
        prop (SOME (SLc (PSG actionsIn))))])
  (CFG inputOK secContext secAuthorization
    ([Name Omni says
      prop (SOME (SLc (OMNI ssmSecureComplete)));
      Name PlatoonSergeant says
      prop (SOME (SLc (PSG actionsIn))))::ins) SECURE
    outs)
  (CFG inputOK secContext secAuthorization ins
    (NS SECURE
      (exec
        (inputList
          [Name Omni says
            prop (SOME (SLc (OMNI ssmSecureComplete))));
```

```

Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn))))])
(Out SECURE
(exec
(inputList
[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn))))]):::
outs)) ⇔
authenticationTest inputOK
[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))] ∧
CFGInterpret (M , Oi , Os)
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn))))]::ins) SECURE
outs) ∧
(M , Oi , Os) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))]
```

[PlatoonSergeant_SECURE_exec_justified_thm]

```

⊢ ∀ NS Out M Oi Os .
TR (M , Oi , Os)
(exec
[SOME (SLc (OMNI ssmSecureComplete));
 SOME (SLc (PSG actionsIn))])
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn))))]::ins) SECURE
outs)
(CFG inputOK secContext secAuthorization ins
(NS SECURE
(exec
[SOME (SLc (OMNI ssmSecureComplete));
 SOME (SLc (PSG actionsIn))]))
(Out SECURE
(exec
[SOME (SLc (OMNI ssmSecureComplete));
```

```

        SOME (SLc (PSG actionsIn)))]]::outs))  $\iff$ 
authenticationTest inputOK
[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))]  $\wedge$ 
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))])::ins) SECURE
outs)  $\wedge$ 
(M, Oi, Os) satList
[prop (SOME (SLc (OMNI ssmSecureComplete)));
prop (SOME (SLc (PSG actionsIn)))]

```

[PlatoonSergeant_SECURE_exec_lemma]

```

 $\vdash \forall M\ Oi\ Os.$ 
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))])::ins) SECURE
outs)  $\Rightarrow$ 
(M, Oi, Os) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmSecureComplete)));
Name PlatoonSergeant says
prop (SOME (SLc (PSG actionsIn)))]

```

9 ConductORPType Theory

Built: 11 June 2018

Parent Theories: indexedLists, patternMatches

9.1 Datatypes

```

omniCommand = ssmSecureComplete | ssmActionsIncomplete
| ssmWithdrawComplete | invalidOmniCommand

```

```

plCommand = secure | withdraw | complete | plIncomplete

```

```

psgCommand = actionsIn | psgIncomplete

```

```

 $slCommand =$ 
  PL ConductORPType$plCommand
  | PSG ConductORPType$psgCommand
  | OMNI omniCommand

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

 $slState =$  CONDUCT_ORP | SECURE | ACTIONS_IN | WITHDRAW
  | COMPLETE

 $stateRole =$  PlatoonLeader | PlatoonSergeant | Omni

```

9.2 Theorems

[omniCommand_distinct_clauses]

```

 $\vdash ssmSecureComplete \neq ssmActionsIncomplete \wedge$ 
 $ssmSecureComplete \neq ssmWithdrawComplete \wedge$ 
 $ssmSecureComplete \neq invalidOmniCommand \wedge$ 
 $ssmActionsIncomplete \neq ssmWithdrawComplete \wedge$ 
 $ssmActionsIncomplete \neq invalidOmniCommand \wedge$ 
 $ssmWithdrawComplete \neq invalidOmniCommand$ 

```

[plCommand_distinct_clauses]

```

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

```

[psgCommand_distinct_clauses]

```

 $\vdash actionsIn \neq psgIncomplete$ 

```

[slCommand_distinct_clauses]

```

 $\vdash (\forall a' a. PL a \neq PSG a') \wedge (\forall a' a. PL a \neq OMNI a') \wedge$ 
 $\forall a' a. PSG a \neq OMNI a'$ 

```

[slCommand_one_one]

```

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

```

[slOutput_distinct_clauses]

```

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

```

```
[slRole_distinct_clauses]
└ PlatoonLeader ≠ PlatoonSergeant ∧ PlatoonLeader ≠ Omni ∧
    PlatoonSergeant ≠ Omni

[slState_distinct_clauses]
└ CONDUCT_ORP ≠ SECURE ∧ CONDUCT_ORP ≠ ACTIONS_IN ∧
    CONDUCT_ORP ≠ WITHDRAW ∧ CONDUCT_ORP ≠ COMPLETE ∧
    SECURE ≠ ACTIONS_IN ∧ SECURE ≠ WITHDRAW ∧ SECURE ≠ COMPLETE ∧
    ACTIONS_IN ≠ WITHDRAW ∧ ACTIONS_IN ≠ COMPLETE ∧
    WITHDRAW ≠ COMPLETE
```

10 ConductORPDef Theory

Built: 11 June 2018

Parent Theories: ConductORPType, ssm, OMNIType

10.1 Definitions

```
[secAuthorization_def]
└ ∀xs. secAuthorization xs = secHelper (getOmniCommand xs)

[secContext_def]
└ (forall xs.
    secContext CONDUCT_ORP xs =
    [Name PlatoonLeader controls
     prop (SOME (SLc (PL secure)))])) ∧
    (forall xs.
    secContext SECURE xs =
    if getOmniCommand xs = ssmSecureComplete then
        [prop (SOME (SLc (OMNI ssmSecureComplete))) impf
         Name PlatoonSergeant controls
         prop (SOME (SLc (PSG actionsIn)))]
    else [prop NONE]) ∧
    (forall xs.
    secContext ACTIONS_IN xs =
    if getOmniCommand xs = ssmActionsIncomplete then
        [prop (SOME (SLc (OMNI ssmActionsIncomplete))) impf
         Name PlatoonLeader controls
         prop (SOME (SLc (PL withdraw)))]
    else [prop NONE]) ∧
    (forall xs.
    secContext WITHDRAW xs =
    if getOmniCommand xs = ssmWithdrawComplete then
        [prop (SOME (SLc (OMNI ssmWithdrawComplete))) impf
         Name PlatoonLeader controls
         prop (SOME (SLc (PL complete)))]
    else [prop NONE])
```

[secHelper_def]

```

 $\vdash \forall cmd.$ 
  secHelper cmd =
  [Name Omni controls prop (SOME (SLc (OMNI cmd)))]

```

10.2 Theorems

[getOmniCommand_def]

```

 $\vdash (\text{getOmniCommand } [] = \text{invalidOmniCommand}) \wedge$ 
 $(\forall xs \ cmd.$ 
  getOmniCommand
  (Name Omni says prop (SOME (SLc (OMNI cmd))))::xs) =
  cmd) \wedge
```

 $(\forall xs. \text{getOmniCommand (TT::}xs) = \text{getOmniCommand } xs) \wedge$
 $(\forall xs. \text{getOmniCommand (FF::}xs) = \text{getOmniCommand } xs) \wedge$
 $(\forall xs \ v_2. \text{getOmniCommand (prop }v_2::}xs) = \text{getOmniCommand } xs) \wedge$
 $(\forall xs \ v_3. \text{getOmniCommand (notf }v_3::}xs) = \text{getOmniCommand } xs) \wedge$
 $(\forall xs \ v_5 \ v_4.$
 getOmniCommand (v₄ andf v₅::xs) = getOmniCommand xs) \wedge
 $(\forall xs \ v_7 \ v_6.$
 getOmniCommand (v₆ orf v₇::xs) = getOmniCommand xs) \wedge
 $(\forall xs \ v_9 \ v_8.$
 getOmniCommand (v₈ impf v₉::xs) = getOmniCommand xs) \wedge
 $(\forall xs \ v_{11} \ v_{10}.$
 getOmniCommand (v₁₀ eqf v₁₁::xs) = getOmniCommand xs) \wedge
 $(\forall xs \ v_{12}.$
 getOmniCommand (v₁₂ says TT::xs) = getOmniCommand xs) \wedge
 $(\forall xs \ v_{12}.$
 getOmniCommand (v₁₂ says FF::xs) = getOmniCommand xs) \wedge
 $(\forall xs \ v_{134}.$
 getOmniCommand (Name v₁₃₄ says prop NONE::xs) =
 getOmniCommand xs) \wedge
 $(\forall xs \ v_{144}.$
 getOmniCommand
 (Name PlatoonLeader says prop (SOME v₁₄₄)::xs) =
 getOmniCommand xs) \wedge
 $(\forall xs \ v_{144}.$
 getOmniCommand
 (Name PlatoonSergeant says prop (SOME v₁₄₄)::xs) =
 getOmniCommand xs) \wedge
 $(\forall xs \ v_{146}.$
 getOmniCommand
 (Name Omni says prop (SOME (ESCc v₁₄₆))::xs) =
 getOmniCommand xs) \wedge
 $(\forall xs \ v_{150}.$
 getOmniCommand
 (Name Omni says prop (SOME (SLc (PL v₁₅₀)))::xs) =
 getOmniCommand xs) \wedge

```

(∀ xs v151 .
  getOmniCommand
    (Name Omni says prop (SOME (SLc (PSG v151))))::xs) =
  getOmniCommand xs) ∧
(∀ xs v68 v136 v135 .
  getOmniCommand (v135 meet v136 says prop v68)::xs) =
  getOmniCommand xs) ∧
(∀ xs v68 v138 v137 .
  getOmniCommand (v137 quoting v138 says prop v68)::xs) =
  getOmniCommand xs) ∧
(∀ xs v69 v12 .
  getOmniCommand (v12 says notf v69)::xs) =
  getOmniCommand xs) ∧
(∀ xs v71 v70 v12 .
  getOmniCommand (v12 says (v70 andf v71))::xs) =
  getOmniCommand xs) ∧
(∀ xs v73 v72 v12 .
  getOmniCommand (v12 says (v72 orf v73))::xs) =
  getOmniCommand xs) ∧
(∀ xs v75 v74 v12 .
  getOmniCommand (v12 says (v74 impf v75))::xs) =
  getOmniCommand xs) ∧
(∀ xs v77 v76 v12 .
  getOmniCommand (v12 says (v76 eqf v77))::xs) =
  getOmniCommand xs) ∧
(∀ xs v79 v78 v12 .
  getOmniCommand (v12 says v78 says v79)::xs) =
  getOmniCommand xs) ∧
(∀ xs v81 v80 v12 .
  getOmniCommand (v12 says v80 speaks_for v81)::xs) =
  getOmniCommand xs) ∧
(∀ xs v83 v82 v12 .
  getOmniCommand (v12 says v82 controls v83)::xs) =
  getOmniCommand xs) ∧
(∀ xs v86 v85 v84 v12 .
  getOmniCommand (v12 says reps v84 v85 v86)::xs) =
  getOmniCommand xs) ∧
(∀ xs v88 v87 v12 .
  getOmniCommand (v12 says v87 domi v88)::xs) =
  getOmniCommand xs) ∧
(∀ xs v90 v89 v12 .
  getOmniCommand (v12 says v89 eqi v90)::xs) =
  getOmniCommand xs) ∧
(∀ xs v92 v91 v12 .
  getOmniCommand (v12 says v91 doms v92)::xs) =
  getOmniCommand xs) ∧
(∀ xs v94 v93 v12 .
  getOmniCommand (v12 says v93 eqs v94)::xs) =
  getOmniCommand xs) ∧

```

```

(∀ xs v96 v95 v12.
  getOmniCommand (v12 says v95 eqn v96::xs) =
  getOmniCommand xs) ∧
(∀ xs v98 v97 v12.
  getOmniCommand (v12 says v97 lte v98::xs) =
  getOmniCommand xs) ∧
(∀ xs v99 v12 v100.
  getOmniCommand (v12 says v99 lt v100::xs) =
  getOmniCommand xs) ∧
(∀ xs v15 v14.
  getOmniCommand (v14 speaks_for v15::xs) =
  getOmniCommand xs) ∧
(∀ xs v17 v16.
  getOmniCommand (v16 controls v17::xs) =
  getOmniCommand xs) ∧
(∀ xs v20 v19 v18.
  getOmniCommand (reps v18 v19 v20::xs) =
  getOmniCommand xs) ∧
(∀ xs v22 v21.
  getOmniCommand (v21 domi v22::xs) = getOmniCommand xs) ∧
(∀ xs v24 v23.
  getOmniCommand (v23 eqi v24::xs) = getOmniCommand xs) ∧
(∀ xs v26 v25.
  getOmniCommand (v25 doms v26::xs) = getOmniCommand xs) ∧
(∀ xs v28 v27.
  getOmniCommand (v27 eqs v28::xs) = getOmniCommand xs) ∧
(∀ xs v30 v29.
  getOmniCommand (v29 eqn v30::xs) = getOmniCommand xs) ∧
(∀ xs v32 v31.
  getOmniCommand (v31 lte v32::xs) = getOmniCommand xs) ∧
∀ xs v34 v33.
  getOmniCommand (v33 lt v34::xs) = getOmniCommand xs

```

[getOmniCommand_ind]

```

⊢ ∀ P.
  P [] ∧
  (∀ cmd xs.
    P (Name Omni says prop (SOME (SLc (OMNI cmd)))::xs)) ∧
    (∀ xs. P xs ⇒ P (TT::xs)) ∧ (∀ xs. P xs ⇒ P (FF::xs)) ∧
    (∀ v2 xs. P xs ⇒ P (prop v2::xs)) ∧
    (∀ v3 xs. P xs ⇒ P (notf v3::xs)) ∧
    (∀ v4 v5 xs. P xs ⇒ P (v4 andf v5::xs)) ∧
    (∀ v6 v7 xs. P xs ⇒ P (v6 orf v7::xs)) ∧
    (∀ v8 v9 xs. P xs ⇒ P (v8 impf v9::xs)) ∧
    (∀ v10 v11 xs. P xs ⇒ P (v10 eqf v11::xs)) ∧
    (∀ v12 xs. P xs ⇒ P (v12 says TT::xs)) ∧
    (∀ v12 xs. P xs ⇒ P (v12 says FF::xs)) ∧
    (∀ v134 xs. P xs ⇒ P (Name v134 says prop NONE::xs)) ∧
    (∀ v144 xs.

```

```


$$\begin{aligned}
& P \text{ } xs \Rightarrow \\
& P \text{ } (\text{Name PlatoonLeader says prop (SOME } v144 :: xs)) \wedge \\
& (\forall v144 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow \\
& \quad P \text{ } (\text{Name PlatoonSergeant says prop (SOME } v144 :: xs)) \wedge \\
& (\forall v146 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (\text{Name Omni says prop (SOME (ESCc } v146)) :: xs)) \wedge \\
& (\forall v150 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow \\
& \quad P \text{ } (\text{Name Omni says prop (SOME (SLc (PL } v150))) :: xs)) \wedge \\
& (\forall v151 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow \\
& \quad P \text{ } (\text{Name Omni says prop (SOME (SLc (PSG } v151))) :: xs)) \wedge \\
& (\forall v135 \text{ } v136 \text{ } v68 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v135 \text{ } \text{meet } v136 \text{ } \text{says prop } v68 :: xs)) \wedge \\
& (\forall v137 \text{ } v138 \text{ } v68 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v137 \text{ } \text{quoting } v138 \text{ } \text{says prop } v68 :: xs)) \wedge \\
& (\forall v12 \text{ } v69 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says notf } v69 :: xs)) \wedge \\
& (\forall v12 \text{ } v70 \text{ } v71 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says (v70 andf } v71) :: xs)) \wedge \\
& (\forall v12 \text{ } v72 \text{ } v73 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says (v72 orf } v73) :: xs)) \wedge \\
& (\forall v12 \text{ } v74 \text{ } v75 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says (v74 impf } v75) :: xs)) \wedge \\
& (\forall v12 \text{ } v76 \text{ } v77 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says (v76 eqf } v77) :: xs)) \wedge \\
& (\forall v12 \text{ } v78 \text{ } v79 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v78 says v79} :: xs)) \wedge \\
& (\forall v12 \text{ } v80 \text{ } v81 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v80 speaks_for v81} :: xs)) \wedge \\
& (\forall v12 \text{ } v82 \text{ } v83 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v82 controls v83} :: xs)) \wedge \\
& (\forall v12 \text{ } v84 \text{ } v85 \text{ } v86 \text{ } xs. \\
& \quad P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says reps v84 v85 v86} :: xs)) \wedge \\
& (\forall v12 \text{ } v87 \text{ } v88 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v87 domi v88} :: xs)) \wedge \\
& (\forall v12 \text{ } v89 \text{ } v90 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v89 eqi v90} :: xs)) \wedge \\
& (\forall v12 \text{ } v91 \text{ } v92 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v91 doms v92} :: xs)) \wedge \\
& (\forall v12 \text{ } v93 \text{ } v94 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v93 eqs v94} :: xs)) \wedge \\
& (\forall v12 \text{ } v95 \text{ } v96 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v95 eqn v96} :: xs)) \wedge \\
& (\forall v12 \text{ } v97 \text{ } v98 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v97 lte v98} :: xs)) \wedge \\
& (\forall v12 \text{ } v99 \text{ } v100 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v12 \text{ } \text{says v99 lt v100} :: xs)) \wedge \\
& (\forall v14 \text{ } v15 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v14 \text{ } \text{speaks_for v15} :: xs)) \wedge \\
& (\forall v16 \text{ } v17 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v16 \text{ } \text{controls v17} :: xs)) \wedge \\
& (\forall v18 \text{ } v19 \text{ } v20 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (\text{reps v18 v19 v20} :: xs)) \wedge \\
& (\forall v21 \text{ } v22 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v21 \text{ } \text{domi v22} :: xs)) \wedge \\
& (\forall v23 \text{ } v24 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v23 \text{ } \text{eqi v24} :: xs)) \wedge \\
& (\forall v25 \text{ } v26 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v25 \text{ } \text{doms v26} :: xs)) \wedge \\
& (\forall v27 \text{ } v28 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v27 \text{ } \text{eqs v28} :: xs)) \wedge \\
& (\forall v29 \text{ } v30 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v29 \text{ } \text{eqn v30} :: xs)) \wedge \\
& (\forall v31 \text{ } v32 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v31 \text{ } \text{lte v32} :: xs)) \wedge \\
& (\forall v33 \text{ } v34 \text{ } xs. P \text{ } xs \Rightarrow P \text{ } (v33 \text{ } \text{lt v34} :: xs)) \Rightarrow \\
& \forall v. \text{ } P \text{ } v
\end{aligned}$$


```

[getPlCom_def]

```

 $\vdash (\text{getPlCom} [] = \text{plIncomplete}) \wedge$ 
 $(\forall xs \ cmd. \text{getPlCom} (\text{SOME} (\text{SLc} (\text{PL} \ cmd)))::xs) = cmd) \wedge$ 
 $(\forall xs. \text{getPlCom} (\text{NONE}::xs) = \text{getPlCom} \ xs) \wedge$ 
 $(\forall xs \ v_4. \text{getPlCom} (\text{SOME} (\text{ESCC} \ v_4)::xs) = \text{getPlCom} \ xs) \wedge$ 
 $(\forall xs \ v_9. \text{getPlCom} (\text{SOME} (\text{SLc} (\text{PSG} \ v_9)))::xs) = \text{getPlCom} \ xs) \wedge$ 
 $\forall xs \ v_{10}. \text{getPlCom} (\text{SOME} (\text{SLc} (\text{OMNI} \ v_{10})))::xs) = \text{getPlCom} \ xs$ 

```

[getPlCom_ind]

```

 $\vdash \forall P.$ 
 $P [] \wedge (\forall cmd \ xs. \text{P} (\text{SOME} (\text{SLc} (\text{PL} \ cmd)))::xs)) \wedge$ 
 $(\forall xs. \text{P} \ xs \Rightarrow \text{P} (\text{NONE}::xs)) \wedge$ 
 $(\forall v_4 \ xs. \text{P} \ xs \Rightarrow \text{P} (\text{SOME} (\text{ESCC} \ v_4)::xs)) \wedge$ 
 $(\forall v_9 \ xs. \text{P} \ xs \Rightarrow \text{P} (\text{SOME} (\text{SLc} (\text{PSG} \ v_9)))::xs)) \wedge$ 
 $(\forall v_{10} \ xs. \text{P} \ xs \Rightarrow \text{P} (\text{SOME} (\text{SLc} (\text{OMNI} \ v_{10})))::xs)) \Rightarrow$ 
 $\forall v. \text{P} \ v$ 

```

[getPsgCom_def]

```

 $\vdash (\text{getPsgCom} [] = \text{psgIncomplete}) \wedge$ 
 $(\forall xs \ cmd. \text{getPsgCom} (\text{SOME} (\text{SLc} (\text{PSG} \ cmd)))::xs) = cmd) \wedge$ 
 $(\forall xs. \text{getPsgCom} (\text{NONE}::xs) = \text{getPsgCom} \ xs) \wedge$ 
 $(\forall xs \ v_4. \text{getPsgCom} (\text{SOME} (\text{ESCC} \ v_4)::xs) = \text{getPsgCom} \ xs) \wedge$ 
 $(\forall xs \ v_8. \text{getPsgCom} (\text{SOME} (\text{SLc} (\text{PL} \ v_8)))::xs) = \text{getPsgCom} \ xs) \wedge$ 
 $\forall xs \ v_{10}. \text{getPsgCom} (\text{SOME} (\text{SLc} (\text{OMNI} \ v_{10})))::xs) = \text{getPsgCom} \ xs$ 

```

[getPsgCom_ind]

