

Shiu-Kai

Systems-Theoretic Operational Risk Management (STORM) Applied to United States Army Ranger Patrol Base Operations:

Applicability to A Non-automated, Human-Centered System

Lori Pickering

A thesis presented for the degree of
Master of Science in Computer Science

College of Engineering & Computer Science
Syracuse University
U.S.A.
Fall 2018
©Copyright 2018 – Lori D. Pickering

Abstract

Disclaimer

The views 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) or the College of Engineering and Computer Science at Syracuse University.

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.

A significant contributor to this research is U.S. Army Captain Jesse Nathaniel Hall who is also a graduate student at Syracuse University in the School of Information Science (iSchool). The diagram in the front cover is his work. His translation of the patrol base operations from the U.S. Ranger Handbook is a significant contribution and it is noted where appropriate in this master thesis.

Another contributor to this research is YiHong Guo, an undergraduate student at Syracuse University in the College of Engineering and Computer Science. YiHong's contribution includes the original documentation¹ of this work in LaTeX.

¹YiHong's work does not appear in this master thesis. However, he was very helpful and deserves some credit.

Table Of Contents

Abstract	i
Table Of Contents	iii
List of Figures	ix
List of Tables	xi
List of Acronyms	xii
1 Introduction	1
1.1 In Context of Systems Security Engineering	2
1.2 System-Theoretic Operational Risk Management (STORM)	3
1.3 Extending The Range of Applicability	3
1.4 This Master Thesis	1
2 Background And Introduction to STORM	2
2.1 Systems Engineering	2
2.2 Systems Security Engineering	4
2.2.1 Systems Security Engineering Framework	6
2.2.1.1 Conforming to The SSE Framework	8
2.3 STORM	8
2.3.1 STAMP	10
2.3.2 STPA/STPA-Sec	13
2.4 STPA/STPA-Sec Overview: Four Steps	14
2.5 CSBD	17
2.5.1 Confidentiality, Integrity, and Availability (CIA)	18
2.5.2 Complete Mediation	18
2.5.3 Secure State Machines as Transition Systems	19
2.6 Patrol Base Operations	19
3 STPA/STPA-Sec on Patrol Base Operations	21
3.1 Step 1: Define The Purpose of The Analysis	21
3.1.1 Organization FMA	22
3.1.2 Patrol Base Operations FMA	23
3.1.3 Assumptions about The System	24
3.1.4 System Entities	25
3.1.5 Accidents/Losses	26
3.1.6 System-level Hazards/Vulnerabilities And Constraints	27

3.2	Step 2: Model The Control Structure	28
3.2.1	Phases of The Patrol Base Operations	28
3.2.2	Controllers And Process Models	29
3.2.3	Control Actions	31
3.2.4	Functional Control Structure	32
3.3	Step 3: Identify Unsafe Control Actions (UCAs)	34
3.3.1	Unsafe Control Actions (UCAs): Thomas Model	36
3.4	Step 4: Identify Loss Scenarios	39
3.4.1	Scenarios	39
3.5	Discussion And Conclusions	50
4	Certified Security by Design (CSBD) & Access-Control Logic (ACL)	52
4.1	Certified Security by Design (CSBD)	52
4.2	Access-Control Logic (ACL)	53
4.2.1	ACL: A Command and Control (C2) Calculus	53
4.2.2	Principals	54
4.2.3	Propositional Variables, Requests, Authority, Jurisdiction, and Delegation	54
4.2.4	Well-formed Formulas	56
4.2.5	Kripke Structures & Semantics	56
4.2.5.1	Kripke Structures	57
4.2.5.2	Kripke Semantics	57
4.2.5.3	Satisfies	58
4.2.5.4	Soundness	58
4.2.6	Inference Rules	59
4.2.7	Complete mediation	60
4.3	ACL in HOL	61
4.3.1	Principals	61
4.3.2	Well-Formed Formulas	62
4.3.3	Kripke structures	64
4.3.4	ACL Formulas	65
4.3.5	Kripke Semantics: The Evaluation Function	66
4.3.6	Satisfies And Soundness	66
5	Patrol Base Operations	68
5.1	Patrol Base Operations	69
5.2	Overview of The Model	69
5.3	Hierarchy of Secure State Machines	71
5.3.1	Diagrammatic Description in Visio	71
5.3.2	Descriptions of Individual Modules	74
5.3.3	OMNI-Level	76
5.3.4	Escape	77
5.3.5	Top Level	78
5.3.6	Horizontal Slice	80
5.3.6.1	ssmPlanPB	80
5.3.6.2	ssmMoveToORP	82
5.3.6.3	ssmConductORP	83
5.3.6.4	ssmMoveToPB	84