```

 $\vdash \forall P.$ 
 $P [] \wedge (\forall cmd \ xs. \text{P} (\text{SOME} (\text{SLc} (\text{PSG} \ cmd)))::xs)) \wedge$ 
 $(\forall xs. \text{P} \ xs \Rightarrow \text{P} (\text{NONE}::xs)) \wedge$ 
 $(\forall v_4 \ xs. \text{P} \ xs \Rightarrow \text{P} (\text{SOME} (\text{ESCC} \ v_4)::xs)) \wedge$ 
 $(\forall v_8 \ xs. \text{P} \ xs \Rightarrow \text{P} (\text{SOME} (\text{SLc} (\text{PL} \ v_8)))::xs)) \wedge$ 
 $(\forall v_{10} \ xs. \text{P} \ xs \Rightarrow \text{P} (\text{SOME} (\text{SLc} (\text{OMNI} \ v_{10})))::xs)) \Rightarrow$ 
 $\forall v. \text{P} \ v$ 

```

11 ssmConductPB Theory

Built: 10 June 2018

Parent Theories: ConductPBType, ssm11, OMNIType

11.1 Definitions

[secContextConductPB_def]

```

 $\vdash \forall plcmd \ psgcmd \ incomplete.$ 
 $\text{secContextConductPB} \ plcmd \ psgcmd \ incomplete =$ 
 $[\text{Name} \ \text{PlatoonLeader} \ \text{controls} \ \text{prop} \ (\text{SOME} (\text{SLc} (\text{PL} \ plcmd)));$ 
 $\text{Name} \ \text{PlatoonSergeant} \ \text{controls}$ 
 $\text{prop} \ (\text{SOME} (\text{SLc} (\text{PSG} \ psgcmd)));$ 
 $\text{Name} \ \text{PlatoonLeader} \ \text{says}$ 

```

```

prop (SOME (SLc (PSG psgcmd))) impf prop NONE;
Name PlatoonSergeant says
prop (SOME (SLc (PL plcmsg))) impf prop NONE]

```

[ssmConductPBStateInterp_def]

```

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

```

11.2 Theorems

[authTestConductPB_cmd_reject_lemma]

```

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

```

[authTestConductPB_def]

```

 $\vdash (\text{authTestConductPB} (\text{Name PlatoonLeader says prop } cmd) \iff \text{TT}) \wedge$ 
 $(\text{authTestConductPB} (\text{Name PlatoonSergeant says prop } cmd) \iff \text{TT}) \wedge$ 
 $(\text{authTestConductPB} \text{ TT} \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} \text{ FF} \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{prop } v) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{notf } v_1) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{andf } v_2 \text{ andf } v_3) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{orf } v_4 \text{ orf } v_5) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{impf } v_6 \text{ impf } v_7) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{eqf } v_8 \text{ eqf } v_9) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says TT}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says FF}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v133 meet v134 says prop } v_{66}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v135 quoting v136 says prop } v_{66}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says notf } v_{67}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says (v68 andf v69)}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says (v70 orf v71)}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says (v72 impf v73)}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says (v74 eqf v75)}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v76 says v77}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v78 speaks_for v79}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v80 controls v81}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says reps v82 v83 v84}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v85 domi v86}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v87 eqi v88}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v89 doms v90}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v91 eqs v92}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v93 eqn v94}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v95 lte v96}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v10 says v97 lt v98}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v12 speaks_for v13}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v14 controls v15}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{reps v16 v17 v18}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v19 domi v20}) \iff \text{F}) \wedge$ 
 $(\text{authTestConductPB} (\text{v21 eqi v22}) \iff \text{F}) \wedge$ 

```

```
(authTestConductPB (v23 doms v24)  $\iff$  F)  $\wedge$ 
(authTestConductPB (v25 eqs v26)  $\iff$  F)  $\wedge$ 
(authTestConductPB (v27 eqn v28)  $\iff$  F)  $\wedge$ 
(authTestConductPB (v29 lte v30)  $\iff$  F)  $\wedge$ 
(authTestConductPB (v31 lt v32)  $\iff$  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 v1))  $\wedge$ 
( $\forall v_2 v_3. P$  (v2 andf v3))  $\wedge$  ( $\forall v_4 v_5. P$  (v4 orf v5))  $\wedge$ 
( $\forall v_6 v_7. P$  (v6 impf v7))  $\wedge$  ( $\forall v_8 v_9. P$  (v8 eqf v9))  $\wedge$ 
( $\forall v_{10}. P$  (v10 says TT))  $\wedge$  ( $\forall v_{10}. P$  (v10 says FF))  $\wedge$ 
( $\forall v_{133} v_{134} v_{66}. P$  (v133 meet v134 says prop v66))  $\wedge$ 
( $\forall v_{135} v_{136} v_{66}. P$  (v135 quoting v136 says prop v66))  $\wedge$ 
( $\forall v_{10} v_{67}. P$  (v10 says notf v67))  $\wedge$ 
( $\forall v_{10} v_{68} v_{69}. P$  (v10 says (v68 andf v69)))  $\wedge$ 
( $\forall v_{10} v_{70} v_{71}. P$  (v10 says (v70 orf v71)))  $\wedge$ 
( $\forall v_{10} v_{72} v_{73}. P$  (v10 says (v72 impf v73)))  $\wedge$ 
( $\forall v_{10} v_{74} v_{75}. P$  (v10 says (v74 eqf v75)))  $\wedge$ 
( $\forall v_{10} v_{76} v_{77}. P$  (v10 says v76 says v77))  $\wedge$ 
( $\forall v_{10} v_{78} v_{79}. P$  (v10 says v78 speaks_for v79))  $\wedge$ 
( $\forall v_{10} v_{80} v_{81}. P$  (v10 says v80 controls v81))  $\wedge$ 
( $\forall v_{10} v_{82} v_{83} v_{84}. P$  (v10 says reps v82 v83 v84))  $\wedge$ 
( $\forall v_{10} v_{85} v_{86}. P$  (v10 says v85 doms v86))  $\wedge$ 
( $\forall v_{10} v_{87} v_{88}. P$  (v10 says v87 eqi v88))  $\wedge$ 
( $\forall v_{10} v_{89} v_{90}. P$  (v10 says v89 doms v90))  $\wedge$ 
( $\forall v_{10} v_{91} v_{92}. P$  (v10 says v91 eqs v92))  $\wedge$ 
( $\forall v_{10} v_{93} v_{94}. P$  (v10 says v93 eqn v94))  $\wedge$ 
( $\forall v_{10} v_{95} v_{96}. P$  (v10 says v95 lte v96))  $\wedge$ 
( $\forall v_{10} v_{97} v_{98}. P$  (v10 says v97 lt v98))  $\wedge$ 
( $\forall v_{12} v_{13}. P$  (v12 speaks_for v13))  $\wedge$ 
( $\forall v_{14} v_{15}. P$  (v14 controls v15))  $\wedge$ 
( $\forall v_{16} v_{17} v_{18}. P$  (reps v16 v17 v18))  $\wedge$ 
( $\forall v_{19} v_{20}. P$  (v19 domi v20))  $\wedge$ 
( $\forall v_{21} v_{22}. P$  (v21 eqi v22))  $\wedge$ 
( $\forall v_{23} v_{24}. P$  (v23 doms v24))  $\wedge$ 
( $\forall v_{25} v_{26}. P$  (v25 eqs v26))  $\wedge$  ( $\forall v_{27} v_{28}. P$  (v27 eqn v28))  $\wedge$ 
( $\forall v_{29} v_{30}. P$  (v29 lte v30))  $\wedge$  ( $\forall v_{31} v_{32}. P$  (v31 lt v32))  $\Rightarrow$ 
 $\forall v. P v$ 
```

[conductPBNS_def]

\vdash (conductPBNS CONDUCT_PB (exec (PL securePB)) = SECURE_PB) \wedge
(conductPBNS CONDUCT_PB (exec (PL plIncompletePB)) =
CONDUCT_PB) \wedge
(conductPBNS SECURE_PB (exec (PSG actionsInPB)) =
ACTIONS_IN_PB) \wedge
(conductPBNS SECURE_PB (exec (PSG psgIncompletePB)) =

```

SECURE_PB) ∧
(conductPBNS ACTIONS_IN_PB (exec (PL withdrawPB)) =
WITHDRAW_PB) ∧
(conductPBNS ACTIONS_IN_PB (exec (PL p1IncompletePB)) =
ACTIONS_IN_PB) ∧
(conductPBNS WITHDRAW_PB (exec (PL completePB)) =
COMPLETE_PB) ∧
(conductPBNS WITHDRAW_PB (exec (PL p1IncompletePB)) =
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]

$$\begin{aligned}
&\vdash \forall P. \\
&P \text{ CONDUCT_PB } (\text{exec } (\text{PL securePB})) \wedge \\
&P \text{ CONDUCT_PB } (\text{exec } (\text{PL p1IncompletePB})) \wedge \\
&P \text{ SECURE_PB } (\text{exec } (\text{PSG actionsInPB})) \wedge \\
&P \text{ SECURE_PB } (\text{exec } (\text{PSG psgIncompletePB})) \wedge \\
&P \text{ ACTIONS_IN_PB } (\text{exec } (\text{PL withdrawPB})) \wedge \\
&P \text{ ACTIONS_IN_PB } (\text{exec } (\text{PL p1IncompletePB})) \wedge \\
&P \text{ WITHDRAW_PB } (\text{exec } (\text{PL completePB})) \wedge \\
&P \text{ WITHDRAW_PB } (\text{exec } (\text{PL p1IncompletePB})) \wedge \\
&(\forall s \ cmd. P \ s \ (\text{trap } (\text{PL cmd}))) \wedge \\
&(\forall s \ cmd. P \ s \ (\text{trap } (\text{PSG cmd}))) \wedge \\
&(\forall s \ cmd. P \ s \ (\text{discard } (\text{PL cmd}))) \wedge \\
&(\forall s \ cmd. P \ s \ (\text{discard } (\text{PSG cmd}))) \wedge \\
&P \text{ CONDUCT_PB } (\text{exec } (\text{PL withdrawPB})) \wedge \\
&P \text{ CONDUCT_PB } (\text{exec } (\text{PL completePB})) \wedge \\
&(\forall v_{11}. P \text{ CONDUCT_PB } (\text{exec } (\text{PSG } v_{11}))) \wedge \\
&(\forall v_{13}. P \text{ SECURE_PB } (\text{exec } (\text{PL } v_{13}))) \wedge \\
&P \text{ ACTIONS_IN_PB } (\text{exec } (\text{PL securePB})) \wedge \\
&P \text{ ACTIONS_IN_PB } (\text{exec } (\text{PL completePB})) \wedge \\
&(\forall v_{17}. P \text{ ACTIONS_IN_PB } (\text{exec } (\text{PSG } v_{17}))) \wedge \\
&P \text{ WITHDRAW_PB } (\text{exec } (\text{PL securePB})) \wedge \\
&P \text{ WITHDRAW_PB } (\text{exec } (\text{PL withdrawPB})) \wedge \\
&(\forall v_{20}. P \text{ WITHDRAW_PB } (\text{exec } (\text{PSG } v_{20}))) \wedge \\
&(\forall v_{21}. P \text{ COMPLETE_PB } (\text{exec } v_{21})) \Rightarrow \\
&\forall v \ v_1. P \ v \ v_1
\end{aligned}$$

[conductPBOut_def]

$$\begin{aligned}
&\vdash (\text{conductPBOut CONDUCT_PB } (\text{exec } (\text{PL securePB})) = \text{ConductPB}) \wedge \\
&(\text{conductPBOut CONDUCT_PB } (\text{exec } (\text{PL p1IncompletePB})) = \\
&\text{ConductPB}) \wedge \\
&(\text{conductPBOut SECURE_PB } (\text{exec } (\text{PSG actionsInPB})) = \\
&\text{SecurePB}) \wedge \\
&(\text{conductPBOut SECURE_PB } (\text{exec } (\text{PSG psgIncompletePB})) = \\
&\text{SecurePB}) \wedge \\
&(\text{conductPBOut ACTIONS_IN_PB } (\text{exec } (\text{PL withdrawPB})) =
\end{aligned}$$

```

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]

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

[PlatoonLeader_exec_plCommandPB_justified_thm]

$$\begin{aligned}
&\vdash \forall NS \ Out \ M \ Oi \ Os. \\
&\text{TR } (M, Oi, Os) \ (\text{exec } (\text{SLc } (\text{PL } plCommand))) \\
&\quad (\text{CFG authTestConductPB ssmConductPBStateInterp} \\
&\quad \quad (\text{secContextConductPB } plCommand \ psgCommand \ incomplete) \\
&\quad \quad (\text{Name PlatoonLeader says} \\
&\quad \quad \quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL } plCommand)))::ins \ s \ outs) \\
&\quad (\text{CFG authTestConductPB ssmConductPBStateInterp} \\
&\quad \quad (\text{secContextConductPB } plCommand \ psgCommand \ incomplete) \\
&\quad \quad ins \ (NS \ s \ (\text{exec } (\text{SLc } (\text{PL } plCommand))))))
\end{aligned}$$

```

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

```

[PlatoonLeader_plCommandPB_lemma]

```

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

```

[PlatoonSergeant_exec_psgCommandPB_justified_thm]

```

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

```

[PlatoonSergeant_psgCommandPB_lemma]

```

 $\vdash$  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)))

```

12 ConductPBType Theory

Built: 10 June 2018

Parent Theories: indexedLists, patternMatches

12.1 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
```

12.2 Theorems

[plCommandPB_distinct_clauses]

```
⊤ securePB ≠ withdrawPB ∧ securePB ≠ completePB ∧  
securePB ≠ plIncompletePB ∧ withdrawPB ≠ completePB ∧  
withdrawPB ≠ plIncompletePB ∧ completePB ≠ plIncompletePB
```

[psgCommandPB_distinct_clauses]

```
⊤ actionsInPB ≠ psgIncompletePB
```

[slCommand_distinct_clauses]

```
⊤ ∀ a' a. PL a ≠ PSG a'
```

[slCommand_one_one]

```
⊤ (∀ a a'. (PL a = PL a') ⇔ (a = a')) ∧  
∀ a a'. (PSG a = PSG a') ⇔ (a = a')
```

[slOutput_distinct_clauses]

```
⊤ ConductPB ≠ SecurePB ∧ ConductPB ≠ ActionsInPB ∧  
ConductPB ≠ WithdrawPB ∧ ConductPB ≠ CompletePB ∧  
ConductPB ≠ unAuthenticated ∧ ConductPB ≠ unAuthorized ∧  
SecurePB ≠ ActionsInPB ∧ SecurePB ≠ WithdrawPB ∧  
SecurePB ≠ CompletePB ∧ SecurePB ≠ unAuthenticated ∧  
SecurePB ≠ unAuthorized ∧ ActionsInPB ≠ WithdrawPB ∧  
ActionsInPB ≠ CompletePB ∧ ActionsInPB ≠ unAuthenticated ∧  
ActionsInPB ≠ unAuthorized ∧ WithdrawPB ≠ CompletePB ∧  
WithdrawPB ≠ unAuthenticated ∧ WithdrawPB ≠ unAuthorized ∧  
CompletePB ≠ unAuthenticated ∧ CompletePB ≠ unAuthorized ∧  
unAuthenticated ≠ unAuthorized
```

```
[slRole_distinct_clauses]
  ⊢ PlatoonLeader ≠ PlatoonSergeant

[slState_distinct_clauses]
  ⊢ CONDUCT_PB ≠ SECURE_PB ∧ CONDUCT_PB ≠ ACTIONS_IN_PB ∧
    CONDUCT_PB ≠ WITHDRAW_PB ∧ CONDUCT_PB ≠ COMPLETE_PB ∧
    SECURE_PB ≠ ACTIONS_IN_PB ∧ SECURE_PB ≠ WITHDRAW_PB ∧
    SECURE_PB ≠ COMPLETE_PB ∧ ACTIONS_IN_PB ≠ WITHDRAW_PB ∧
    ACTIONS_IN_PB ≠ COMPLETE_PB ∧ WITHDRAW_PB ≠ COMPLETE_PB
```

13 ssmMoveToORP Theory

Built: 10 June 2018

Parent Theories: MoveToORPType, ssm11, OMNIType

13.1 Definitions

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

```
[ssmMoveToORPStateInterp_def]
  ⊢ ∀ state. ssmMoveToORPStateInterp state = TT
```

13.2 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 v1) ⇔ F) ∧
    (authTestMoveToORP (v2 andf v3) ⇔ F) ∧
    (authTestMoveToORP (v4 orf v5) ⇔ F) ∧
    (authTestMoveToORP (v6 impf v7) ⇔ F) ∧
    (authTestMoveToORP (v8 eqf v9) ⇔ F) ∧
    (authTestMoveToORP (v10 says TT) ⇔ F) ∧
    (authTestMoveToORP (v10 says FF) ⇔ F) ∧
    (authTestMoveToORP (v133 meet v134 says prop v66) ⇔ F) ∧
    (authTestMoveToORP (v135 quoting v136 says prop v66) ⇔ F) ∧
    (authTestMoveToORP (v10 says notf v67) ⇔ F) ∧
    (authTestMoveToORP (v10 says (v68 andf v69)) ⇔ F) ∧
    (authTestMoveToORP (v10 says (v70 orf v71)) ⇔ F) ∧
```

```

(authTestMoveToORP (v10 says (v72 impf v73))  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says (v74 eqf v75))  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v76 says v77)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v78 speaks_for v79)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v80 controls v81)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says reps v82 v83 v84)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v85 domi v86)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v87 equi v88)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v89 doms v90)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v91 eqs v92)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v93 eqn v94)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v95 lte v96)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v10 says v97 lt v98)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v12 speaks_for v13)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v14 controls v15)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (reps v16 v17 v18)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v19 domi v20)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v21 equi v22)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v23 doms v24)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v25 eqs v26)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v27 eqn v28)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v29 lte v30)  $\iff$  F)  $\wedge$ 
(authTestMoveToORP (v31 lt v32)  $\iff$  F)

```

[authTestMoveToORP_ind]

$\vdash \forall P.$

$$\begin{aligned}
& (\forall cmd. P (\text{Name PlatoonLeader says prop } cmd)) \wedge P \text{ TT} \wedge \\
& P \text{ FF} \wedge (\forall v. P (\text{prop } v)) \wedge (\forall v_1. P (\text{notf } v_1)) \wedge \\
& (\forall v_2 v_3. P (v_2 \text{ andf } v_3)) \wedge (\forall v_4 v_5. P (v_4 \text{ orf } v_5)) \wedge \\
& (\forall v_6 v_7. P (v_6 \text{ impf } v_7)) \wedge (\forall v_8 v_9. P (v_8 \text{ eqf } v_9)) \wedge \\
& (\forall v_{10}. P (v_{10} \text{ says TT})) \wedge (\forall v_{10}. P (v_{10} \text{ says FF})) \wedge \\
& (\forall v_{133} v_{134} v_{66}. P (v_{133} \text{ meet } v_{134} \text{ says prop } v_{66})) \wedge \\
& (\forall v_{135} v_{136} v_{66}. P (v_{135} \text{ quoting } v_{136} \text{ says prop } v_{66})) \wedge \\
& (\forall v_{10} v_{67}. P (v_{10} \text{ says notf } v_{67})) \wedge \\
& (\forall v_{10} v_{68} v_{69}. P (v_{10} \text{ says } (v_{68} \text{ andf } v_{69}))) \wedge \\
& (\forall v_{10} v_{70} v_{71}. P (v_{10} \text{ says } (v_{70} \text{ orf } v_{71}))) \wedge \\
& (\forall v_{10} v_{72} v_{73}. P (v_{10} \text{ says } (v_{72} \text{ impf } v_{73}))) \wedge \\
& (\forall v_{10} v_{74} v_{75}. P (v_{10} \text{ says } (v_{74} \text{ eqf } v_{75}))) \wedge \\
& (\forall v_{10} v_{76} v_{77}. P (v_{10} \text{ says } v_{76} \text{ says } v_{77})) \wedge \\
& (\forall v_{10} v_{78} v_{79}. P (v_{10} \text{ says } v_{78} \text{ speaks_for } v_{79})) \wedge \\
& (\forall v_{10} v_{80} v_{81}. P (v_{10} \text{ says } v_{80} \text{ controls } v_{81})) \wedge \\
& (\forall v_{10} v_{82} v_{83} v_{84}. P (v_{10} \text{ says } \text{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{ equi } 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
\end{aligned}$$

$$\begin{aligned}
 & (\forall v_{12} v_{13}. P(v_{12} \text{ speaks_for } v_{13})) \wedge \\
 & (\forall v_{14} v_{15}. P(v_{14} \text{ controls } v_{15})) \wedge \\
 & (\forall v_{16} v_{17} v_{18}. P(\text{reps } v_{16} v_{17} v_{18})) \wedge \\
 & (\forall v_{19} v_{20}. P(v_{19} \text{ domi } v_{20})) \wedge \\
 & (\forall v_{21} v_{22}. P(v_{21} \text{ eqi } v_{22})) \wedge \\
 & (\forall v_{23} v_{24}. P(v_{23} \text{ doms } v_{24})) \wedge \\
 & (\forall v_{25} v_{26}. P(v_{25} \text{ eqs } v_{26})) \wedge (\forall v_{27} v_{28}. P(v_{27} \text{ eqn } v_{28})) \wedge \\
 & (\forall v_{29} v_{30}. P(v_{29} \text{ lte } v_{30})) \wedge (\forall v_{31} v_{32}. P(v_{31} \text{ lt } v_{32})) \Rightarrow \\
 & \forall v. P v
 \end{aligned}$$

[moveToORPNS_def]

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

[moveToORPNS_ind]

$$\begin{aligned}
 \vdash & \forall P. \\
 & P \text{ MOVE_TO_ORP (exec (SLc pltForm))} \wedge \\
 & P \text{ MOVE_TO_ORP (exec (SLc incomplete))} \wedge \\
 & P \text{ PLT_FORM (exec (SLc pltMove))} \wedge \\
 & P \text{ PLT_FORM (exec (SLc incomplete))} \wedge \\
 & P \text{ PLT_MOVE (exec (SLc pltSecureHalt))} \wedge \\
 & P \text{ PLT_MOVE (exec (SLc incomplete))} \wedge \\
 & P \text{ PLT_SECURE_HALT (exec (SLc complete))} \wedge \\
 & P \text{ PLT_SECURE_HALT (exec (SLc incomplete))} \wedge \\
 & (\forall s \text{ cmd. } P s \text{ (trap (SLc cmd))}) \wedge \\
 & (\forall s \text{ cmd. } P s \text{ (discard (SLc cmd))}) \wedge \\
 & (\forall s v_6. P s \text{ (discard (ESCc } v_6\text{)))} \wedge \\
 & (\forall s v_9. P s \text{ (trap (ESCc } v_9\text{)))} \wedge \\
 & (\forall v_{12}. P \text{ MOVE_TO_ORP (exec (ESCc } v_{12}\text{)))} \wedge \\
 & P \text{ MOVE_TO_ORP (exec (SLc pltMove))} \wedge \\
 & P \text{ MOVE_TO_ORP (exec (SLc pltSecureHalt))} \wedge \\
 & P \text{ MOVE_TO_ORP (exec (SLc complete))} \wedge \\
 & (\forall v_{15}. P \text{ PLT_FORM (exec (ESCc } v_{15}\text{)))} \wedge \\
 & P \text{ PLT_FORM (exec (SLc pltForm))} \wedge \\
 & P \text{ PLT_FORM (exec (SLc pltSecureHalt))} \wedge \\
 & P \text{ PLT_FORM (exec (SLc complete))} \wedge \\
 & (\forall v_{18}. P \text{ PLT_MOVE (exec (ESCc } v_{18}\text{)))} \wedge \\
 & P \text{ PLT_MOVE (exec (SLc pltForm))} \wedge
 \end{aligned}$$

```

P PLT_MOVE (exec (SLc pltMove)) ∧
P PLT_MOVE (exec (SLc complete)) ∧
(∀ v21. P PLT_SECURE_HALTI (exec (ESCc v21))) ∧
P PLT_SECURE_HALTI (exec (SLc pltForm)) ∧
P PLT_SECURE_HALTI (exec (SLc pltMove)) ∧
P PLT_SECURE_HALTI (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_HALTI (exec (SLc complete)) =
 Complete) ∧
(moveToORPOut PLT_SECURE_HALTI (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_HALTI (exec (SLc complete)) ∧
P PLT_SECURE_HALTI (exec (SLc incomplete)) ∧
(∀ s cmd. P s (trap (SLc cmd))) ∧
(∀ s cmd. P s (discard (SLc cmd))) ∧
(∀ s v6. P s (discard (ESCc v6))) ∧
(∀ s v9. P s (trap (ESCc v9))) ∧
(∀ v12. P MOVE_TO_ORP (exec (ESCc v12))) ∧
P MOVE_TO_ORP (exec (SLc pltMove)) ∧
P MOVE_TO_ORP (exec (SLc pltSecureHalt)) ∧
P MOVE_TO_ORP (exec (SLc complete)) ∧
(∀ v15. P PLT_FORM (exec (ESCc v15))) ∧
P PLT_FORM (exec (SLc pltForm)) ∧
P PLT_FORM (exec (SLc pltSecureHalt)) ∧
P PLT_FORM (exec (SLc complete)) ∧
(∀ v18. P PLT_MOVE (exec (ESCc v18))) ∧
P PLT_MOVE (exec (SLc pltForm)) ∧

```

```

P PLT_MOVE (exec (SLc pltMove)) ∧
P PLT_MOVE (exec (SLc complete)) ∧
(∀ v21. P PLT_SECURE_HALT (exec (ESCc v21))) ∧
P PLT_SECURE_HALT (exec (SLc pltForm)) ∧
P PLT_SECURE_HALT (exec (SLc pltMove)) ∧
P PLT_SECURE_HALT (exec (SLc pltSecureHalt)) ∧
(∀ v23. P COMPLETE (exec v23)) ⇒
∀ v v1. P v v1

```

[PlatoonLeader_exec_slCommand_justified_thm]

```

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

```

[PlatoonLeader_slCommand_lemma]

```

⊢ CFGInterpret (M, Oi, Os)
  (CFG authTestMoveToORP ssmMoveToORPStateInterp
    (secContextMoveToORP slCommand)
    (Name PlatoonLeader says prop (SOME (SLc slCommand))::ins) s outs) ⇒
  (M, Oi, Os) sat prop (SOME (SLc slCommand))

```

14 MoveToORPType Theory

Built: 10 June 2018

Parent Theories: indexedLists, patternMatches

14.1 Datatypes

```

slCommand = pltForm | pltMove | pltSecureHalt | complete
  | incomplete

```

```

 $slOutput = \text{MoveToORP} \mid \text{PLTForm} \mid \text{PLTMove} \mid \text{PLTSecureHalt}$ 
 $\mid \text{Complete} \mid \text{unAuthorized} \mid \text{unAuthenticated}$ 
 $slState = \text{MOVE\_TO\_ORP} \mid \text{PLT\_FORM} \mid \text{PLT\_MOVE} \mid \text{PLT\_SECURE\_HALT}$ 
 $\mid \text{COMPLETE}$ 
 $stateRole = \text{PlatoonLeader}$ 

```

14.2 Theorems

[s1Command_distinct_clauses]

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

[s1Output_distinct_clauses]

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

[s1State_distinct_clauses]

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

15 ssmMoveToPB Theory

Built: 10 June 2018

Parent Theories: MoveToPBType, ssm11, OMNIType

15.1 Definitions

[secContextMoveToPB_def]

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

[`ssmMoveToPBStateInterp_def`]

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

15.2 Theorems

[`authTestMoveToPB_cmd_reject_lemma`]

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

[`authTestMoveToPB_def`]

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

[authTestMoveToPB_ind]

$\vdash \forall P.$

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

[moveToPBNS_def]

$\vdash (\text{moveToPBNS MOVE_TO_PB } (\text{exec } (\text{SLc pltForm})) = \text{PLT_FORM}) \wedge$

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

[moveToPBNS_ind]

$\vdash \forall P.$

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

[moveToPBOut_def]

$\vdash (\text{moveToPBOut MOVE_TO_PB } (\text{exec } (\text{SLc pltForm})) = \text{PLTForm}) \wedge$

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

[moveToPBOut_ind]

$\vdash \forall P.$

$$P \text{ MOVE_TO_PB } (\text{exec } (\text{SLc pltForm})) \wedge$$

```

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

```

[PlatoonLeader_exec_s1Command_justified_thm]

```

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

```

[PlatoonLeader_slCommand_lemma]

```

 $\vdash \text{CFGInterpret } (M, O_i, O_s)$ 
  (CFG authTestMoveToPB ssmMoveToPBStateInterp
   (secContextMoveToPB slCommand)
   (Name PlatoonLeader says prop (SOME (SLc slCommand)):::
    ins) s outs)  $\Rightarrow$ 
  (M, O_i, O_s) sat prop (SOME (SLc slCommand))

```

16 MoveToPBType Theory

Built: 10 June 2018

Parent Theories: indexedLists, patternMatches

16.1 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

```

16.2 Theorems

[slCommand_distinct_clauses]

```

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

```

[slOutput_distinct_clauses]

```

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

```

[s1State_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
```

17 ssmPlanPB Theory

Built: 10 June 2018

Parent Theories: PlanPBDef, ssm

17.1 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 v1) ⇔ F) ∧  
(inputOK (v2 andf v3) ⇔ F) ∧ (inputOK (v4 orf v5) ⇔ F) ∧  
(inputOK (v6 impf v7) ⇔ F) ∧ (inputOK (v8 eqf v9) ⇔ F) ∧  
(inputOK (v10 says TT) ⇔ F) ∧ (inputOK (v10 says FF) ⇔ F) ∧  
(inputOK (v133 meet v134 says prop v66) ⇔ F) ∧  
(inputOK (v135 quoting v136 says prop v66) ⇔ F) ∧  
(inputOK (v10 says notf v67) ⇔ F) ∧  
(inputOK (v10 says (v68 andf v69)) ⇔ F) ∧  
(inputOK (v10 says (v70 orf v71)) ⇔ F) ∧  
(inputOK (v10 says (v72 impf v73)) ⇔ F) ∧  
(inputOK (v10 says (v74 eqf v75)) ⇔ F) ∧  
(inputOK (v10 says v76 says v77) ⇔ F) ∧  
(inputOK (v10 says v78 speaks_for v79) ⇔ F) ∧  
(inputOK (v10 says v80 controls v81) ⇔ F) ∧  
(inputOK (v10 says reps v82 v83 v84) ⇔ F) ∧  
(inputOK (v10 says v85 domi v86) ⇔ F) ∧  
(inputOK (v10 says v87 equi v88) ⇔ F) ∧  
(inputOK (v10 says v89 doms v90) ⇔ F) ∧  
(inputOK (v10 says v91 eqs v92) ⇔ F) ∧  
(inputOK (v10 says v93 eqn v94) ⇔ F) ∧  
(inputOK (v10 says v95 lte v96) ⇔ F) ∧  
(inputOK (v10 says v97 lt v98) ⇔ F) ∧  
(inputOK (v12 speaks_for v13) ⇔ F) ∧  
(inputOK (v14 controls v15) ⇔ F) ∧  
(inputOK (reps v16 v17 v18) ⇔ F) ∧  
(inputOK (v19 domi v20) ⇔ F) ∧  
(inputOK (v21 equi v22) ⇔ F) ∧  
(inputOK (v23 doms v24) ⇔ F) ∧
```

$$(\text{inputOK } (v_{25} \text{ eqs } v_{26}) \iff F) \wedge (\text{inputOK } (v_{27} \text{ eqn } v_{28}) \iff F) \wedge \\ (\text{inputOK } (v_{29} \text{ lte } v_{30}) \iff F) \wedge (\text{inputOK } (v_{31} \text{ lt } v_{32}) \iff F)$$

[**inputOK_ind**]

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

[**planPBNS_def**]

$$\vdash (\text{planPBNS WARNO (exec } x) = \\ \text{if} \\ (\text{getRecon } x = [\text{SOME (SLc (PL recon))}] \wedge \\ (\text{getTenativePlan } x = [\text{SOME (SLc (PL tentativePlan))}] \wedge \\ (\text{getReport } x = [\text{SOME (SLc (PL report1))}] \wedge \\ (\text{getInitMove } x = [\text{SOME (SLc (PSG initiateMovement))}])) \\ \text{then} \\ \quad \text{REPORT1} \\ \text{else WARNO}) \wedge$$

```

(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 v0) = s)  $\wedge$  (planPBNS s (discard v1) = s)

```

[planPBNS_ind]

$\vdash \forall P.$

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

[planPBOut_def]

$\vdash (\text{planPBOut WARNO (exec } x) =$

if

$$\begin{aligned} & (\text{getRecon } x = [\text{SOME (SLc (PL recon))}]) \wedge \\ & (\text{getTenativePlan } x = [\text{SOME (SLc (PL tentativePlan))}]) \wedge \\ & (\text{getReport } x = [\text{SOME (SLc (PL report1))}]) \wedge \\ & (\text{getInitMove } x = [\text{SOME (SLc (PSG initiateMovement))}]) \end{aligned}$$

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) ∧
(planPBOut OPOID (exec x) =
  if getPlCom x = supervise then Supervise
  else unAuthorized) ∧
(planPBOut SUPERVISE (exec x) =
  if getPlCom x = report2 then Report2 else unAuthorized) ∧
(planPBOut REPORT2 (exec x) =
  if getPlCom x = complete then Complete else unAuthorized) ∧
(planPBOut s (trap v0) = unAuthorized) ∧
(planPBOut s (discard v1) = unAuthenticated)
```

[planPBOut_ind]

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

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma]

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

```

        outs))  $\iff$ 
authenticationTest inputOK
  [Name PlatoonLeader says
    prop (SOME (SLc (PL plCommand)))]  $\wedge$ 
CFGInterpret (M, Oi, Os)
  (CFG inputOK secContext secContextNull
    ([Name PlatoonLeader says
      prop (SOME (SLc (PL plCommand)))])::ins) s outs)  $\wedge$ 
(M, Oi, Os) satList
propCommandList
  [Name PlatoonLeader says
    prop (SOME (SLc (PL plCommand)))]

```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]

```

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

```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma]

```

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

```

[PlatoonLeader_psgCommand_notDiscard_thm]

$\vdash \forall NS\ Out\ M\ Oi\ Os.$
 $\neg \text{TR} (M, Oi, Os) (\text{discard} [\text{SOME} (\text{SLc} (\text{PSG} \ psgCommand))])$
 $(\text{CFG} \ \text{inputOK} \ \text{secContext} \ \text{secContextNull}$
 $([\text{Name} \ \text{PlatoonLeader} \ \text{says}$
 $\text{prop} (\text{SOME} (\text{SLc} (\text{PSG} \ psgCommand)))) :: ins) \ s \ outs)$
 $(\text{CFG} \ \text{inputOK} \ \text{secContext} \ \text{secContextNull} \ ins$
 $(NS \ s \ (\text{discard} [\text{SOME} (\text{SLc} (\text{PSG} \ psgCommand))]))))$
 $(Out \ s \ (\text{discard} [\text{SOME} (\text{SLc} (\text{PSG} \ psgCommand))])) ::$
 $outs))$

[PlatoonLeader_trap_psgCommand_justified_lemma]

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

[PlatoonLeader_trap_psgCommand_lemma]

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

[PlatoonLeader_WARNO_exec_report1_justified_lemma]

$$\begin{aligned}
 & \vdash \forall NS \ Out \ M \ Oi \ Os. \\
 & \quad \text{TR} (M, Oi, Os) \\
 & \quad (\text{exec} \\
 & \quad \quad (\text{inputList} \\
 & \quad \quad \quad [\text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL recon)))}; \\
 & \quad \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL tentativePlan)))}; \\
 & \quad \quad \quad \text{Name PlatoonSergeant says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PSG initiateMovement)))}; \\
 & \quad \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL report1))))})) \\
 & \quad (\text{CFG inputOK secContext secContextNull} \\
 & \quad \quad ([\text{Name PlatoonLeader says} \\
 & \quad \quad \text{prop (SOME (SLc (PL recon)))}; \\
 & \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \text{prop (SOME (SLc (PL tentativePlan)))}; \\
 & \quad \quad \text{Name PlatoonSergeant says} \\
 & \quad \quad \text{prop (SOME (SLc (PSG initiateMovement)))}; \\
 & \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \text{prop (SOME (SLc (PL report1))))] :: ins) \text{ WARNO outs}) \\
 & \quad (\text{CFG inputOK secContext secContextNull} \\
 & \quad \quad (\text{NS WARNO} \\
 & \quad \quad (\text{exec} \\
 & \quad \quad \quad (\text{inputList} \\
 & \quad \quad \quad \quad [\text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL recon)))}; \\
 & \quad \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL tentativePlan)))}; \\
 & \quad \quad \quad \text{Name PlatoonSergeant says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PSG initiateMovement)))}; \\
 & \quad \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL report1))))})) \\
 & \quad (\text{Out WARNO} \\
 & \quad \quad (\text{exec} \\
 & \quad \quad \quad (\text{inputList} \\
 & \quad \quad \quad \quad [\text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL recon)))}; \\
 & \quad \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL tentativePlan)))}; \\
 & \quad \quad \quad \text{Name PlatoonSergeant says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PSG initiateMovement)))}; \\
 & \quad \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL report1))))] :: outs)) \iff \\
 & \quad \quad \quad \text{authenticationTest inputOK} \\
 & \quad \quad \quad [\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))}; \\
 & \quad \quad \quad \text{Name PlatoonLeader says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL tentativePlan)))};
 \end{aligned}$$

```

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

[PlatoonLeader_WARNO_exec_report1_justified_thm]

```

⊤ ∀ NS Out M Oi Os.
  TR ( $M, O_i, O_s$ )
  (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))  $\iff$ 
authenticationTest inputOK
  [Name PlatoonLeader says prop (SOME (SLc (PL recon)));
   Name PlatoonLeader says
   prop (SOME (SLc (PL tentativePlan)));
   Name PlatoonSergeant says
   prop (SOME (SLc (PSG initiateMovement)));
   Name PlatoonLeader says
   prop (SOME (SLc (PL report1)))]  $\wedge$ 
CFGInterpret ( $M, O_i, O_s$ )
  (CFG inputOK secContext secContextNull
    ([Name PlatoonLeader says
     prop (SOME (SLc (PL recon)));
     Name PlatoonLeader says
     prop (SOME (SLc (PL tentativePlan)));
     Name PlatoonSergeant says
     prop (SOME (SLc (PSG initiateMovement)));
     Name PlatoonLeader says
     prop (SOME (SLc (PL report1))))])::ins) WARNO outs)  $\wedge$ 
( $M, O_i, O_s$ ) satList
  [prop (SOME (SLc (PL recon)));
   prop (SOME (SLc (PL tentativePlan)));
   prop (SOME (SLc (PSG initiateMovement)));
   prop (SOME (SLc (PL report1)))]

```

[PlatoonLeader_WARNO_exec_report1_lemma]

```

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

```

[PlatoonSergeant_trap_plCommand_justified_lemma]

```

 $\vdash \forall NS\ Out\ M\ Oi\ Os.$ 
  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) \wedge
           ( $M, Oi, Os$ ) sat prop NONE

```

[PlatoonSergeant_trap_plCommand_justified_thm]

```

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

```

[PlatoonSergeant_trap_plCommand_lemma]

```

 $\vdash \forall M \ Oi \ Os.$ 
    CFGInterpret (M, Oi, Os)
    (CFG inputOK secContext secContextNull
     ([Name PlatoonSergeant says
      prop (SOME (SLc (PL plCommand)))::ins) s outs)  $\Rightarrow$ 
     (M, Oi, Os) sat prop NONE
  
```

18 PlanPBType Theory

Built: 10 June 2018

Parent Theories: indexedLists, patternMatches

18.1 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
  
```

18.2 Theorems

[plCommand_distinct_clauses]

```

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

```

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

```

[psgCommand_distinct_clauses]

```

└ initiateMovement ≠ psgIncomplete ∧
initiateMovement ≠ invalidPsgCommand ∧
psgIncomplete ≠ invalidPsgCommand

```

[s1Command_distinct_clauses]

```

└ ∀ a' a. PL a ≠ PSG a'

```

[s1Command_one_one]

```

└ ( ∀ a a'. (PL a = PL a') ⇔ (a = a')) ∧
  ∀ a a'. (PSG a = PSG a') ⇔ (a = a')

```

[s1Output_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 ≠ unUnauthorized ∧
Report2 ≠ Complete ∧ Report2 ≠ unAuthenticated ∧
Report2 ≠ unAuthorized ∧ Complete ≠ unAuthenticated ∧
Complete ≠ unAuthorized ∧ unAuthenticated ≠ unAuthorized

```

[s1Role_distinct_clauses]

```

└ PlatoonLeader ≠ PlatoonSergeant

```

[s1State_distinct_clauses]

```

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

```

```

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

```

19 PlanPBDef Theory

Built: 10 June 2018

Parent Theories: PlanPBType, aclfoundation, OMNIType

19.1 Definitions

[PL_notWARNO_Auth_def]

```

 $\vdash \forall cmd.$ 
  PL_notWARNO_Auth cmd =
    if cmd = report1 then prop NONE
    else
      Name PlatoonLeader says prop (SOME (SLc (PL cmd))) impf
      Name PlatoonLeader controls prop (SOME (SLc (PL cmd)))

```

[PL_WARNO_Auth_def]

```

 $\vdash PL\_WARNO\_Auth =$ 
  prop (SOME (SLc (PL recon))) impf
  prop (SOME (SLc (PL tentativePlan))) impf
  prop (SOME (SLc (PSG initiateMovement))) impf
  Name PlatoonLeader controls prop (SOME (SLc (PL report1)))

```

[secContext_def]

```

 $\vdash \forall s\ x.$ 
  secContext s x =
    if s = WARNO then

```

```

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
  [PL_WARNO_Auth;
   Name PlatoonLeader controls
   prop (SOME (SLc (PL recon)));
   Name PlatoonLeader controls
   prop (SOME (SLc (PL tentativePlan)));
   Name PlatoonSergeant controls
   prop (SOME (SLc (PSG initiateMovement)))]
 else [prop NONE]
else if getPlCom  $x$  = invalidPlCommand then [prop NONE]
else [PL_notWARNO_Auth (getPlCom  $x$ )]

[secContextNull_def]

```

$\vdash \forall x. \text{secContextNull } x = [\text{TT}]$

19.2 Theorems

[getInitMove_def]