5.3.6.5	ssmConductPB	85
5.3.7	Vertical Slice	86
5.3.7.1	ssmSecureHalt	87
5.3.7.2	ssmORPRecon	88
5.3.7.3	ssmMoveToORP4L	89
5.3.7.4	ssmFormRT	90
6	Secure State Machine Model	91
6.1	State Machines	91
6.1.1	States	92
6.1.2	Transition Commands	92
6.1.3	Next-state Function	93
6.1.4	Next-output Function	94
6.1.5	Configuration	95
6.1.6	TR Relations	95
6.2	Secure State Machines	97
6.2.1	State Machine Versus Secure State Machine	97
6.2.2	Monitors	98
6.2.3	Transition Types	98
6.2.4	Commands	98
6.2.5	Principals And Requests	99
6.2.6	Authentication	100
6.2.7	Authorization	100
6.2.8	Next-state And Next-output Functions	101
6.2.9	Configurations	102
6.2.10	TR Relations	102
6.2.11	Configuration Interpretation	103
6.3	Secure State Machines in HOL	104
6.3.1	Parameterizable Secure State Machine	104
6.3.2	Input Stream	105
6.3.3	Commands	106
6.3.3.1	Transition Types	110
6.3.4	Authentication	111
6.3.5	Authorization	111
6.3.6	Next-state And Next-output Functions	112
6.3.7	Configurations	112
6.3.8	Configuration Interpretation	114
6.3.9	TR Rules	115
7	Patrol Base Operations as Secure State Machines	120
7.1	ssmPB: A Typical Example from the Hierarchy	121
7.1.1	Principals	122
7.1.2	States	122
7.1.3	Outputs	123
7.1.4	Commands	124
7.1.5	Next-State Function	125
7.1.6	Next-Output Function	126
7.1.7	Authentication	126

7.1.8	Authorization	129
7.1.9	Proved Theorems	132
7.2	ssmConductORP: Multiple Principals	144
7.2.1	Principals	144
7.2.2	States	145
7.2.3	Outputs	145
7.2.4	Commands	145
7.2.5	Next-State Function	146
7.2.6	Next-Output Function	148
7.2.7	Authentication	148
7.2.8	Authorization	149
7.2.9	Proved Theorems	151
7.3	ssmPlanPB: Non-sequential Transitions	162
7.3.1	Principals	163
7.3.2	States	164
7.3.3	Output	164
7.3.4	Commands	165
7.3.5	Next-State Function	166
7.3.6	Next-Output Function	168
7.3.7	Authentication	169
7.3.8	Authorization	169
7.3.9	Proved Theorems	171
8	Discussion	185
8.1	STPA Analysis	185
8.2	CSBD Analysis	185
8.2.1	Authentication	185
8.2.2	Roles	186
8.3	Soldier, Squad, and Platoon Theories	186
8.4	Non-sequential Variations	187
8.5	Concluding Comments	190
9	Future Work	192
9.1	Patrol Base Operations	192
9.2	Automation	193
9.2.1	STORM	193
A	Appendices	195
A	Background	196
A.1	Formal Methods	196
A.2	Functional Programming	197
A.3	Algebraic Data Types in ML	198
A.4	Higher Order Logic (HOL) Interactive Theorem Prover	200
A.5	Other Interactive Theorem Provers	201
A.6	How to Compile The Included Files	201
B	Access Control Logic Theories: Pretty-Printed Theories	202