```

 $\vdash (\text{getInitMove } [] = [\text{NONE}]) \wedge$ 
 $(\forall xs.$ 
  getInitMove
    (Name PlatoonSergeant says
     prop (SOME (SLc (PSG initiateMovement)))):: $xs$ ) =
  [SOME (SLc (PSG initiateMovement))]) \wedge
 $(\forall xs. \text{getInitMove } (\text{TT}::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs. \text{getInitMove } (\text{FF}::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v_2. \text{getInitMove } (\text{prop } v_2::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v_3. \text{getInitMove } (\text{notf } v_3::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v_5 v_4. \text{getInitMove } (v_4 \text{ andf } v_5::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v_7 v_6. \text{getInitMove } (v_6 \text{ orf } v_7::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v_9 v_8. \text{getInitMove } (v_8 \text{ impf } v_9::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v_{11} v_{10}.$ 
  getInitMove ( $v_{10}$  eqf  $v_{11}::xs$ ) = getInitMove  $xs$ )  $\wedge$ 
 $(\forall xs v_{12}. \text{getInitMove } (v_{12} \text{ says TT}::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v_{12}. \text{getInitMove } (v_{12} \text{ says FF}::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v134.$ 
  getInitMove (Name  $v134$  says prop NONE):: $xs$ ) =
  getInitMove  $xs$ )  $\wedge$ 
 $(\forall xs v144.$ 
  getInitMove
    (Name PlatoonLeader says prop (SOME  $v144$ )):: $xs$ ) =
  getInitMove  $xs$ )  $\wedge$ 
 $(\forall xs v146.$ 

```

```

getInitMove
  (Name PlatoonSergeant says prop (SOME (ESCc v146))::xs) =
    getInitMove xs) ∧
(∀ xs v150.
  getInitMove
    (Name PlatoonSergeant says prop (SOME (SLc (PL v150)))::xs) =
      getInitMove xs) ∧
(∀ xs.
  getInitMove
    (Name PlatoonSergeant says
      prop (SOME (SLc (PSG psgIncomplete)))::xs) =
        getInitMove xs) ∧
(∀ xs.
  getInitMove
    (Name PlatoonSergeant says
      prop (SOME (SLc (PSG invalidPsgCommand)))::xs) =
        getInitMove xs) ∧
(∀ xs v68 v136 v135.
  getInitMove (v135 meet v136 says prop v68::xs) =
    getInitMove xs) ∧
(∀ xs v68 v138 v137.
  getInitMove (v137 quoting v138 says prop v68::xs) =
    getInitMove xs) ∧
(∀ xs v69 v12.
  getInitMove (v12 says notf v69::xs) = getInitMove xs) ∧
(∀ xs v71 v70 v12.
  getInitMove (v12 says (v70 andf v71)::xs) =
    getInitMove xs) ∧
(∀ xs v73 v72 v12.
  getInitMove (v12 says (v72 orf v73)::xs) =
    getInitMove xs) ∧
(∀ xs v75 v74 v12.
  getInitMove (v12 says (v74 impf v75)::xs) =
    getInitMove xs) ∧
(∀ xs v77 v76 v12.
  getInitMove (v12 says (v76 eqf v77)::xs) =
    getInitMove xs) ∧
(∀ xs v79 v78 v12.
  getInitMove (v12 says v78 says v79::xs) =
    getInitMove xs) ∧
(∀ xs v81 v80 v12.
  getInitMove (v12 says v80 speaks_for v81::xs) =
    getInitMove xs) ∧
(∀ xs v83 v82 v12.
  getInitMove (v12 says v82 controls v83::xs) =
    getInitMove xs) ∧
(∀ xs v86 v85 v84 v12.

```

```

getInitMove (v12 says reps v84 v85 v86::xs) =
getInitMove xs) ∧
(∀xs v88 v87 v12.
  getInitMove (v12 says v87 domi v88::xs) =
  getInitMove xs) ∧
(∀xs v90 v89 v12.
  getInitMove (v12 says v89 eqi v90::xs) = getInitMove xs) ∧
(∀xs v92 v91 v12.
  getInitMove (v12 says v91 doms v92::xs) =
  getInitMove xs) ∧
(∀xs v94 v93 v12.
  getInitMove (v12 says v93 eqs v94::xs) = getInitMove xs) ∧
(∀xs v96 v95 v12.
  getInitMove (v12 says v95 eqn v96::xs) = getInitMove xs) ∧
(∀xs v98 v97 v12.
  getInitMove (v12 says v97 lte v98::xs) = getInitMove xs) ∧
(∀xs v99 v12 v100.
  getInitMove (v12 says v99 lt v100::xs) = getInitMove xs) ∧
(∀xs v15 v14.
  getInitMove (v14 speaks_for v15::xs) = getInitMove xs) ∧
(∀xs v17 v16.
  getInitMove (v16 controls v17::xs) = getInitMove xs) ∧
(∀xs v20 v19 v18.
  getInitMove (reps v18 v19 v20::xs) = getInitMove xs) ∧
(∀xs v22 v21.
  getInitMove (v21 domi v22::xs) = getInitMove xs) ∧
(∀xs v24 v23.
  getInitMove (v23 eqi v24::xs) = getInitMove xs) ∧
(∀xs v26 v25.
  getInitMove (v25 doms v26::xs) = getInitMove xs) ∧
(∀xs v28 v27.
  getInitMove (v27 eqs v28::xs) = getInitMove xs) ∧
(∀xs v30 v29.
  getInitMove (v29 eqn v30::xs) = getInitMove xs) ∧
(∀xs v32 v31.
  getInitMove (v31 lte v32::xs) = getInitMove xs) ∧
∀xs v33. getInitMove (v33 lt v34::xs) = getInitMove xs

```

[getInitMove_ind]

```

⊢ ∀P.
  P [] ∧
  (∀xs.
    P
    (Name PlatoonSergeant says
      prop (SOME (SLc (PSG initiateMovement)))::xs)) ∧
    (∀xs. P xs ⇒ P (TT::xs)) ∧ (∀xs. P xs ⇒ P (FF::xs)) ∧
    (∀v2 xs. P xs ⇒ P (prop v2::xs)) ∧
    (∀v3 xs. P xs ⇒ P (notf v3::xs)) ∧
    (∀v4 v5 xs. P xs ⇒ P (v4 andf v5::xs)) ∧

```

```

(∀ v6 v7 xs. P xs ⇒ P (v6 orf v7::xs)) ∧
(∀ v8 v9 xs. P xs ⇒ P (v8 impf v9::xs)) ∧
(∀ v10 v11 xs. P xs ⇒ P (v10 eqf v11::xs)) ∧
(∀ v12 xs. P xs ⇒ P (v12 says TT::xs)) ∧
(∀ v12 xs. P xs ⇒ P (v12 says FF::xs)) ∧
(∀ v134 xs. P xs ⇒ P (Name v134 says prop NONE::xs)) ∧
(∀ v144 xs.
  P xs ⇒
  P (Name PlatoonLeader says prop (SOME v144)::xs)) ∧
(∀ v146 xs.
  P xs ⇒
  P
    (Name PlatoonSergeant says prop (SOME (ESCc v146))::xs)) ∧
(∀ v150 xs.
  P xs ⇒
  P
    (Name PlatoonSergeant says
      prop (SOME (SLc (PL v150)))::xs)) ∧
(∀ xs.
  P xs ⇒
  P
    (Name PlatoonSergeant says
      prop (SOME (SLc (PSG psgIncomplete)))::xs)) ∧
(∀ xs.
  P xs ⇒
  P
    (Name PlatoonSergeant says
      prop (SOME (SLc (PSG invalidPsgCommand)))::xs)) ∧
(∀ v135 v136 v68 xs.
  P xs ⇒ P (v135 meet v136 says prop v68::xs)) ∧
(∀ v137 v138 v68 xs.
  P xs ⇒ P (v137 quoting v138 says prop v68::xs)) ∧
(∀ v12 v69 xs. P xs ⇒ P (v12 says notf v69::xs)) ∧
(∀ v12 v70 v71 xs. P xs ⇒ P (v12 says (v70 andf v71)::xs)) ∧
(∀ v12 v72 v73 xs. P xs ⇒ P (v12 says (v72 orf v73)::xs)) ∧
(∀ v12 v74 v75 xs. P xs ⇒ P (v12 says (v74 impf v75)::xs)) ∧
(∀ v12 v76 v77 xs. P xs ⇒ P (v12 says (v76 eqf v77)::xs)) ∧
(∀ v12 v78 v79 xs. P xs ⇒ P (v12 says v78 says v79::xs)) ∧
(∀ v12 v80 v81 xs.
  P xs ⇒ P (v12 says v80 speaks_for v81::xs)) ∧
(∀ v12 v82 v83 xs.
  P xs ⇒ P (v12 says v82 controls v83::xs)) ∧
(∀ v12 v84 v85 v86 xs.
  P xs ⇒ P (v12 says reps v84 v85 v86::xs)) ∧
(∀ v12 v87 v88 xs. P xs ⇒ P (v12 says v87 domi v88::xs)) ∧
(∀ v12 v89 v90 xs. P xs ⇒ P (v12 says v89 equi v90::xs)) ∧
(∀ v12 v91 v92 xs. P xs ⇒ P (v12 says v91 doms v92::xs)) ∧
(∀ v12 v93 v94 xs. P xs ⇒ P (v12 says v93 eqs v94::xs)) ∧

```

$$\begin{aligned}
& (\forall v_{12} v_{95} v_{96} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{95} \text{ eqn } v_{96}::xs)) \wedge \\
& (\forall v_{12} v_{97} v_{98} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{97} \text{ lte } v_{98}::xs)) \wedge \\
& (\forall v_{12} v_{99} v_{100} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{99} \text{ lt } v_{100}::xs)) \wedge \\
& (\forall v_{14} v_{15} xs. P xs \Rightarrow P (v_{14} \text{ speaks_for } v_{15}::xs)) \wedge \\
& (\forall v_{16} v_{17} xs. P xs \Rightarrow P (v_{16} \text{ controls } v_{17}::xs)) \wedge \\
& (\forall v_{18} v_{19} v_{20} xs. P xs \Rightarrow P (\text{reps } v_{18} v_{19} v_{20}::xs)) \wedge \\
& (\forall v_{21} v_{22} xs. P xs \Rightarrow P (v_{21} \text{ domi } v_{22}::xs)) \wedge \\
& (\forall v_{23} v_{24} xs. P xs \Rightarrow P (v_{23} \text{ eqi } v_{24}::xs)) \wedge \\
& (\forall v_{25} v_{26} xs. P xs \Rightarrow P (v_{25} \text{ doms } v_{26}::xs)) \wedge \\
& (\forall v_{27} v_{28} xs. P xs \Rightarrow P (v_{27} \text{ eqs } v_{28}::xs)) \wedge \\
& (\forall v_{29} v_{30} xs. P xs \Rightarrow P (v_{29} \text{ eqn } v_{30}::xs)) \wedge \\
& (\forall v_{31} v_{32} xs. P xs \Rightarrow P (v_{31} \text{ lte } v_{32}::xs)) \wedge \\
& (\forall v_{33} v_{34} xs. P xs \Rightarrow P (v_{33} \text{ lt } v_{34}::xs)) \Rightarrow \\
& \forall v. P v
\end{aligned}$$

[getPlCom_def]

$$\begin{aligned}
& \vdash (\text{getPlCom } [] = \text{invalidPlCommand}) \wedge \\
& (\forall xs cmd. \\
& \quad \text{getPlCom} \\
& \quad (\text{Name PlatoonLeader says prop (SOME (SLc (PL cmd)))} :: \\
& \quad \quad xs) = \\
& \quad cmd) \wedge (\forall xs. \text{getPlCom (TT} :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs. \text{getPlCom (FF} :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_2. \text{getPlCom (prop } v_2 :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_3. \text{getPlCom (notf } v_3 :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_5 v_4. \text{getPlCom (v}_4 \text{ andf } v_5 :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_7 v_6. \text{getPlCom (v}_6 \text{ orf } v_7 :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_9 v_8. \text{getPlCom (v}_8 \text{ impf } v_9 :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_{11} v_{10}. \text{getPlCom (v}_{10} \text{ eqf } v_{11} :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_{12}. \text{getPlCom (v}_{12} \text{ says TT} :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_{12}. \text{getPlCom (v}_{12} \text{ says FF} :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_{134}. \\
& \quad \quad \text{getPlCom (Name } v_{134} \text{ says prop NONE} :: xs) = \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_{146}. \\
& \quad \quad \text{getPlCom} \\
& \quad \quad (\text{Name PlatoonLeader says prop (SOME (ESCc } v_{146})) :: xs) = \\
& \quad \quad \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_{151}. \\
& \quad \quad \text{getPlCom} \\
& \quad \quad (\text{Name PlatoonLeader says prop (SOME (SLc (PSG } v_{151}))} :: \\
& \quad \quad \quad xs) = \\
& \quad \quad \quad \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_{144}. \\
& \quad \quad \text{getPlCom} \\
& \quad \quad (\text{Name PlatoonSergeant says prop (SOME } v_{144}) :: xs) = \\
& \quad \quad \text{getPlCom } xs) \wedge \\
& \quad (\forall xs v_{68} v_{136} v_{135}. \\
& \quad \quad \text{getPlCom (v}_{135} \text{ meet } v_{136} \text{ says prop } v_{68} :: xs) = \\
& \quad \quad \text{getPlCom } xs) \wedge
\end{aligned}$$

```

(∀ xs v68 v138 v137.
  getPlCom (v137 quoting v138 says prop v68::xs) =
  getPlCom xs) ∧
(∀ xs v69 v12.
  getPlCom (v12 says notf v69::xs) = getPlCom xs) ∧
(∀ xs v71 v70 v12.
  getPlCom (v12 says (v70 andf v71)::xs) = getPlCom xs) ∧
(∀ xs v73 v72 v12.
  getPlCom (v12 says (v72 orf v73)::xs) = getPlCom xs) ∧
(∀ xs v75 v74 v12.
  getPlCom (v12 says (v74 impf v75)::xs) = getPlCom xs) ∧
(∀ xs v77 v76 v12.
  getPlCom (v12 says (v76 eqf v77)::xs) = getPlCom xs) ∧
(∀ xs v79 v78 v12.
  getPlCom (v12 says v78 says v79::xs) = getPlCom xs) ∧
(∀ xs v81 v80 v12.
  getPlCom (v12 says v80 speaks_for v81::xs) =
  getPlCom xs) ∧
(∀ xs v83 v82 v12.
  getPlCom (v12 says v82 controls v83::xs) = getPlCom xs) ∧
(∀ xs v86 v85 v84 v12.
  getPlCom (v12 says reps v84 v85 v86::xs) = getPlCom xs) ∧
(∀ xs v88 v87 v12.
  getPlCom (v12 says v87 domi v88::xs) = getPlCom xs) ∧
(∀ xs v90 v89 v12.
  getPlCom (v12 says v89 eqi v90::xs) = getPlCom xs) ∧
(∀ xs v92 v91 v12.
  getPlCom (v12 says v91 doms v92::xs) = getPlCom xs) ∧
(∀ xs v94 v93 v12.
  getPlCom (v12 says v93 eqs v94::xs) = getPlCom xs) ∧
(∀ xs v96 v95 v12.
  getPlCom (v12 says v95 eqn v96::xs) = getPlCom xs) ∧
(∀ xs v98 v97 v12.
  getPlCom (v12 says v97 lte v98::xs) = getPlCom xs) ∧
(∀ xs v99 v12 v100.
  getPlCom (v12 says v99 lt v100::xs) = getPlCom xs) ∧
(∀ xs v15 v14.
  getPlCom (v14 speaks_for v15::xs) = getPlCom xs) ∧
(∀ xs v17 v16.
  getPlCom (v16 controls v17::xs) = getPlCom xs) ∧
(∀ xs v20 v19 v18.
  getPlCom (reps v18 v19 v20::xs) = getPlCom xs) ∧
(∀ xs v22 v21. getPlCom (v21 domi v22::xs) = getPlCom xs) ∧
(∀ xs v24 v23. getPlCom (v23 eqi v24::xs) = getPlCom xs) ∧
(∀ xs v26 v25. getPlCom (v25 doms v26::xs) = getPlCom xs) ∧
(∀ xs v28 v27. getPlCom (v27 eqs v28::xs) = getPlCom xs) ∧
(∀ xs v30 v29. getPlCom (v29 eqn v30::xs) = getPlCom xs) ∧
(∀ xs v32 v31. getPlCom (v31 lte v32::xs) = getPlCom xs) ∧
  ∀ xs v34 v33. getPlCom (v33 lt v34::xs) = getPlCom xs

```

[getPlCom_ind]

```

 $\vdash \forall P.$ 
 $P [] \wedge$ 
 $(\forall cmd\ xs.$ 
 $P$ 
 $(\text{Name PlatoonLeader says prop (SOME (SLc (PL cmd)))::}$ 
 $xs)) \wedge (\forall xs.\ P\ xs \Rightarrow P\ (\text{TT}::xs)) \wedge$ 
 $(\forall xs.\ P\ xs \Rightarrow P\ (\text{FF}::xs)) \wedge$ 
 $(\forall v_2\ xs.\ P\ xs \Rightarrow P\ (\text{prop } v_2::xs)) \wedge$ 
 $(\forall v_3\ xs.\ P\ xs \Rightarrow P\ (\text{notf } v_3::xs)) \wedge$ 
 $(\forall v_4\ v_5\ xs.\ P\ xs \Rightarrow P\ (v_4 \text{ andf } v_5::xs)) \wedge$ 
 $(\forall v_6\ v_7\ xs.\ P\ xs \Rightarrow P\ (v_6 \text{ orf } v_7::xs)) \wedge$ 
 $(\forall v_8\ v_9\ xs.\ P\ xs \Rightarrow P\ (v_8 \text{ impf } v_9::xs)) \wedge$ 
 $(\forall v_{10}\ v_{11}\ xs.\ P\ xs \Rightarrow P\ (v_{10} \text{ eqf } v_{11}::xs)) \wedge$ 
 $(\forall v_{12}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says TT}::xs)) \wedge$ 
 $(\forall v_{12}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says FF}::xs)) \wedge$ 
 $(\forall v_{134}\ xs.\ P\ xs \Rightarrow P\ (\text{Name } v_{134} \text{ says prop NONE}::xs)) \wedge$ 
 $(\forall v_{146}\ xs.$ 
 $P\ xs \Rightarrow$ 
 $P$ 
 $(\text{Name PlatoonLeader says prop (SOME (ESCc } v_{146}))::$ 
 $xs)) \wedge$ 
 $(\forall v_{151}\ xs.$ 
 $P\ xs \Rightarrow$ 
 $P$ 
 $(\text{Name PlatoonLeader says }$ 
 $\text{prop (SOME (SLc (PSG } v_{151}))::xs)) \wedge$ 
 $(\forall v_{144}\ xs.$ 
 $P\ xs \Rightarrow$ 
 $P\ (\text{Name PlatoonSergeant says prop (SOME } v_{144})::xs)) \wedge$ 
 $(\forall v_{135}\ v_{136}\ v_{68}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{135} \text{ meet } v_{136} \text{ says prop } v_{68}::xs)) \wedge$ 
 $(\forall v_{137}\ v_{138}\ v_{68}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{137} \text{ quoting } v_{138} \text{ says prop } v_{68}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{69}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says notf } v_{69}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{70}\ v_{71}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says (v}_{70} \text{ andf } v_{71})::xs)) \wedge$ 
 $(\forall v_{12}\ v_{72}\ v_{73}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says (v}_{72} \text{ orf } v_{73})::xs)) \wedge$ 
 $(\forall v_{12}\ v_{74}\ v_{75}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says (v}_{74} \text{ impf } v_{75})::xs)) \wedge$ 
 $(\forall v_{12}\ v_{76}\ v_{77}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says (v}_{76} \text{ eqf } v_{77})::xs)) \wedge$ 
 $(\forall v_{12}\ v_{78}\ v_{79}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says v}_{78} \text{ says v}_{79}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{80}\ v_{81}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{12} \text{ says v}_{80} \text{ speaks_for v}_{81}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{82}\ v_{83}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{12} \text{ says v}_{82} \text{ controls v}_{83}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{84}\ v_{85}\ v_{86}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{12} \text{ says reps v}_{84}\ v_{85}\ v_{86}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{87}\ v_{88}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says v}_{87} \text{ domi v}_{88}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{89}\ v_{90}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says v}_{89} \text{ equi v}_{90}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{91}\ v_{92}\ xs.\ P\ xs \Rightarrow P\ (v_{12} \text{ says v}_{91} \text{ doms v}_{92}::xs)) \wedge$ 

```

$$\begin{aligned}
& (\forall v_{12} v_{93} v_{94} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{93} \text{ eqs } v_{94}::xs)) \wedge \\
& (\forall v_{12} v_{95} v_{96} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{95} \text{ eqn } v_{96}::xs)) \wedge \\
& (\forall v_{12} v_{97} v_{98} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{97} \text{ lte } v_{98}::xs)) \wedge \\
& (\forall v_{12} v_{99} v_{100} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{99} \text{ lt } v_{100}::xs)) \wedge \\
& (\forall v_{14} v_{15} xs. P xs \Rightarrow P (v_{14} \text{ speaks_for } v_{15}::xs)) \wedge \\
& (\forall v_{16} v_{17} xs. P xs \Rightarrow P (v_{16} \text{ controls } v_{17}::xs)) \wedge \\
& (\forall v_{18} v_{19} v_{20} xs. P xs \Rightarrow P (\text{reps } v_{18} v_{19} v_{20}::xs)) \wedge \\
& (\forall v_{21} v_{22} xs. P xs \Rightarrow P (v_{21} \text{ domi } v_{22}::xs)) \wedge \\
& (\forall v_{23} v_{24} xs. P xs \Rightarrow P (v_{23} \text{ eqi } v_{24}::xs)) \wedge \\
& (\forall v_{25} v_{26} xs. P xs \Rightarrow P (v_{25} \text{ doms } v_{26}::xs)) \wedge \\
& (\forall v_{27} v_{28} xs. P xs \Rightarrow P (v_{27} \text{ eqs } v_{28}::xs)) \wedge \\
& (\forall v_{29} v_{30} xs. P xs \Rightarrow P (v_{29} \text{ eqn } v_{30}::xs)) \wedge \\
& (\forall v_{31} v_{32} xs. P xs \Rightarrow P (v_{31} \text{ lte } v_{32}::xs)) \wedge \\
& (\forall v_{33} v_{34} xs. P xs \Rightarrow P (v_{33} \text{ lt } v_{34}::xs)) \Rightarrow \\
& \forall v. P v
\end{aligned}$$

[getPsgCom_def]

```


$$\vdash (\text{getPsgCom } [] = \text{invalidPsgCommand}) \wedge
(\forall xs cmd.
\text{getPsgCom}
(\text{Name PlatoonSergeant says prop (SOME (SLc (PSG cmd)))} :: xs) =
cmd) \wedge (\forall xs. \text{getPsgCom (TT::xs)} = \text{getPsgCom xs}) \wedge
(\forall xs. \text{getPsgCom (FF::xs)} = \text{getPsgCom xs}) \wedge
(\forall xs v_2. \text{getPsgCom (prop } v_2::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_3. \text{getPsgCom (notf } v_3::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_5 v_4. \text{getPsgCom (v}_4 \text{ andf } v_5::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_7 v_6. \text{getPsgCom (v}_6 \text{ orf } v_7::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_9 v_8. \text{getPsgCom (v}_8 \text{ impf } v_9::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_{11} v_{10}. \text{getPsgCom (v}_{10} \text{ eqf } v_{11}::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_{12}. \text{getPsgCom (v}_{12} \text{ says TT}::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_{12}. \text{getPsgCom (v}_{12} \text{ says FF}::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_{134}.
\text{getPsgCom (Name } v_{134} \text{ says prop NONE}::xs) = \text{getPsgCom xs}) \wedge
(\forall xs v_{144}.
\text{getPsgCom (Name PlatoonLeader says prop (SOME } v_{144})::xs) =
\text{getPsgCom xs}) \wedge
(\forall xs v_{146}.
\text{getPsgCom}
(\text{Name PlatoonSergeant says prop (SOME (ESCc } v_{146})):: xs) =
\text{getPsgCom xs}) \wedge
(\forall xs v_{150}.
\text{getPsgCom}
(\text{Name PlatoonSergeant says prop (SOME (SLc (PL } v_{150}))} :: xs) =
\text{getPsgCom xs}) \wedge
(\forall xs v_{68} v_{136} v_{135}.
\text{getPsgCom (v}_{135} \text{ meet } v_{136} \text{ says prop } v_{68}::xs) =$$


```

```

getPsgCom xs) ∧
(∀ xs v68 v138 v137.
  getPsgCom (v137 quoting v138 says prop v68::xs) =
  getPsgCom xs) ∧
(∀ xs v69 v12.
  getPsgCom (v12 says notf v69::xs) = getPsgCom xs) ∧
(∀ xs v71 v70 v12.
  getPsgCom (v12 says (v70 andf v71)::xs) = getPsgCom xs) ∧
(∀ xs v73 v72 v12.
  getPsgCom (v12 says (v72 orf v73)::xs) = getPsgCom xs) ∧
(∀ xs v75 v74 v12.
  getPsgCom (v12 says (v74 impf v75)::xs) = getPsgCom xs) ∧
(∀ xs v77 v76 v12.
  getPsgCom (v12 says (v76 eqf v77)::xs) = getPsgCom xs) ∧
(∀ xs v79 v78 v12.
  getPsgCom (v12 says v78 says v79::xs) = getPsgCom xs) ∧
(∀ xs v81 v80 v12.
  getPsgCom (v12 says v80 speaks_for v81::xs) =
  getPsgCom xs) ∧
(∀ xs v83 v82 v12.
  getPsgCom (v12 says v82 controls v83::xs) =
  getPsgCom xs) ∧
(∀ xs v86 v85 v84 v12.
  getPsgCom (v12 says reps v84 v85 v86::xs) =
  getPsgCom xs) ∧
(∀ xs v88 v87 v12.
  getPsgCom (v12 says v87 domi v88::xs) = getPsgCom xs) ∧
(∀ xs v90 v89 v12.
  getPsgCom (v12 says v89 eqi v90::xs) = getPsgCom xs) ∧
(∀ xs v92 v91 v12.
  getPsgCom (v12 says v91 doms v92::xs) = getPsgCom xs) ∧
(∀ xs v94 v93 v12.
  getPsgCom (v12 says v93 eqs v94::xs) = getPsgCom xs) ∧
(∀ xs v96 v95 v12.
  getPsgCom (v12 says v95 eqn v96::xs) = getPsgCom xs) ∧
(∀ xs v98 v97 v12.
  getPsgCom (v12 says v97 lte v98::xs) = getPsgCom xs) ∧
(∀ xs v99 v12 v100.
  getPsgCom (v12 says v99 lt v100::xs) = getPsgCom xs) ∧
(∀ xs v15 v14.
  getPsgCom (v14 speaks_for v15::xs) = getPsgCom xs) ∧
(∀ xs v17 v16.
  getPsgCom (v16 controls v17::xs) = getPsgCom xs) ∧
(∀ xs v20 v19 v18.
  getPsgCom (reps v18 v19 v20::xs) = getPsgCom xs) ∧
(∀ xs v22 v21. getPsgCom (v21 domi v22::xs) = getPsgCom xs) ∧
(∀ xs v24 v23. getPsgCom (v23 eqi v24::xs) = getPsgCom xs) ∧
(∀ xs v26 v25. getPsgCom (v25 doms v26::xs) = getPsgCom xs) ∧
(∀ xs v28 v27. getPsgCom (v27 eqs v28::xs) = getPsgCom xs) ∧

```

$$\begin{aligned}
 & (\forall xs \ v_{30} \ v_{29}. \text{getPsgCom } (v_{29} \text{ eqn } v_{30}::xs) = \text{getPsgCom } xs) \wedge \\
 & (\forall xs \ v_{32} \ v_{31}. \text{getPsgCom } (v_{31} \text{ lte } v_{32}::xs) = \text{getPsgCom } xs) \wedge \\
 & \forall xs \ v_{34} \ v_{33}. \text{getPsgCom } (v_{33} \text{ lt } v_{34}::xs) = \text{getPsgCom } xs
 \end{aligned}$$

[getPsgCom_ind]

$$\begin{aligned}
 & \vdash \forall P. \\
 & \quad P [] \wedge \\
 & \quad (\forall cmd \ xs. \\
 & \quad \quad P \\
 & \quad \quad (\text{Name PlatoonSergeant says} \\
 & \quad \quad \text{prop (SOME (SLc (PSG cmd)))::xs}) \wedge \\
 & \quad \quad (\forall xs. P \ xs \Rightarrow P \ (\text{TT}::xs)) \wedge (\forall xs. P \ xs \Rightarrow P \ (\text{FF}::xs)) \wedge \\
 & \quad \quad (\forall v_2 \ xs. P \ xs \Rightarrow P \ (\text{prop } v_2::xs)) \wedge \\
 & \quad \quad (\forall v_3 \ xs. P \ xs \Rightarrow P \ (\text{notf } v_3::xs)) \wedge \\
 & \quad \quad (\forall v_4 \ v_5 \ xs. P \ xs \Rightarrow P \ (v_4 \ \text{andf } v_5::xs)) \wedge \\
 & \quad \quad (\forall v_6 \ v_7 \ xs. P \ xs \Rightarrow P \ (v_6 \ \text{orf } v_7::xs)) \wedge \\
 & \quad \quad (\forall v_8 \ v_9 \ xs. P \ xs \Rightarrow P \ (v_8 \ \text{impf } v_9::xs)) \wedge \\
 & \quad \quad (\forall v_{10} \ v_{11} \ xs. P \ xs \Rightarrow P \ (v_{10} \ \text{eqf } v_{11}::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says TT}::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says FF}::xs)) \wedge \\
 & \quad \quad (\forall v_{134} \ xs. P \ xs \Rightarrow P \ (\text{Name } v_{134} \ \text{says prop NONE}::xs)) \wedge \\
 & \quad \quad (\forall v_{144} \ xs. \\
 & \quad \quad \quad P \ xs \Rightarrow \\
 & \quad \quad \quad P \ (\text{Name PlatoonLeader says prop (SOME } v_{144})::xs)) \wedge \\
 & \quad \quad (\forall v_{146} \ xs. \\
 & \quad \quad \quad P \ xs \Rightarrow \\
 & \quad \quad \quad P \\
 & \quad \quad \quad (\text{Name PlatoonSergeant says prop (SOME (ESCc } v_{146})::xs}) \wedge \\
 & \quad \quad (\forall v_{150} \ xs. \\
 & \quad \quad \quad P \ xs \Rightarrow \\
 & \quad \quad \quad P \\
 & \quad \quad \quad (\text{Name PlatoonSergeant says} \\
 & \quad \quad \quad \text{prop (SOME (SLc (PL } v_{150}))::xs}) \wedge \\
 & \quad \quad (\forall v_{135} \ v_{136} \ v_{68} \ xs. \\
 & \quad \quad \quad P \ xs \Rightarrow P \ (v_{135} \ \text{meet } v_{136} \ \text{says prop } v_{68}::xs)) \wedge \\
 & \quad \quad (\forall v_{137} \ v_{138} \ v_{68} \ xs. \\
 & \quad \quad \quad P \ xs \Rightarrow P \ (v_{137} \ \text{quoting } v_{138} \ \text{says prop } v_{68}::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{69} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says notf } v_{69}::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{70} \ v_{71} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says } (v_{70} \ \text{andf } v_{71})::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{72} \ v_{73} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says } (v_{72} \ \text{orf } v_{73})::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{74} \ v_{75} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says } (v_{74} \ \text{impf } v_{75})::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{76} \ v_{77} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says } (v_{76} \ \text{eqf } v_{77})::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{78} \ v_{79} \ xs. P \ xs \Rightarrow P \ (v_{12} \ \text{says } v_{78} \ \text{says } v_{79}::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{80} \ v_{81} \ xs. \\
 & \quad \quad \quad P \ xs \Rightarrow P \ (v_{12} \ \text{says } v_{80} \ \text{speaks_for } v_{81}::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{82} \ v_{83} \ xs. \\
 & \quad \quad \quad P \ xs \Rightarrow P \ (v_{12} \ \text{says } v_{82} \ \text{controls } v_{83}::xs)) \wedge \\
 & \quad \quad (\forall v_{12} \ v_{84} \ v_{85} \ v_{86} \ xs.
 \end{aligned}$$

```

 $P \ xs \Rightarrow P \ (\text{v}_{12} \ \text{says} \ \text{reps} \ \text{v}_{84} \ \text{v}_{85} \ \text{v}_{86} :: \ xs)) \wedge$ 
 $(\forall v_{12} \ v_{87} \ v_{88} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{12} \ \text{says} \ \text{v}_{87} \ \text{domi} \ \text{v}_{88} :: \ xs)) \wedge$ 
 $(\forall v_{12} \ v_{89} \ v_{90} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{12} \ \text{says} \ \text{v}_{89} \ \text{eqi} \ \text{v}_{90} :: \ xs)) \wedge$ 
 $(\forall v_{12} \ v_{91} \ v_{92} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{12} \ \text{says} \ \text{v}_{91} \ \text{doms} \ \text{v}_{92} :: \ xs)) \wedge$ 
 $(\forall v_{12} \ v_{93} \ v_{94} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{12} \ \text{says} \ \text{v}_{93} \ \text{eqs} \ \text{v}_{94} :: \ xs)) \wedge$ 
 $(\forall v_{12} \ v_{95} \ v_{96} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{12} \ \text{says} \ \text{v}_{95} \ \text{eqn} \ \text{v}_{96} :: \ xs)) \wedge$ 
 $(\forall v_{12} \ v_{97} \ v_{98} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{12} \ \text{says} \ \text{v}_{97} \ \text{lte} \ \text{v}_{98} :: \ xs)) \wedge$ 
 $(\forall v_{12} \ v_{99} \ v_{100} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{12} \ \text{says} \ \text{v}_{99} \ \text{lt} \ \text{v}_{100} :: \ xs)) \wedge$ 
 $(\forall v_{14} \ v_{15} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{14} \ \text{speaks\_for} \ \text{v}_{15} :: \ xs)) \wedge$ 
 $(\forall v_{16} \ v_{17} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{16} \ \text{controls} \ \text{v}_{17} :: \ xs)) \wedge$ 
 $(\forall v_{18} \ v_{19} \ v_{20} \ xs. \ P \ xs \Rightarrow P \ (\text{reps} \ \text{v}_{18} \ \text{v}_{19} \ \text{v}_{20} :: \ xs)) \wedge$ 
 $(\forall v_{21} \ v_{22} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{21} \ \text{domi} \ \text{v}_{22} :: \ xs)) \wedge$ 
 $(\forall v_{23} \ v_{24} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{23} \ \text{eqi} \ \text{v}_{24} :: \ xs)) \wedge$ 
 $(\forall v_{25} \ v_{26} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{25} \ \text{doms} \ \text{v}_{26} :: \ xs)) \wedge$ 
 $(\forall v_{27} \ v_{28} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{27} \ \text{eqs} \ \text{v}_{28} :: \ xs)) \wedge$ 
 $(\forall v_{29} \ v_{30} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{29} \ \text{eqn} \ \text{v}_{30} :: \ xs)) \wedge$ 
 $(\forall v_{31} \ v_{32} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{31} \ \text{lte} \ \text{v}_{32} :: \ xs)) \wedge$ 
 $(\forall v_{33} \ v_{34} \ xs. \ P \ xs \Rightarrow P \ (\text{v}_{33} \ \text{lt} \ \text{v}_{34} :: \ xs)) \Rightarrow$ 
 $\forall v. \ P \ v$ 

```

[getRecon_def]

```

 $\vdash (\text{getRecon} \ [] = [\text{NONE}]) \wedge$ 
 $(\forall xs.$ 
 $\quad \text{getRecon}$ 
 $\quad (\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))} ::$ 
 $\quad \quad xs) =$ 
 $\quad [\text{SOME (SLc (PL recon))}] \wedge$ 
 $\quad (\forall xs. \ \text{getRecon (TT::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs. \ \text{getRecon (FF::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_2. \ \text{getRecon (prop } v_2 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_3. \ \text{getRecon (notf } v_3 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_5 \ v_4. \ \text{getRecon (v}_4 \ \text{andf } v_5 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_7 \ v_6. \ \text{getRecon (v}_6 \ \text{orf } v_7 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_9 \ v_8. \ \text{getRecon (v}_8 \ \text{impf } v_9 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_{11} \ v_{10}. \ \text{getRecon (v}_{10} \ \text{eqf } v_{11} :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_{12}. \ \text{getRecon (v}_{12} \ \text{says TT::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_{12}. \ \text{getRecon (v}_{12} \ \text{says FF::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_{134}.$ 
 $\quad \quad \text{getRecon (Name } v_{134} \ \text{says prop NONE::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_{146}.$ 
 $\quad \quad \text{getRecon}$ 
 $\quad \quad (\text{Name PlatoonLeader says prop (SOME (ESCc } v_{146})) :: \ xs) =$ 
 $\quad \quad \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs.$ 
 $\quad \quad \text{getRecon}$ 
 $\quad \quad (\text{Name PlatoonLeader says}$ 
 $\quad \quad \quad \text{prop (SOME (SLc (PL receiveMission)))} :: \ xs) =$ 
 $\quad \quad \quad \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs.$ 

```

```

getRecon
  (Name PlatoonLeader says prop (SOME (SLc (PL warno))))::xs) =
    getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says
    prop (SOME (SLc (PL tentativePlan))))::xs) =
  getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says
    prop (SOME (SLc (PL report1))))::xs) =
  getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says
    prop (SOME (SLc (PL completePlan))))::xs) =
  getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says prop (SOME (SLc (PL opoid))))::xs) =
  getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says
    prop (SOME (SLc (PL supervise))))::xs) =
  getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says
    prop (SOME (SLc (PL report2))))::xs) =
  getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says
    prop (SOME (SLc (PL complete))))::xs) =
  getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says
    prop (SOME (SLc (PL plIncomplete))))::xs) =
  getRecon xs) ∧
(∀xs.

getRecon
  (Name PlatoonLeader says
    prop (SOME (SLc (PL invalidPlCommand))))::xs) =
  getRecon xs) ∧

```

```

(∀ xs v151 .
  getRecon
    (Name PlatoonLeader says prop (SOME (SLc (PSG v151)))::
     xs) =
  getRecon xs) ∧
(∀ xs v144 .
  getRecon
    (Name PlatoonSergeant says prop (SOME v144)::xs) =
  getRecon xs) ∧
(∀ xs v68 v136 v135 .
  getRecon (v135 meet v136 says prop v68::xs) =
  getRecon xs) ∧
(∀ xs v68 v138 v137 .
  getRecon (v137 quoting v138 says prop v68::xs) =
  getRecon xs) ∧
(∀ xs v69 v12 .
  getRecon (v12 says notf v69::xs) = getRecon xs) ∧
(∀ xs v71 v70 v12 .
  getRecon (v12 says (v70 andf v71)::xs) = getRecon xs) ∧
(∀ xs v73 v72 v12 .
  getRecon (v12 says (v72 orf v73)::xs) = getRecon xs) ∧
(∀ xs v75 v74 v12 .
  getRecon (v12 says (v74 impf v75)::xs) = getRecon xs) ∧
(∀ xs v77 v76 v12 .
  getRecon (v12 says (v76 eqf v77)::xs) = getRecon xs) ∧
(∀ xs v79 v78 v12 .
  getRecon (v12 says v78 says v79::xs) = getRecon xs) ∧
(∀ xs v81 v80 v12 .
  getRecon (v12 says v80 speaks_for v81::xs) =
  getRecon xs) ∧
(∀ xs v83 v82 v12 .
  getRecon (v12 says v82 controls v83::xs) = getRecon xs) ∧
(∀ xs v86 v85 v84 v12 .
  getRecon (v12 says reps v84 v85 v86::xs) = getRecon xs) ∧
(∀ xs v88 v87 v12 .
  getRecon (v12 says v87 domi v88::xs) = getRecon xs) ∧
(∀ xs v90 v89 v12 .
  getRecon (v12 says v89 eqi v90::xs) = getRecon xs) ∧
(∀ xs v92 v91 v12 .
  getRecon (v12 says v91 doms v92::xs) = getRecon xs) ∧
(∀ xs v94 v93 v12 .
  getRecon (v12 says v93 eqs v94::xs) = getRecon xs) ∧
(∀ xs v96 v95 v12 .
  getRecon (v12 says v95 eqn v96::xs) = getRecon xs) ∧
(∀ xs v98 v97 v12 .
  getRecon (v12 says v97 lte v98::xs) = getRecon xs) ∧
(∀ xs v99 v12 v100 .
  getRecon (v12 says v99 lt v100::xs) = getRecon xs) ∧
(∀ xs v15 v14 .

```

```

    getRecon (v14 speaks_for v15::xs) = getRecon xs) ∧
(∀ xs v17 v16.
    getRecon (v16 controls v17::xs) = getRecon xs) ∧
(∀ xs v20 v19 v18.
    getRecon (reps v18 v19 v20::xs) = getRecon xs) ∧
(∀ xs v22 v21. getRecon (v21 domi v22::xs) = getRecon xs) ∧
(∀ xs v24 v23. getRecon (v23 eqi v24::xs) = getRecon xs) ∧
(∀ xs v26 v25. getRecon (v25 doms v26::xs) = getRecon xs) ∧
(∀ xs v28 v27. getRecon (v27 eqs v28::xs) = getRecon xs) ∧
(∀ xs v30 v29. getRecon (v29 eqn v30::xs) = getRecon xs) ∧
(∀ xs v32 v31. getRecon (v31 lte v32::xs) = getRecon xs) ∧
    ∀ xs v34 v33. getRecon (v33 lt v34::xs) = getRecon xs

```

[getRecon_ind]

```

    ⊢ ∀ P.
        P [] ∧
        (∀ xs.
            P
            (Name PlatoonLeader says
                prop (SOME (SLc (PL recon))))::xs)) ∧
        (∀ xs. P xs ⇒ P (TT::xs)) ∧ (∀ xs. P xs ⇒ P (FF::xs)) ∧
        (∀ v2 xs. P xs ⇒ P (prop v2::xs)) ∧
        (∀ v3 xs. P xs ⇒ P (notf v3::xs)) ∧
        (∀ v4 v5 xs. P xs ⇒ P (v4 andf v5::xs)) ∧
        (∀ v6 v7 xs. P xs ⇒ P (v6 orf v7::xs)) ∧
        (∀ v8 v9 xs. P xs ⇒ P (v8 impf v9::xs)) ∧
        (∀ v10 v11 xs. P xs ⇒ P (v10 eqf v11::xs)) ∧
        (∀ v12 xs. P xs ⇒ P (v12 says TT::xs)) ∧
        (∀ v12 xs. P xs ⇒ P (v12 says FF::xs)) ∧
        (∀ v134 xs. P xs ⇒ P (Name v134 says prop NONE::xs)) ∧
        (∀ v146 xs.
            P xs ⇒
            P
            (Name PlatoonLeader says prop (SOME (ESCc v146))::
                xs)) ∧
        (∀ xs.
            P xs ⇒
            P
            (Name PlatoonLeader says
                prop (SOME (SLc (PL receiveMission))))::xs)) ∧
        (∀ xs.
            P xs ⇒
            P
            (Name PlatoonLeader says
                prop (SOME (SLc (PL warno))))::xs)) ∧
        (∀ xs.
            P xs ⇒
            P
            (Name PlatoonLeader says
                prop (SOME (SLc (PL waro))))::xs))

```