C Parametrizable Secure State Machine & Patrol Base Operations: Pretty-Printed Theories	225
D Parametrizable Secure State Machine Theories: HOL Script Files	340
D.1 ssm	340
D.2 satList	345
E Secure State Machine Theories Applied to Patrol Base Operations: HOL Script Files	347
E.1 OMNILevel	347
E.2 escapeLevel	348
E.3 TopLevel	348
E.3.1 PBTypeIntegrated Theory: Type Definitions	348
E.3.2 PBIntegratedDef Theory: Authentication & Authorization Definitions	350
E.3.3 ssmPlanPBIntegrated Theory: Theorems	352
E.4 Horizontal Slice	356
E.4.1 ssmPlanPB	356
E.4.1.1 PlanPBType Theory: Type Definitions	356
E.4.1.2 PlanPBDef Theory: Authentication & Authorization Definitions	357
E.4.1.3 ssmPlanPB Theory: Theorems	359
E.4.2 ssmMoveToORP	366
E.4.2.1 MoveToORPType Theory: Type Definitions	366
E.4.2.2 MoveToORPDef Theory: Authentication & Authorization Definitions	366
E.4.2.3 ssmMoveToORP Theory: Theorems	366
E.4.3 ssmConductORP	366
E.4.3.1 ConductORPType Theory: Type Definitions	366
E.4.3.2 ConductORPDef Theory: Authentication & Authorization Definitions	367
E.4.3.3 ssmConductORP Theory: Theorems	369
E.4.4 ssmMoveToPB	375
E.4.4.1 MoveToPBType Theory: Type Definitions	375
E.4.4.2 MoveToPBDef Theory: Authentication & Authorization Definitions	375
E.4.4.3 ssmMoveToPB Theory: Theorems	375
E.4.5 ssmConductPB	375
E.4.5.1 ConductPBType Theory: Type Definitions	375
E.4.5.2 ConductPBDef Theory: Authentication & Authorization Definitions	375
E.4.5.3 ssmConductPB Theory: Theorems	375
E.5 Vertical Slice	375
E.5.1 ssmSecureHalt	375
E.5.1.1 SecureHaltType Theory: Type Definitions	375
E.5.1.2 SecureHaltDef Theory: Authentication & Authorization Definitions	375
E.5.1.3 ssmSecureHalt Theory: Theorems	375

E.5.2	ssmORPRecon	375
E.5.2.1	ORPReconType Theory: Type Definitions	375
E.5.2.2	ORPReconDef Theory: Authentication & Authorization Definitions	375
E.5.2.3	ssmORPRecon Theory: Theorems	375
E.5.3	ssmMoveToORP4L	375
E.5.3.1	MoveToORP4LType Theory: Type Definitions	375
E.5.3.2	MoveToORP4LDef Theory: Authentication & Authorization Definitions	375
E.5.3.3	ssmMoveToORP4L Theory: Theorems	375
E.5.4	ssmFormRT	375
E.5.4.1	FormRTType Theory: Type Definitions	375
E.5.4.2	FormRTDef Theory: Authentication & Authorization Definitions	375
E.5.4.3	ssmFormRT Theory: Theorems	375
F	Map of The File Folder Structure	376
References		378

List of Figures

2.1	Systems security engineering Framework. (Image from National Institute of Standards and Technology (NIST) Special Publication 800-160 Vol 1: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.[1])	6
2.2	Relationship between System-Theoretic Process Analysis for Security (STPA-Sec), System-Theoretic Process Analysis (STPA), and System-Theoretic Accident Model and Processes (STAMP). (Image from Dr. William Young and Reed Prada 2017 STAMP Conference presentation in Boston, MA on March 27, 2017 [2])	10
2.3	Example of a control model. (Image captured from the STPA Handbook [3].)	12
2.4	Controller with process model. (Image captured from the <i>Engineering a Safer World</i> Handbook [4].)	13
2.5	Four step process of STPA. (Image captured from the STPA Handbook [3].))	15
2.6	Control structure with potential system-level vulnerabilities/hazards and security considerations. (Image captured from <i>Systems thinking for safety and security</i> [5].)	16
3.1	Control structure for patrol base operations.	33
4.1	Kripke semantics. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [6]	58
4.2	The access-control logic (ACL) inference rules. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [6]	59
4.3	The <i>Controls</i> inference rule. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [6]	60
4.4	The <i>Reps</i> inference rule. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [6]	60
4.5	The <i>Derived Speaks For</i> inference rule. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [6]	61
4.6	The HOL implementation of principle (Princ). Image from <i>Certified Security by Design Using Higher Order Logic</i> [7]	62
4.7	The definition for Form in Higher Order Logic Theorem Prover (HOL). <i>Certified Security by Design Using Higher Order Logic</i> [7]	63
4.8	The HOL implementation of a Kripke structure (Princ). Image from <i>Certified Security by Design Using Higher Order Logic</i> [7]	64
4.9	The ACL formulas in HOL. Image taken from <i>Access Control, Security, and Trust: A Logical Approach</i> [6]	65