```

prop (SOME (SLc (PL tentativePlan))::xs)) ∧
(∀ xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PL report1))::xs)) ∧
(∀ xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PL completePlan))::xs)) ∧
(∀ xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PL opoid))::xs)) ∧
(∀ xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PL supervise))::xs)) ∧
(∀ xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PL report2))::xs)) ∧
(∀ xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PL complete))::xs)) ∧
(∀ xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PL plIncomplete))::xs)) ∧
(∀ xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PL invalidPlCommand))::xs)) ∧
(∀ v151 xs .
P xs ⇒
P
(Name PlatoonLeader says
prop (SOME (SLc (PSG v151))::xs)) ∧
(∀ v144 xs .
P xs ⇒
P (Name PlatoonSergeant says prop (SOME v144)::xs)) ∧

```

$$\begin{aligned}
& (\forall v_{135} v_{136} v_{68} xs. \\
& \quad P xs \Rightarrow P (v_{135} \text{ meet } v_{136} \text{ says prop } v_{68}::xs)) \wedge \\
& (\forall v_{137} v_{138} v_{68} xs. \\
& \quad P xs \Rightarrow P (v_{137} \text{ quoting } v_{138} \text{ says prop } v_{68}::xs)) \wedge \\
& (\forall v_{12} v_{69} xs. P xs \Rightarrow P (v_{12} \text{ says notf } v_{69}::xs)) \wedge \\
& (\forall v_{12} v_{70} v_{71} xs. P xs \Rightarrow P (v_{12} \text{ says } (v_{70} \text{ andf } v_{71})::xs)) \wedge \\
& (\forall v_{12} v_{72} v_{73} xs. P xs \Rightarrow P (v_{12} \text{ says } (v_{72} \text{ orf } v_{73})::xs)) \wedge \\
& (\forall v_{12} v_{74} v_{75} xs. P xs \Rightarrow P (v_{12} \text{ says } (v_{74} \text{ impf } v_{75})::xs)) \wedge \\
& (\forall v_{12} v_{76} v_{77} xs. P xs \Rightarrow P (v_{12} \text{ says } (v_{76} \text{ eqf } v_{77})::xs)) \wedge \\
& (\forall v_{12} v_{78} v_{79} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{78} \text{ says } v_{79}::xs)) \wedge \\
& (\forall v_{12} v_{80} v_{81} xs. \\
& \quad P xs \Rightarrow P (v_{12} \text{ says } v_{80} \text{ speaks_for } v_{81}::xs)) \wedge \\
& (\forall v_{12} v_{82} v_{83} xs. \\
& \quad P xs \Rightarrow P (v_{12} \text{ says } v_{82} \text{ controls } v_{83}::xs)) \wedge \\
& (\forall v_{12} v_{84} v_{85} v_{86} xs. \\
& \quad P xs \Rightarrow P (v_{12} \text{ says } \text{reps } v_{84} v_{85} v_{86}::xs)) \wedge \\
& (\forall v_{12} v_{87} v_{88} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{87} \text{ domi } v_{88}::xs)) \wedge \\
& (\forall v_{12} v_{89} v_{90} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{89} \text{ eqi } v_{90}::xs)) \wedge \\
& (\forall v_{12} v_{91} v_{92} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{91} \text{ doms } v_{92}::xs)) \wedge \\
& (\forall v_{12} v_{93} v_{94} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{93} \text{ eqs } v_{94}::xs)) \wedge \\
& (\forall v_{12} v_{95} v_{96} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{95} \text{ eqn } v_{96}::xs)) \wedge \\
& (\forall v_{12} v_{97} v_{98} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{97} \text{ lte } v_{98}::xs)) \wedge \\
& (\forall v_{12} v_{99} v_{100} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{99} \text{ lt } v_{100}::xs)) \wedge \\
& (\forall v_{14} v_{15} xs. P xs \Rightarrow P (v_{14} \text{ speaks_for } v_{15}::xs)) \wedge \\
& (\forall v_{16} v_{17} xs. P xs \Rightarrow P (v_{16} \text{ controls } v_{17}::xs)) \wedge \\
& (\forall v_{18} v_{19} v_{20} xs. P xs \Rightarrow P (\text{reps } v_{18} v_{19} v_{20}::xs)) \wedge \\
& (\forall v_{21} v_{22} xs. P xs \Rightarrow P (v_{21} \text{ domi } v_{22}::xs)) \wedge \\
& (\forall v_{23} v_{24} xs. P xs \Rightarrow P (v_{23} \text{ eqi } v_{24}::xs)) \wedge \\
& (\forall v_{25} v_{26} xs. P xs \Rightarrow P (v_{25} \text{ doms } v_{26}::xs)) \wedge \\
& (\forall v_{27} v_{28} xs. P xs \Rightarrow P (v_{27} \text{ eqs } v_{28}::xs)) \wedge \\
& (\forall v_{29} v_{30} xs. P xs \Rightarrow P (v_{29} \text{ eqn } v_{30}::xs)) \wedge \\
& (\forall v_{31} v_{32} xs. P xs \Rightarrow P (v_{31} \text{ lte } v_{32}::xs)) \wedge \\
& (\forall v_{33} v_{34} xs. P xs \Rightarrow P (v_{33} \text{ lt } v_{34}::xs)) \Rightarrow \\
& \forall v. P v
\end{aligned}$$

[getReport_def]

$$\begin{aligned}
& \vdash (\text{getReport } [] = [\text{NONE}]) \wedge \\
& (\forall xs. \\
& \quad \text{getReport} \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL report1})))::xs) = \\
& \quad \quad [\text{SOME } (\text{SLc } (\text{PL report1}))]) \wedge \\
& (\forall xs. \text{getReport } (\text{TT}::xs) = \text{getReport } xs) \wedge \\
& (\forall xs. \text{getReport } (\text{FF}::xs) = \text{getReport } xs) \wedge \\
& (\forall xs v_2. \text{getReport } (\text{prop } v_2::xs) = \text{getReport } xs) \wedge \\
& (\forall xs v_3. \text{getReport } (\text{notf } v_3::xs) = \text{getReport } xs) \wedge \\
& (\forall xs v_5 v_4. \text{getReport } (v_4 \text{ andf } v_5::xs) = \text{getReport } xs) \wedge \\
& (\forall xs v_7 v_6. \text{getReport } (v_6 \text{ orf } v_7::xs) = \text{getReport } xs) \wedge \\
& (\forall xs v_9 v_8. \text{getReport } (v_8 \text{ impf } v_9::xs) = \text{getReport } xs) \wedge
\end{aligned}$$

```

(∀ xs v11 v10. getReport (v10 eqf v11::xs) = getReport xs) ∧
(∀ xs v12. getReport (v12 says TT::xs) = getReport xs) ∧
(∀ xs v12. getReport (v12 says FF::xs) = getReport xs) ∧
(∀ xs v134.
    getReport (Name v134 says prop NONE::xs) = getReport xs) ∧
(∀ xs v146.
    getReport
        (Name PlatoonLeader says prop (SOME (ESCc v146))::xs) =
    getReport xs) ∧
(∀ xs.
    getReport
        (Name PlatoonLeader says
            prop (SOME (SLc (PL receiveMission)))::xs) =
    getReport xs) ∧
(∀ xs.
    getReport
        (Name PlatoonLeader says prop (SOME (SLc (PL warno)))::xs) =
    getReport xs) ∧
(∀ xs.
    getReport
        (Name PlatoonLeader says
            prop (SOME (SLc (PL tentativePlan)))::xs) =
    getReport xs) ∧
(∀ xs.
    getReport
        (Name PlatoonLeader says prop (SOME (SLc (PL recon)))::xs) =
    getReport xs) ∧
(∀ xs.
    getReport
        (Name PlatoonLeader says
            prop (SOME (SLc (PL completePlan)))::xs) =
    getReport xs) ∧
(∀ xs.
    getReport
        (Name PlatoonLeader says prop (SOME (SLc (PL opoid)))::xs) =
    getReport xs) ∧
(∀ xs.
    getReport
        (Name PlatoonLeader says
            prop (SOME (SLc (PL supervise)))::xs) =
    getReport xs) ∧
(∀ xs.
    getReport
        (Name PlatoonLeader says
            prop (SOME (SLc (PL report2)))::xs) =
    getReport xs) ∧

```

```

(∀ xs .
  getReport
  (Name PlatoonLeader says
    prop (SOME (SLc (PL complete))):xs) =
  getReport xs) ∧
(∀ xs .
  getReport
  (Name PlatoonLeader says
    prop (SOME (SLc (PL plIncomplete))):xs) =
  getReport xs) ∧
(∀ xs .
  getReport
  (Name PlatoonLeader says
    prop (SOME (SLc (PL invalidPlCommand))):xs) =
  getReport xs) ∧
(∀ xs v151 .
  getReport
  (Name PlatoonLeader says prop (SOME (SLc (PSG v151))):xs) =
  getReport xs) ∧
(∀ xs v144 .
  getReport
  (Name PlatoonSergeant says prop (SOME v144):xs) =
  getReport xs) ∧
(∀ xs v68 v136 v135 .
  getReport (v135 meet v136 says prop v68:xs) =
  getReport xs) ∧
(∀ xs v68 v138 v137 .
  getReport (v137 quoting v138 says prop v68:xs) =
  getReport xs) ∧
(∀ xs v69 v12 .
  getReport (v12 says notf v69:xs) = getReport xs) ∧
(∀ xs v71 v70 v12 .
  getReport (v12 says (v70 andf v71):xs) = getReport xs) ∧
(∀ xs v73 v72 v12 .
  getReport (v12 says (v72 orf v73):xs) = getReport xs) ∧
(∀ xs v75 v74 v12 .
  getReport (v12 says (v74 impf v75):xs) = getReport xs) ∧
(∀ xs v77 v76 v12 .
  getReport (v12 says (v76 eqf v77):xs) = getReport xs) ∧
(∀ xs v79 v78 v12 .
  getReport (v12 says v78 says v79:xs) = getReport xs) ∧
(∀ xs v81 v80 v12 .
  getReport (v12 says v80 speaks_for v81:xs) =
  getReport xs) ∧
(∀ xs v83 v82 v12 .
  getReport (v12 says v82 controls v83:xs) =
  getReport xs) ∧
(∀ xs v86 v85 v84 v12 .

```

```

getReport (v12 says reps v84 v85 v86::xs) =
getReport xs) ∧
(∀xs v88 v87 v12.
  getReport (v12 says v87 domi v88::xs) = getReport xs) ∧
(∀xs v90 v89 v12.
  getReport (v12 says v89 eqi v90::xs) = getReport xs) ∧
(∀xs v92 v91 v12.
  getReport (v12 says v91 doms v92::xs) = getReport xs) ∧
(∀xs v94 v93 v12.
  getReport (v12 says v93 eqs v94::xs) = getReport xs) ∧
(∀xs v96 v95 v12.
  getReport (v12 says v95 eqn v96::xs) = getReport xs) ∧
(∀xs v98 v97 v12.
  getReport (v12 says v97 lte v98::xs) = getReport xs) ∧
(∀xs v99 v12 v100.
  getReport (v12 says v99 lt v100::xs) = getReport xs) ∧
(∀xs v15 v14.
  getReport (v14 speaks_for v15::xs) = getReport xs) ∧
(∀xs v17 v16.
  getReport (v16 controls v17::xs) = getReport xs) ∧
(∀xs v20 v19 v18.
  getReport (reps v18 v19 v20::xs) = getReport xs) ∧
(∀xs v22 v21. getReport (v21 domi v22::xs) = getReport xs) ∧
(∀xs v24 v23. getReport (v23 eqi v24::xs) = getReport xs) ∧
(∀xs v26 v25. getReport (v25 doms v26::xs) = getReport xs) ∧
(∀xs v28 v27. getReport (v27 eqs v28::xs) = getReport xs) ∧
(∀xs v30 v29. getReport (v29 eqn v30::xs) = getReport xs) ∧
(∀xs v32 v31. getReport (v31 lte v32::xs) = getReport xs) ∧
  ∀xs v34 v33. getReport (v33 lt v34::xs) = getReport xs

```

[getReport_ind]

```

⊢ ∀P.
  P [] ∧
  (∀xs.
    P
      (Name PlatoonLeader says
        prop (SOME (SLc (PL report1))::xs)) ∧
      (∀xs. P xs ⇒ P (TT::xs)) ∧ (∀xs. P xs ⇒ P (FF::xs)) ∧
      (∀v2 xs. P xs ⇒ P (prop v2::xs)) ∧
      (∀v3 xs. P xs ⇒ P (notf v3::xs)) ∧
      (∀v4 v5 xs. P xs ⇒ P (v4 andf v5::xs)) ∧
      (∀v6 v7 xs. P xs ⇒ P (v6 orf v7::xs)) ∧
      (∀v8 v9 xs. P xs ⇒ P (v8 impf v9::xs)) ∧
      (∀v10 v11 xs. P xs ⇒ P (v10 eqf v11::xs)) ∧
      (∀v12 xs. P xs ⇒ P (v12 says TT::xs)) ∧
      (∀v12 xs. P xs ⇒ P (v12 says FF::xs)) ∧
      (∀v134 xs. P xs ⇒ P (Name v134 says prop NONE::xs)) ∧
      (∀v146 xs.
        P xs ⇒

```

```

P
  (Name PlatoonLeader says prop (SOME (ESCc v146))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL receiveMission)))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL warno)))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL tentativePlan)))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL recon)))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL completePlan)))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL opoid)))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL supervise)))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL report2)))::xs)) ∧
(∀ xs .
  P xs ⇒
  P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL complete)))::xs)) ∧
(∀ xs .

```

```

P xs ⇒
P
  (Name PlatoonLeader says
   prop (SOME (SLc (PL plIncomplete)))::xs)) ∧
(∀ xs.
  P xs ⇒
  P
    (Name PlatoonLeader says
     prop (SOME (SLc (PL invalidPlCommand)))::xs)) ∧
(∀ v151 xs.
  P xs ⇒
  P
    (Name PlatoonLeader says
     prop (SOME (SLc (PSG v151)))::xs)) ∧
(∀ v144 xs.
  P xs ⇒
  P (Name PlatoonSergeant says prop (SOME v144)::xs)) ∧
(∀ v135 v136 v68 xs.
  P xs ⇒ P (v135 meet v136 says prop v68::xs)) ∧
(∀ v137 v138 v68 xs.
  P xs ⇒ P (v137 quoting v138 says prop v68::xs)) ∧
(∀ v12 v69 xs. P xs ⇒ P (v12 says notf v69::xs)) ∧
(∀ v12 v70 v71 xs. P xs ⇒ P (v12 says (v70 andf v71)::xs)) ∧
(∀ v12 v72 v73 xs. P xs ⇒ P (v12 says (v72 orf v73)::xs)) ∧
(∀ v12 v74 v75 xs. P xs ⇒ P (v12 says (v74 impf v75)::xs)) ∧
(∀ v12 v76 v77 xs. P xs ⇒ P (v12 says (v76 eqf v77)::xs)) ∧
(∀ v12 v78 v79 xs. P xs ⇒ P (v12 says v78 says v79::xs)) ∧
(∀ v12 v80 v81 xs.
  P xs ⇒ P (v12 says v80 speaks_for v81::xs)) ∧
(∀ v12 v82 v83 xs.
  P xs ⇒ P (v12 says v82 controls v83::xs)) ∧
(∀ v12 v84 v85 v86 xs.
  P xs ⇒ P (v12 says reps v84 v85 v86::xs)) ∧
(∀ v12 v87 v88 xs. P xs ⇒ P (v12 says v87 domi v88::xs)) ∧
(∀ v12 v89 v90 xs. P xs ⇒ P (v12 says v89 eqi v90::xs)) ∧
(∀ v12 v91 v92 xs. P xs ⇒ P (v12 says v91 doms v92::xs)) ∧
(∀ v12 v93 v94 xs. P xs ⇒ P (v12 says v93 eqs v94::xs)) ∧
(∀ v12 v95 v96 xs. P xs ⇒ P (v12 says v95 eqn v96::xs)) ∧
(∀ v12 v97 v98 xs. P xs ⇒ P (v12 says v97 lte v98::xs)) ∧
(∀ v12 v99 v100 xs. P xs ⇒ P (v12 says v99 lt v100::xs)) ∧
(∀ v14 v15 xs. P xs ⇒ P (v14 speaks_for v15::xs)) ∧
(∀ v16 v17 xs. P xs ⇒ P (v16 controls v17::xs)) ∧
(∀ v18 v19 v20 xs. P xs ⇒ P (reps v18 v19 v20::xs)) ∧
(∀ v21 v22 xs. P xs ⇒ P (v21 domi v22::xs)) ∧
(∀ v23 v24 xs. P xs ⇒ P (v23 eqi v24::xs)) ∧
(∀ v25 v26 xs. P xs ⇒ P (v25 doms v26::xs)) ∧
(∀ v27 v28 xs. P xs ⇒ P (v27 eqs v28::xs)) ∧
(∀ v29 v30 xs. P xs ⇒ P (v29 eqn v30::xs)) ∧
(∀ v31 v32 xs. P xs ⇒ P (v31 lte v32::xs)) ∧

```

$$(\forall v_{33} v_{34} xs. P xs \Rightarrow P (v_{33} \text{ lt } v_{34} :: xs)) \Rightarrow \\ \forall v. P v$$

[getTenativePlan_def]

$$\vdash (\text{getTenativePlan} [] = [\text{NONE}]) \wedge \\ (\forall xs. \\ \text{getTenativePlan} \\ (\text{Name PlatoonLeader says} \\ \text{prop (SOME (SLc (PL tentativePlan))) :: xs}) = \\ [\text{SOME (SLc (PL tentativePlan))}]) \wedge \\ (\forall xs. \text{getTenativePlan (TT :: xs)} = \text{getTenativePlan xs}) \wedge \\ (\forall xs. \text{getTenativePlan (FF :: xs)} = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_2. \\ \text{getTenativePlan (prop } v_2 :: xs) = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_3. \\ \text{getTenativePlan (notf } v_3 :: xs) = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_5 v_4. \\ \text{getTenativePlan (v}_4 \text{ andf } v_5 :: xs) = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_7 v_6. \\ \text{getTenativePlan (v}_6 \text{ orf } v_7 :: xs) = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_9 v_8. \\ \text{getTenativePlan (v}_8 \text{ impf } v_9 :: xs) = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_{11} v_{10}. \\ \text{getTenativePlan (v}_{10} \text{ eqf } v_{11} :: xs) = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_{12}. \\ \text{getTenativePlan (v}_{12} \text{ says TT :: xs}) = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_{12}. \\ \text{getTenativePlan (v}_{12} \text{ says FF :: xs}) = \text{getTenativePlan xs}) \wedge \\ (\forall xs v_{134}. \\ \text{getTenativePlan (Name } v_{134} \text{ says prop NONE :: xs}) = \\ \text{getTenativePlan xs}) \wedge \\ (\forall xs v_{146}. \\ \text{getTenativePlan} \\ (\text{Name PlatoonLeader says prop (SOME (ESCc } v_{146})) :: xs) = \\ \text{getTenativePlan xs}) \wedge \\ (\forall xs. \\ \text{getTenativePlan} \\ (\text{Name PlatoonLeader says} \\ \text{prop (SOME (SLc (PL receiveMission))) :: xs}) = \\ \text{getTenativePlan xs}) \wedge \\ (\forall xs. \\ \text{getTenativePlan} \\ (\text{Name PlatoonLeader says} \\ \text{prop (SOME (SLc (PL warno))) :: } \\ xs) = \\ \text{getTenativePlan xs}) \wedge \\ (\forall xs. \\ \text{getTenativePlan} \\ (\text{Name PlatoonLeader says} \\ \text{prop (SOME (SLc (PL recon))) :: } \\ xs) =$$

```

getTenativePlan xs) ∧
(∀ xs .
getTenativePlan
  (Name PlatoonLeader says
    prop (SOME (SLc (PL report1))))::xs) =
getTenativePlan xs) ∧
(∀ xs .
getTenativePlan
  (Name PlatoonLeader says
    prop (SOME (SLc (PL completePlan))))::xs) =
getTenativePlan xs) ∧
(∀ xs .
getTenativePlan
  (Name PlatoonLeader says prop (SOME (SLc (PL opoid))))::
  xs) =
getTenativePlan xs) ∧
(∀ xs .
getTenativePlan
  (Name PlatoonLeader says
    prop (SOME (SLc (PL supervise))))::xs) =
getTenativePlan xs) ∧
(∀ xs .
getTenativePlan
  (Name PlatoonLeader says
    prop (SOME (SLc (PL report2))))::xs) =
getTenativePlan xs) ∧
(∀ xs .
getTenativePlan
  (Name PlatoonLeader says
    prop (SOME (SLc (PL complete))))::xs) =
getTenativePlan xs) ∧
(∀ xs .
getTenativePlan
  (Name PlatoonLeader says
    prop (SOME (SLc (PL plIncomplete))))::xs) =
getTenativePlan xs) ∧
(∀ xs .
getTenativePlan
  (Name PlatoonLeader says
    prop (SOME (SLc (PL invalidPlCommand))))::xs) =
getTenativePlan xs) ∧
(∀ xs v151 .
getTenativePlan
  (Name PlatoonLeader says prop (SOME (SLc (PSG v151))))::
  xs) =
getTenativePlan xs) ∧
(∀ xs v144 .
getTenativePlan
  (Name PlatoonSergeant says prop (SOME v144)::xs) =

```

```

        getTenativePlan xs) ∧
(∀ xs v68 v136 v135.
    getTenativePlan (v135 meet v136 says prop v68::xs) =
    getTenativePlan xs) ∧
(∀ xs v68 v138 v137.
    getTenativePlan (v137 quoting v138 says prop v68::xs) =
    getTenativePlan xs) ∧
(∀ xs v69 v12.
    getTenativePlan (v12 says notf v69::xs) =
    getTenativePlan xs) ∧
(∀ xs v71 v70 v12.
    getTenativePlan (v12 says (v70 andf v71)::xs) =
    getTenativePlan xs) ∧
(∀ xs v73 v72 v12.
    getTenativePlan (v12 says (v72 orf v73)::xs) =
    getTenativePlan xs) ∧
(∀ xs v75 v74 v12.
    getTenativePlan (v12 says (v74 impf v75)::xs) =
    getTenativePlan xs) ∧
(∀ xs v77 v76 v12.
    getTenativePlan (v12 says (v76 eqf v77)::xs) =
    getTenativePlan xs) ∧
(∀ xs v79 v78 v12.
    getTenativePlan (v12 says v78 says v79::xs) =
    getTenativePlan xs) ∧
(∀ xs v81 v80 v12.
    getTenativePlan (v12 says v80 speaks_for v81::xs) =
    getTenativePlan xs) ∧
(∀ xs v83 v82 v12.
    getTenativePlan (v12 says v82 controls v83::xs) =
    getTenativePlan xs) ∧
(∀ xs v86 v85 v84 v12.
    getTenativePlan (v12 says reps v84 v85 v86::xs) =
    getTenativePlan xs) ∧
(∀ xs v88 v87 v12.
    getTenativePlan (v12 says v87 domi v88::xs) =
    getTenativePlan xs) ∧
(∀ xs v90 v89 v12.
    getTenativePlan (v12 says v89 equi v90::xs) =
    getTenativePlan xs) ∧
(∀ xs v92 v91 v12.
    getTenativePlan (v12 says v91 doms v92::xs) =
    getTenativePlan xs) ∧
(∀ xs v94 v93 v12.
    getTenativePlan (v12 says v93 eqs v94::xs) =
    getTenativePlan xs) ∧
(∀ xs v96 v95 v12.
    getTenativePlan (v12 says v95 eqn v96::xs) =
    getTenativePlan xs) ∧

```

```

(∀ xs v98 v97 v12.
  getTenativePlan (v12 says v97 lte v98::xs) =
  getTenativePlan xs) ∧
(∀ xs v99 v12 v100.
  getTenativePlan (v12 says v99 lt v100::xs) =
  getTenativePlan xs) ∧
(∀ xs v15 v14.
  getTenativePlan (v14 speaks_for v15::xs) =
  getTenativePlan xs) ∧
(∀ xs v17 v16.
  getTenativePlan (v16 controls v17::xs) =
  getTenativePlan xs) ∧
(∀ xs v20 v19 v18.
  getTenativePlan (reps v18 v19 v20::xs) =
  getTenativePlan xs) ∧
(∀ xs v22 v21.
  getTenativePlan (v21 domi v22::xs) = getTenativePlan xs) ∧
(∀ xs v24 v23.
  getTenativePlan (v23 eqi v24::xs) = getTenativePlan xs) ∧
(∀ xs v26 v25.
  getTenativePlan (v25 doms v26::xs) = getTenativePlan xs) ∧
(∀ xs v28 v27.
  getTenativePlan (v27 eqs v28::xs) = getTenativePlan xs) ∧
(∀ xs v30 v29.
  getTenativePlan (v29 eqn v30::xs) = getTenativePlan xs) ∧
(∀ xs v32 v31.
  getTenativePlan (v31 lte v32::xs) = getTenativePlan xs) ∧
  ∃ xs v34 v33.
    getTenativePlan (v33 lt v34::xs) = getTenativePlan xs

```

[getTenativePlan_ind]

```

⊢ ∀ P.
  P [] ∧
  (∀ xs.
    P
    (Name PlatoonLeader says
      prop (SOME (SLc (PL tentativePlan)))::xs)) ∧
    (∀ xs. P xs ⇒ P (TT::xs)) ∧ (∀ xs. P xs ⇒ P (FF::xs)) ∧
    (∀ v2 xs. P xs ⇒ P (prop v2::xs)) ∧
    (∀ v3 xs. P xs ⇒ P (notf v3::xs)) ∧
    (∀ v4 v5 xs. P xs ⇒ P (v4 andf v5::xs)) ∧
    (∀ v6 v7 xs. P xs ⇒ P (v6 orf v7::xs)) ∧
    (∀ v8 v9 xs. P xs ⇒ P (v8 impf v9::xs)) ∧
    (∀ v10 v11 xs. P xs ⇒ P (v10 eqf v11::xs)) ∧
    (∀ v12 xs. P xs ⇒ P (v12 says TT::xs)) ∧
    (∀ v12 xs. P xs ⇒ P (v12 says FF::xs)) ∧
    (∀ v134 xs. P xs ⇒ P (Name v134 says prop NONE::xs)) ∧
    (∀ v146 xs.
      P xs ⇒

```

```


$$\begin{aligned}
& P \\
& \quad (\text{Name PlatoonLeader says prop (SOME (ESCc v146))} :: \\
& \quad \quad xs) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL receiveMission)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL warno)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL recon)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL report1)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL completePlan)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL opoid)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL supervise)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL report2)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \quad \text{prop (SOME (SLc (PL complete)))} :: \\
& \quad \quad \quad xs)) \wedge \\
(\forall xs. &
\end{aligned}$$


```

```

P xs ⇒
P
  (Name PlatoonLeader says
   prop (SOME (SLc (PL plIncomplete)))::xs)) ∧
(∀ xs.
  P xs ⇒
  P
    (Name PlatoonLeader says
     prop (SOME (SLc (PL invalidPlCommand)))::xs)) ∧
(∀ v151 xs.
  P xs ⇒
  P
    (Name PlatoonLeader says
     prop (SOME (SLc (PSG v151)))::xs)) ∧
(∀ v144 xs.
  P xs ⇒
  P (Name PlatoonSergeant says prop (SOME v144)::xs)) ∧
(∀ v135 v136 v68 xs.
  P xs ⇒ P (v135 meet v136 says prop v68::xs)) ∧
(∀ v137 v138 v68 xs.
  P xs ⇒ P (v137 quoting v138 says prop v68::xs)) ∧
(∀ v12 v69 xs. P xs ⇒ P (v12 says notf v69::xs)) ∧
(∀ v12 v70 v71 xs. P xs ⇒ P (v12 says (v70 andf v71)::xs)) ∧
(∀ v12 v72 v73 xs. P xs ⇒ P (v12 says (v72 orf v73)::xs)) ∧
(∀ v12 v74 v75 xs. P xs ⇒ P (v12 says (v74 impf v75)::xs)) ∧
(∀ v12 v76 v77 xs. P xs ⇒ P (v12 says (v76 eqf v77)::xs)) ∧
(∀ v12 v78 v79 xs. P xs ⇒ P (v12 says v78 says v79::xs)) ∧
(∀ v12 v80 v81 xs.
  P xs ⇒ P (v12 says v80 speaks_for v81::xs)) ∧
(∀ v12 v82 v83 xs.
  P xs ⇒ P (v12 says v82 controls v83::xs)) ∧
(∀ v12 v84 v85 v86 xs.
  P xs ⇒ P (v12 says reps v84 v85 v86::xs)) ∧
(∀ v12 v87 v88 xs. P xs ⇒ P (v12 says v87 domi v88::xs)) ∧
(∀ v12 v89 v90 xs. P xs ⇒ P (v12 says v89 eqi v90::xs)) ∧
(∀ v12 v91 v92 xs. P xs ⇒ P (v12 says v91 doms v92::xs)) ∧
(∀ v12 v93 v94 xs. P xs ⇒ P (v12 says v93 eqs v94::xs)) ∧
(∀ v12 v95 v96 xs. P xs ⇒ P (v12 says v95 eqn v96::xs)) ∧
(∀ v12 v97 v98 xs. P xs ⇒ P (v12 says v97 lte v98::xs)) ∧
(∀ v12 v99 v100 xs. P xs ⇒ P (v12 says v99 lt v100::xs)) ∧
(∀ v14 v15 xs. P xs ⇒ P (v14 speaks_for v15::xs)) ∧
(∀ v16 v17 xs. P xs ⇒ P (v16 controls v17::xs)) ∧
(∀ v18 v19 v20 xs. P xs ⇒ P (reps v18 v19 v20::xs)) ∧
(∀ v21 v22 xs. P xs ⇒ P (v21 domi v22::xs)) ∧
(∀ v23 v24 xs. P xs ⇒ P (v23 eqi v24::xs)) ∧
(∀ v25 v26 xs. P xs ⇒ P (v25 doms v26::xs)) ∧
(∀ v27 v28 xs. P xs ⇒ P (v27 eqs v28::xs)) ∧
(∀ v29 v30 xs. P xs ⇒ P (v29 eqn v30::xs)) ∧
(∀ v31 v32 xs. P xs ⇒ P (v31 lte v32::xs)) ∧

```

$$\begin{array}{c} (\forall v_{33} \ v_{34} \ xs. \ P \ xs \Rightarrow P \ (v_{33} \ 1t \ v_{34} :: xs)) \Rightarrow \\ \forall v. \ P \ v \end{array}$$

Index

ConductORPDef Theory, 46

- Definitions, 46
 - secAuthorization_def, 46
 - secContext_def, 46
 - secHelper_def, 47
- Theorems, 47
 - getOmniCommand_def, 47
 - getOmniCommand_ind, 49
 - getPlCom_def, 50
 - getPlCom_ind, 51
 - getPsgCom_def, 51
 - getPsgCom_ind, 51

ConductORPType Theory, 44

- Datatypes, 44
- Theorems, 45
 - omniCommand_distinct_clauses, 45
 - plCommand_distinct_clauses, 45
 - psgCommand_distinct_clauses, 45
 - slCommand_distinct_clauses, 45
 - slCommand_one_one, 45
 - slOutput_distinct_clauses, 45
 - slRole_distinct_clauses, 46
 - slState_distinct_clauses, 46

ConductPBType Theory, 57

- Datatypes, 57
- Theorems, 57
 - plCommandPB_distinct_clauses, 57
 - psgCommandPB_distinct_clauses, 57
 - slCommand_distinct_clauses, 57
 - slCommand_one_one, 57
 - slOutput_distinct_clauses, 57
 - slRole_distinct_clauses, 58
 - slState_distinct_clauses, 58

MoveToORPType Theory, 62

- Datatypes, 62
- Theorems, 63
 - slCommand_distinct_clauses, 63
 - slOutput_distinct_clauses, 63
 - slState_distinct_clauses, 63

MoveToPBType Theory, 68

- Datatypes, 68
- Theorems, 68
 - slCommand_distinct_clauses, 68
 - slOutput_distinct_clauses, 68
 - slState_distinct_clauses, 69

OMNIType Theory, 3

- Datatypes, 3
- Theorems, 3
 - command_distinct_clauses, 3
 - command_one_one, 3
 - escCommand_distinct_clauses, 3
 - escOutput_distinct_clauses, 3
 - escState_distinct_clauses, 3
 - output_distinct_clauses, 4
 - output_one_one, 4
 - principal_one_one, 4
 - state_distinct_clauses, 4
 - state_one_one, 4

PBIntegratedDef Theory, 23

- Definitions, 23
 - secAuthorization_def, 23
 - secContext_def, 23
 - secHelper_def, 24
- Theorems, 24
 - getOmniCommand_def, 24
 - getOmniCommand_ind, 27
 - getPlCom_def, 28
 - getPlCom_ind, 28

PBTypeIntegrated Theory, 21

- Datatypes, 21
- Theorems, 22
 - omniCommand_distinct_clauses, 22
 - plCommand_distinct_clauses, 22
 - slCommand_distinct_clauses, 22
 - slCommand_one_one, 22
 - slOutput_distinct_clauses, 23
 - slState_distinct_clauses, 23
 - stateRole_distinct_clauses, 23

PlanPBDef Theory, 82

Definitions, 82
 PL_notWARNO_Auth_def, 82
 PL_WARNO_Auth_def, 82
 secContext_def, 82
 secContextNull_def, 83
 Theorems, 83
 getInitMove_def, 83
 getInitMove_ind, 85
 getPlCom_def, 87
 getPlCom_ind, 89
 getPsgCom_def, 90
 getPsgCom_ind, 92
 getRecon_def, 93
 getRecon_ind, 96
 getReport_def, 98
 getReport_ind, 101
 getTentativePlan_def, 104
 getTentativePlan_ind, 107
PlanPBType Theory, 79
 Datatypes, 79
 Theorems, 79
 plCommand_distinct_clauses, 79
 psgCommand_distinct_clauses, 80
 slCommand_distinct_clauses, 80
 slCommand_one_one, 80
 slOutput_distinct_clauses, 80
 slRole_distinct_clauses, 81
 slState_distinct_clauses, 81
satList Theory, 21
 Definitions, 21
 satList_def, 21
 Theorems, 21
 satList_conj, 21
 satList_CONS, 21
 satList_nil, 21
ssm Theory, 11
 Datatypes, 11
 Definitions, 12
 authenticationTest_def, 12
 commandList_def, 12
 inputList_def, 12
 propCommandList_def, 12
 TR_def, 12
 Theorems, 13
 CFGInterpret_def, 13
 CFGInterpret_ind, 13
 configuration_one_one, 13
 extractCommand_def, 13
 extractCommand_ind, 13
 extractInput_def, 14
 extractInput_ind, 14
 extractPropCommand_def, 15
 extractPropCommand_ind, 15
 TR_cases, 16
 TR_discard_cmd_rule, 17
 TR_EQ_rules_thm, 17
 TR_exec_cmd_rule, 17
 TR_ind, 18
 TR_rules, 18
 TR_strongind, 19
 TR_trap_cmd_rule, 20
 TRrule0, 20
 TRrule1, 20
 trType_distinct_clauses, 20
 trType_one_one, 21
ssm11 Theory, 4
 Datatypes, 4
 Definitions, 4
 TR_def, 4
 Theorems, 5
 CFGInterpret_def, 5
 CFGInterpret_ind, 6
 configuration_one_one, 6
 order_distinct_clauses, 6
 order_one_one, 6
 TR_cases, 6
 TR_discard_cmd_rule, 7
 TR_EQ_rules_thm, 7
 TR_exec_cmd_rule, 8
 TR_ind, 8
 TR_rules, 9
 TR_strongind, 9
 TR_trap_cmd_rule, 10
 TRrule0, 10
 TRrule1, 11
 trType_distinct_clauses, 11

trType_one_one, 11
ssmConductORP Theory, 35
 Theorems, 35
 conductORPNS_def, 35
 conductORPNS_ind, 35
 conductORPOut_def, 36
 conductORPOut_ind, 36
 inputOK_cmd_reject_lemma, 36
 inputOK_def, 36
 inputOK_ind, 37
 PlatoonLeader_ACTIONS_IN_exec_jus-
 tified_lemma, 38
 PlatoonLeader_ACTIONS_IN_exec_jus-
 tified_thm, 39
 PlatoonLeader_ACTIONS_IN_exec_lemma,
 39
 PlatoonLeader_ACTIONS_IN_trap_jus-
 tified_lemma, 40
 PlatoonLeader_ACTIONS_IN_trap_jus-
 tified_thm, 41
 PlatoonLeader_ACTIONS_IN_trap_lemma,
 41
 PlatoonLeader_CONDUCT_ORP_exec_-
 secure_justified_thm, 41
 PlatoonLeader_CONDUCT_ORP_exec_-
 secure_lemma, 42
 PlatoonSergeant_SECURE_exec_justi-
 fied_lemma, 42
 PlatoonSergeant_SECURE_exec_justi-
 fied_thm, 43
 PlatoonSergeant_SECURE_exec_lemma,
 44
ssmConductPB Theory, 51
 Definitions, 51
 secContextConductPB_def, 51
 ssmConductPBStateInterp_def, 52
 Theorems, 52
 authTestConductPB_cmd_reject_lemma,
 52
 authTestConductPB_def, 52
 authTestConductPB_ind, 53
 conductPBNS_def, 53
 conductPBNS_ind, 54
 conductPBOut_def, 54
 conductPBOut_ind, 55
 PlatoonLeader_exec_plCommandPB_-
 justified_thm, 55
 PlatoonLeader_plCommandPB_lemma,
 56
 PlatoonSergeant_exec_psgCommandPB_-
 justified_thm, 56
 PlatoonSergeant_psgCommandPB_lemma,
 56
ssmMoveToORP Theory, 58
 Definitions, 58
 secContextMoveToORP_def, 58
 ssmMoveToORPStateInterp_def, 58
 Theorems, 58
 authTestMoveToORP_cmd_reject_lemma,
 58
 authTestMoveToORP_def, 58
 authTestMoveToORP_ind, 59
 moveToORPNS_def, 60
 moveToORPNS_ind, 60
 moveToORPOut_def, 61
 moveToORPOut_ind, 61
 PlatoonLeader_exec_slCommand_jus-
 tified_thm, 62
 PlatoonLeader_slCommand_lemma, 62
ssmMoveToPB Theory, 63
 Definitions, 63
 secContextMoveToPB_def, 63
 ssmMoveToPBStateInterp_def, 64
 Theorems, 64
 authTestMoveToPB_cmd_reject_lemma,
 64
 authTestMoveToPB_def, 64
 authTestMoveToPB_ind, 65
 moveToPBNS_def, 65
 moveToPBNS_ind, 66
 moveToPBOut_def, 66
 moveToPBOut_ind, 66
 PlatoonLeader_exec_slCommand_jus-
 tified_thm, 67
 PlatoonLeader_slCommand_lemma, 68
ssmPBIntegrated Theory, 28

Theorems, 28

- inputOK_cmd_reject_lemma, 28
- inputOK_def, 28
- inputOK_ind, 29
- PBNS_def, 30
- PBNS_ind, 30
- PBOut_def, 30
- PBOut_ind, 31
- PlatoonLeader_Omni_notDiscard_slCommand_thm, 31
- PlatoonLeader_PLAN_PB_exec_justified_lemma, 31
- PlatoonLeader_PLAN_PB_exec_justified_thm, 32
- PlatoonLeader_PLAN_PB_exec_lemma, 33
- PlatoonLeader_PLAN_PB_trap_justified_lemma, 33
- PlatoonLeader_PLAN_PB_trap_justified_thm, 34
- PlatoonLeader_PLAN_PB_trap_lemma, 35

ssmPlanPB Theory, 69

Theorems, 69

- inputOK_def, 69
- inputOK_ind, 70
- planPBNS_def, 70
- planPBNS_ind, 71
- planPBOut_def, 71
- planPBOut_ind, 72
- PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma, 72
- PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm, 73
- PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma, 73
- PlatoonLeader_psgCommand_notDiscard_thm, 74
- PlatoonLeader_trap_psgCommand_justified_lemma, 74
- PlatoonLeader_trap_psgCommand_lemma, 74
- PlatoonLeader_WARNO_exec_report1_justified_lemma, 75
- PlatoonLeader_WARNO_exec_report1_justified_thm, 76
- PlatoonLeader_WARNO_exec_report1_lemma, 77
- PlatoonSergeant_trap_plCommand_justified_lemma, 77
- PlatoonSergeant_trap_plCommand_justified_thm, 78
- PlatoonSergeant_trap_plCommand_lemma, 78

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 interpretation of the state, (3) the security context, *)
(* (4) the input stream, (5) the state, and (6) the output stream. *)
(* -----
*)
val _ =
Datatype
`configuration =
CFG
  (('command option , 'principal , 'd , 'e)Form -> bool)
  (('state -> ('command option , 'principal , 'd , 'e)Form list ->
    ('command option , 'principal , 'd , 'e)Form list))
  (((('command option , 'principal , 'd , 'e)Form list) ->
    (('command option , 'principal , 'd , 'e)Form list))
  (((('command option , 'principal , 'd , 'e)Form) list) list)
  ('state)
  ('output list)`

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

```

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 *)

```

```

(*      ( $M, Oi, Os$ ) *)  

(*      ( $\text{CFG elementTest stateInterpret certList}$ ) *)  

(*      ( $((P \text{ says } (\text{prop } (\text{CMD cmd})))::ins) s outs$ ) ==> *)  

(*      ( $((M, Oi, Os) \text{ sat } (\text{prop } (\text{CMD cmd})))$ ) *)  

(*      is a valid inference rule, then executing cmd the exec(CMD cmd) transition *)  

(*      occurs if and only if prop (CMD cmd), elementTest, and *)  

(*      CFGInterpret ( $M, Oi, Os$ ) *)  

(*      ( $\text{CFG elementTest stateInterpret certList } (P \text{ says } \text{prop } (\text{CMD cmd})::ins) s outs$ ) *)  

(*      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  

(*      ( $M, Oi, Os$ ) *)  

(*      ( $\text{CFG elementTest stateInterpret certList}$ ) *)  

(*      ( $((P \text{ says } (\text{prop } (\text{CMD cmd})))::ins) s outs$ ) ==> *)  

(*      ( $((M, Oi, Os) \text{ sat } (\text{prop } \text{TRAP}))$ ) *)  

(*      is a valid inference rule, then executing cmd the trap(CMD cmd) transition *)  

(*      occurs if and only if prop TRAP, elementTest, and *)  

(*      CFGInterpret ( $M, Oi, Os$ ) *)  

(*      ( $\text{CFG elementTest stateInterpret certList } (P \text{ says } \text{prop } (\text{CMD cmd})::ins)$ ) *)  

(*      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))) ==>
```

```

(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(
[],``!11 12 M Oi Os.(((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po))
satList 11) /\
((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po))
satList 12) =
(((M:('prop,'world,'pName,'Int,'Sec)Kripke),(Oi:'Int po),(Os:'Sec po))
satList (11 ++ 12))``),
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 escapeLevel

D.3 TopLevel

D.3.1 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  

| ssmMoveToORPCComplete  

| ssmConductORPCComplete  

| 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.3.2 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",
"PBIIntegratedDefTheory", "PBTypeIntegratedTheory"];

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

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

val _ = new_theory "PBIIntegratedDef";

(* ****
(* Helper functions for extracting commands
(* ****)
val getPlCom_def =
Define` 
  (getPlCom ([]:(slCommand command)option)list)
    = incomplete:plCommand) /\
  (getPlCom (SOME (SLc (PL cmd)):(slCommand command)option :: xs))
    = cmd:plCommand) /\
  (getPlCom (_ ::( xs :(slCommand command)option list)))
    = (getPlCom xs))` 

(* _____
(* state Interpretation function
(* _____
(* This function doesn't do anything but is necessary to specialize other
(* theorems.
(* _____

```

```

(* ----- *)  

(*  

val secContextNull_def = Define `  

  secContext (x:((slCommand command)option , stateRole , 'd , 'e)Form list) =  

    [(Name Omni) controls prop (SOME (SLc (OMNI omniCommand)))  

     :((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) says prop (SOME (SLc (OMNI cmd))))):xs)  

   = (cmd:omniCommand)) /\  

  (getOmniCommand ((x:((slCommand command)option , stateRole , 'd , 'e)Form):xs)  

   = (getOmniCommand xs))`  

val secAuthorization_def =  

Define `  

  (secAuthorization (xs:((slCommand command)option , stateRole , 'd , 'e)Form list)  

   = secHelper (getOmniCommand xs))`  

val secContext_def =  

Define `  

  (secContext (PLAN_PB) (xs:((slCommand command)option , stateRole , 'd , 'e)Form list) =  

   if ((getOmniCommand xs) = ssmPlanPBComplete:omniCommand)  

   then  

     [(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]  

   else [prop NONE:((slCommand command)option , stateRole , 'd , 'e)Form]) /\  

  (secContext (MOVE_TO_ORP) (xs:((slCommand command)option , stateRole , 'd , 'e)Form list) =  

   if (getOmniCommand xs = ssmMoveToORPComplete)  

   then  

     [prop (SOME (SLc (OMNI (ssmMoveToORPComplete)))) impf  

      (Name PlatoonLeader) controls prop (SOME (SLc (PL conductORP)))]  

   else [prop NONE]) /\  

  (secContext (CONDUCT_ORP) (xs:((slCommand command)option , stateRole , 'd , 'e)Form list) =  

   if (getOmniCommand xs = ssmConductORPComplete)  

   then  

     [prop (SOME (SLc (OMNI (ssmConductORPComplete)))) impf  

      (Name PlatoonLeader) controls prop (SOME (SLc (PL moveToPB)))]  

   else [prop NONE]) /\  

  (secContext (MOVE_TO_PB) (xs:((slCommand command)option , stateRole , 'd , 'e)Form list) =  

   if (getOmniCommand xs = ssmConductORPComplete)  

   then  

     [prop (SOME (SLc (OMNI (ssmMoveToPBComplete)))) impf  

      (Name PlatoonLeader) controls prop (SOME (SLc (PL conductPB)))]  

   else [prop NONE]) /\  

  (secContext (CONDUCT_PB) (xs:((slCommand command)option , stateRole , 'd , 'e)Form list) =  

   if (getOmniCommand xs = ssmConductPBComplete)  

   then  

     [prop (SOME (SLc (OMNI (ssmConductPBComplete)))) impf  

      (Name PlatoonLeader) controls prop (SOME (SLc (PL completePB)))]  

   else [prop NONE])`  

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

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

val _ = export_theory();  

end

```

D.3.3 ssmPBIntegrated Theory: Theorems

```
(* ****
(* ssmPBIntegratedTheory
(* Author: Lori Pickering
(* Date: 7 May 2018
(* This theory aims to integrate the topLevel ssm with the sublevel ssms. It *)
(* does this by adding a condition to the security context. In particular, *)
(* it requires that the "COMPLETE" state in the subLevel ssm must precede *)
(* transition to the next state at the 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", "PBTTypeIntegratedTheory", "PBIntegratedDefTheory",
    "ssmPBIntegratedTheory"];
  open TypeBase listTheory optionTheory listSyntax
  acl_infRules aclDrulesTheory aclrulesTheory
  aclsemanticsTheory aclfoundationTheory
  satListTheory ssmTheory ssminfRules uavUtilities
  OMNITypeTheory PBTTypeIntegratedTheory 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 PBTTypeIntegratedTheory PBIntegratedDefTheory

  val _ = new_theory "ssmPBIntegrated";
  (* ****
  (* Define next-state and next-output functions
  (* ****)
  val PBNS_def =
  Define `

  (PBNS PLAN_PB (exec x) =
    if (getPlCom x) = crossLD then MOVE_TO_ORP else PLAN_PB) /\

  (PBNS MOVE_TO_ORP (exec x) =
    if (getPlCom x) = conductORP then CONDUCT_ORP else MOVE_TO_ORP) /\

  (PBNS CONDUCT_ORP (exec x) =
    if (getPlCom x) = moveToPB then MOVE_TO_PB else CONDUCT_ORP) /\

  (PBNS MOVE_TO_PB (exec x) =
    if (getPlCom x) = conductPB then CONDUCT_PB else MOVE_TO_PB) /\

  (PBNS CONDUCT_PB (exec x) =
    if (getPlCom x) = completePB then COMPLETE_PB else CONDUCT_PB) /\

  (PBNS (s:slState) (trap _) = s) /\

  (PBNS (s:slState) (discard _) = s)`)

  val PBOut_def =
  Define `

  (PBOut PLAN_PB (exec x) =
    if (getPlCom x) = crossLD then MoveToORP else PlanPB) /\

  (PBOut MOVE_TO_ORP (exec x) =
    if (getPlCom x) = conductORP then ConductORP else MoveToORP) /\
```

```

(PBOut CONDUCT_ORP (exec x) =
  if (getPICom x) = moveToPB then MoveToORP else ConductORP) /\ 
(PBOut MOVE_TO_PB (exec x) =
  if (getPICom x) = conductPB then ConductPB else MoveToPB) /\ 
(PBOut CONDUCT_PB (exec x) =
  if (getPICom x) = completePB then CompletePB else ConductPB) /\ 
(PBOut (s:slState) (trap _) = unAuthorized) /\ 
(PBOut (s:slState) (discard _) = unAuthenticated)`

(* ****)
(* Define authentication function *)
(* ****)
val inputOK_def =
Define`  


(* ****)
(* 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]))`  

val _ = save_thm("inputOK_cmd_reject_lemma",
                  inputOK_cmd_reject_lemma)
(* _____ *)
(* Theorem: PlatoonLeader is authorized on crossLD if
(*   Omni says ssmPlanPBComplete
(* _____ *)
val thPlanPB =
ISPECL
[``inputOK:((slCommand command) option , stateRole , 'd , 'e)Form -> bool`` ,
 ``secAuthorization :((slCommand command) option , stateRole , 'd , 'e)Form list ->
   ((slCommand command) option , stateRole , 'd , 'e)Form list `` ,
 ``secContext: (slState) ->
   ((slCommand command) option , stateRole , 'd , 'e)Form list ->
   ((slCommand command) option , stateRole , 'd , 'e)Form list `` ,
 ``[(Name Omni) says (prop (SOME (SLC (OMNI ssmPlanPBComplete))) :
         ((slCommand command) option , stateRole , 'd , 'e)Form ;
        (Name PlatoonLeader) says (prop (SOME (SLC (PL crossLD)))) :
         ((slCommand command) option , stateRole , 'd , 'e)Form] `` ,
 ``ins:((slCommand command) option , stateRole , 'd , 'e)Form list list `` ,
 ``(PLAN_PB)` ,
 ``outs:slOutput output list trType list ``] TR_exec_cmd_rule`  

val PlatoonLeader_PLAN_PB_exec_lemma =
TAC PROOF(
  ([] , fst(dest_imp(concl thPlanPB))) ,
  REWRITE_TAC[CFGInterpret_def , secContext_def , secAuthorization_def , secHelper_def ,
             propCommandList_def , extractPropCommand_def , inputList_def ,
             getOmniCommand_def ,
             MAP , extractInput_def , satList_CONS , satList_nil , GSYM satList_conj] THEN
  PROVE_TAC[Controls , Modus_Ponens])  

val _ = save_thm("PlatoonLeader_PLAN_PB_exec_lemma",
                  PlatoonLeader_PLAN_PB_exec_lemma)  

val PlatoonLeader_PLAN_PB_exec_justified_lemma =
TAC PROOF(
  ([] , snd(dest_imp(concl thPlanPB))) ,
  PROVE_TAC[PlatoonLeader_PLAN_PB_exec_lemma , TR_exec_cmd_rule])  

val _ = save_thm("PlatoonLeader_PLAN_PB_exec_justified_lemma",

```

```

PlatoonLeader _ PLAN _ PB _ exec _ justified _ lemma)

val PlatoonLeader _ PLAN _ PB _ exec _ justified _ thm =
REWRITE_RULE[inputList _ def , extractInput _ def , MAP, propCommandList _ def ,
extractPropCommand _ def , PlatoonLeader _ PLAN _ PB _ exec _ lemma]
PlatoonLeader _ PLAN _ PB _ exec _ justified _ lemma

val _ = save _ thm ("PlatoonLeader _ PLAN _ PB _ exec _ justified _ thm",
PlatoonLeader _ PLAN _ PB _ exec _ justified _ lemma)

(* _____ *)
(* Theorem: PlatoonLeader is trapped on crossLD if *)
(* state = PLAN_PB and *)
(* and not Omni says ssmPlanPBComplete *)
(* _____ *)
*)  

val thPlanPBTrap =
ISPECL
[ ` inputOK:((slCommand command)option , stateRole , 'd,'e)Form -> bool `` ,
` ` secAuthorization :((slCommand command)option , stateRole , 'd,'e)Form list ->
((slCommand command)option , stateRole , 'd,'e)Form list `` ,
` ` secContext: (s1State) ->
((slCommand command)option , stateRole , 'd,'e)Form list ->
((slCommand command)option , stateRole , 'd,'e)Form list `` ,
` ` [(Name Omni) says (prop (SOME (SLc (OMNI omniCommand)))) :
((slCommand command)option , stateRole , 'd,'e)Form;
(Name PlatoonLeader) says (prop (SOME (SLc (PL crossLD)))) :
((slCommand command)option , stateRole , 'd,'e)Form] `` ,
` ` ins:((slCommand command)option , stateRole , 'd,'e)Form list list `` ,
` ` (PLAN_PB) `` ,
` ` outs:slOutput output list trType list `` ] TR_trap_cmd_rule

val temp2 = fst(dest _ imp(concl thPlanPBTrap))

val PlatoonLeader _ PLAN _ PB _ trap _ lemma =
TAC PROOF(
[],  

Term`(~(omniCommand:omniCommand) = ssmPlanPBComplete)) ==>
((s:s1State) = PLAN_PB) ==>
^temp2`),  

DISCH_TAC THEN  

DISCH_TAC THEN  

ASM REWRITE_TAC[CFGInterpret _ def , secContext _ def , secAuthorization _ def , secHelper _ def ,
propCommandList _ def , extractPropCommand _ def , inputList _ def ,
getOmniCommand _ def ,
MAP, extractInput _ def , satList _ CONS , satList _ nil , GSYM satList _ conj] THEN  

PROVE_TAC[Controls , Modus_Ponens])

val _ = save _ thm ("PlatoonLeader _ PLAN _ PB _ trap _ lemma",
PlatoonLeader _ PLAN _ PB _ trap _ lemma)

val temp3 = snd(dest _ imp(concl thPlanPBTrap))

val PlatoonLeader _ PLAN _ PB _ trap _ justified _ lemma =
TAC PROOF(
[],  

Term`(~(omniCommand:omniCommand) = ssmPlanPBComplete)) ==>
((s:s1State) = PLAN_PB) ==>
^temp3`),  

DISCH_TAC THEN  

DISCH_TAC THEN  

PROVE_TAC[PlatoonLeader _ PLAN _ PB _ trap _ lemma , TR_trap_cmd_rule])

val _ = save _ thm ("PlatoonLeader _ PLAN _ PB _ trap _ justified _ lemma",
PlatoonLeader _ PLAN _ PB _ trap _ justified _ lemma)

val PlatoonLeader _ PLAN _ PB _ trap _ justified _ thm =
REWRITE_RULE[inputList _ def , extractInput _ def , MAP, propCommandList _ def ,
extractPropCommand _ def ,
PlatoonLeader _ PLAN _ PB _ trap _ lemma]
PlatoonLeader _ PLAN _ PB _ trap _ justified _ lemma

```

```

val _ = save_thm ("PlatoonLeader_PLAN_PB_trap_justified_thm",
                  PlatoonLeader_PLAN_PB_trap_justified_thm)

(* _____ *)
(* Theorem: PlatoonLeader is not discarded on omniCommand and *) *)
(* Omni is not discarded on plCommand *) *)
(* _____ *)
*)

val thgen =
GENL
[``(elementTest :('command option , 'principal , 'd, 'e) Form -> bool)``,
 ``(``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)``,
 ``(``s :'state)``,
 ``(``outs :'output list)``,
 ``(``NS :'state -> 'command option list trType -> 'state)``,
 ``(``Out :'state -> 'command option list trType -> 'output)``,
 ``(``M :('command option , 'b, 'principal , 'd, 'e) Kripke)``,
 ``(``Oi :'d po)``, ``(``Os :'e po)``]
TR_discard_cmd_rule

val thPlanPBdiscard =
ISPECL
[``inputOK:((slCommand command)option , stateRole , 'd,'e)Form -> bool``,
 ``secAuthorization :((slCommand command)option , stateRole , 'd,'e)Form list ->
   ((slCommand command)option , stateRole , 'd,'e)Form list``,
 ``secContext: (slState) ->
   ((slCommand command)option , stateRole , 'd,'e)Form list ->
   ((slCommand command)option , stateRole , 'd,'e)Form list``,
 ``[(Name Omni) says (prop (SOME (SLc (PL plCommand))))
    :((slCommand command)option , stateRole , 'd,'e)Form;
   (Name PlatoonLeader) says (prop (SOME (SLc (OMNI omniCommand))))
    :((slCommand command)option , stateRole , 'd,'e)Form]``,
 ``ins :((slCommand command)option , stateRole , 'd,'e)Form list list``,
 ``(``PLAN_PB)``,
 ``outs:slOutput output list trType list``] thgen

val th3d = LIST_BETA_CONV (Term `(\p q. p /\ q) F ((\p q. p /\ q) T ((\p q. p /\ q) T T))`)
val th3d2 = LIST_BETA_CONV (Term `(\p q. p /\ q) T T`)

val PlatoonLeader_Omni_notDiscard_slCommand_thm =
REWRITE_RULE[inputList_def , extractInput_def ,
            authenticationTest_def , MAP, inputOK_def , FOLDR,
            th3d , th3d2] thPlanPBdiscard

val _ = save_thm ("PlatoonLeader_Omni_notDiscard_slCommand_thm",
                  PlatoonLeader_Omni_notDiscard_slCommand_thm)

(* ===== Just playing around with this =====
   ===== OK, done fooling around ===== *)

val _ = export_theory();

end

```


D.4 Horizontal Slice

D.4.1 ssmPlanPB

D.4.1.1 PlanPBType Theory: Type Definitions

D.4.1.2 PlanPBDef Theory: Authentication & Authorization Definitions

D.4.1.3 ssmPlanPB Theory: Theorems

D.4.2 ssmMoveToORP

D.4.2.1 MoveToORPType Theory: Type Definitions

D.4.2.2 MoveToORPDef Theory: Authentication & Authorization Definitions

D.4.2.3 ssmMoveToORP Theory: Theorems

D.4.3 ssmConductORP

D.4.3.1 ConductORPType Theory: Type Definitions

D.4.3.2 ConductORPDef Theory: Authentication & Authorization Definitions

D.4.3.3 ssmConductORP Theory: Theorems

D.4.4 ssmMoveToPB

D.4.4.1 MoveToPBType Theory: Type Definitions

D.4.4.2 MoveToPBDef Theory: Authentication & Authorization Definitions

D.4.4.3 ssmMoveToPB Theory: Theorems

D.4.5 ssmConductPB

Appendix E

Map of The File Folder Structure

References

- [1] United States Army Ranger School, ATTN: ATSH-RB, 10850 Schneider Rd, Bldg 5024, Ft Benning, GA 31905. *Ranger handbook*, April 2017.
- [2] *Introduction to Programming Languages/Algebraic Data Types*. used under the Creative Commons Attribution-ShareAlike 3.0 License, August 2017.
- [3] 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.
- [4] Shiu-Kai Chin and Susan Older. *Certified Security by Design Using Higher Order Logic*, volume Version 1.5. Department of Electrical Engineering and Computer Science Syracuse University, Syracuse University, Syracuse, New York 13244, November 2017.
- [5] Director Peter Berg and Writer Matthew Michael Carnahan. The kingdom. Motion Picture Movie, September 2007.
- [6] ISO/IEC/IEEE 15288 international standard - systems and software engineering – system life cycle processes. Technical Report ISO/IEC/IEEE 15288:2015(E), International Organization for Standardization (ISO) & International Electrotechnical Commission (IEC) & Institute of Electrical and Electronics Engineers (IEEE), 3 Park Avenue, New York, NY 10016-5997, USA, May 2015.
- [7] Ron Ross, Michael McEvilley, and Janet Carrier Oren. Systems security engineering considerations for a multidisciplinary approach in the engineering of trustworthy secure systems. Special Publication 800-160, National Institute of Standards and Technology (NIST), 100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930, May 2016.
- [8] JPMC's SWIFT protocols Glenn Benson, Shiu-Kai Chin, Sean Croston, Karthick Jayaraman, and Susan Older. Banking on interoperability: Secure, interoperable credential management. *Computer Networks*, 67:235–251, 2014.
- [9] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems, April 1975.
- [10] Availability, January 2018.

- [11] Michael Collins. Formal methods: 18-849b dependable embedded systems. Technical report, Carnegie Mellon University, Spring 1998.
- [12] Edmund M. Clarke and et al Jeannette M. Wing. Formal methods: State of the art and future directions, December 1996.
- [13] A. Prasad Sistla, V.N.Venkatakrishnan, Michelle Zhou, and Hilary Branske. Cmv: Automatic verification of complete mediation for java virtual machines. In *ASIACCS '08 Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 100–111, March 2008.
- [14] TOKUOKA Hiroki, MIYAZAKI Yoshiaki, and HASHIMOTO Yuusuke. C-language verifiction tool using formal methods "varvel".
- [15] ZHU Xin-feng, WANG Jian-dong, Li Bin, ZHU Jun-wu, and Wu Jun. Methods to tackle state explosion problem in model checking. In *2009 Third International Symposium on Intelligent Information Technology Application*, number ISBN: New-2005 _ POD _ 978-0-7695-3859-4, pages 329–331. IEEE, December 2009.
- [16] Formal methods. *Wikipedia*, February 2018.
- [17] David Turner and Yannick Welsch. Reliable by design: Applying formal methods to distributed systems.
- [18] Jagatheesan Kunasaikaran, Azlan Iqbal, Jalan Dua, and Chan Sow Lin. A brief overview of functional programming languages, 2016.
- [19] Mathew Fluet. Typeconstructor, January 2015.
- [20] Bsd licenses, May 2018.
- [21] Hol interactive theorem prover.
- [22] Hol (proof assistant), April 2016.
- [23] ISO. Systems and software engineering—life cycle management—part 1: guide for life cycle management. Technical Report ISO/IEC TR 24748-1:2010(E), International Organization for Standardization (ISO), September 2010.
- [24] Concepts of operations, May 2018.
- [25] Lt Gen Charles F. Wald or Lt Col Sanders. Air force policy directive 10-28: Conops development. Operations 10-28, Secretary of The Air Force, January 2002.
- [26] NIST. Nist mission, vision, core competencies, and core values.
- [27] 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.
- [28] Saul kripke, May 2018.
- [29] Charles Q. Choi. Alien planets with extra suns can have strange orbits. *Space.com*.