4.10	The HOL implementation of the Kripke semantics (evaluation function). Image from <i>Certified Security by Design Using Higher Order Logic</i> [7]	67
4.11	The HOL implementation of the "satisfies" property. Image from <i>Certified Security by Design Using Higher Order Logic</i> [7]	67
5.1	A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel Hall.	70
5.2	Diagrammatic description of patrol base operations as a modularized hierarchy of secure state machines. (Generated by Jesse Nathaniel Hall.)	71
5.3	Escape level diagram.	77
5.4	Top level diagram.	78
5.5	Horizontal slice: PlanPB diagram.	80
5.6	Horizontal slice: MoveToORP diagram.	82
5.7	Horizontal slice: ConductORP diagram.	83
5.8	Horizontal slice: MoveToPB diagram.	84
5.9	Horizontal slice: ConductPB diagram.	85
5.10	Vertical slice: SecureHalt diagram.	87
5.11	Vertical slice: ORPRecon diagram.	88
5.12	Vertical slice: MoveToORP4L diagram.	89
5.13	Vertical slice: FormRT diagram.	90
6.1	A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel Hall.	92
6.2	State machine versus secure state machine with a monitor. Image taken from <i>Certified Security by Design Using Higher Order Logic</i> [7].	97
6.3	Top level diagram.	99
6.4	Transition rules in HOL. Image taken from Certified Security by Design Using Higher Order Logic [7]	115
7.1	Top level diagram.	121
7.2	Horizontal slice: ConductORP diagram.	144
7.3	Horizontal slice: PlanPB diagram.	162
8.1	An abbreviated concept for a soldier module.	187
8.2	Horizontal slice: PlanPB diagram.	188
8.3	Alternative path through the three non-sequential states.	188
8.4	Permutation paths through the three non-sequential states..	189
A.1	An implementation of the boolean datatype in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [8]	198
A.2	An implementation of a tree datatype in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [8]	199
A.3	An implementation of the weekday datatype in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [8]	199
A.4	An implementation of the function <i>is_weekend</i> in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [8]	199
A.5	An implementation of a parametrized tree datatype in ML. Image from <i>Introduction to Programming Languages/Algebraic Data Types</i> [8]	200

List of Tables

3.1	Organization goals.	22
3.2	Patrol Base Operations Functional Mission Analysis from the U.S. Ranger Handbook [9].	23
3.3	System accidents/losses.	26
3.4	System-level hazards/vulnerabilities and constraints.	27
3.5	Scenarios for UCA E1A3.	28
3.6	Controllers and process model.	29
3.7	Unsafe control actions unsafe control actionss (UCAs) for control action "PL says planComplete."	36
3.8	Unsafe control actions UCAs for control action "exec(planComplete)."	37
3.9	Unsafe control actions UCAs for control action "discard(anyCommand)."	38
3.10	Scenarios for UCA P1B1.	40
3.11	Scenarios for UCA P1B1/P1C1.	41
3.12	Scenarios for UCA P1A1.	42
3.13	Scenarios for UCA P1A2.	43
3.14	Scenarios for UCA P1A3.	44
3.15	Scenarios for UCA P1A4.	44
3.16	Scenarios for UCAs E1B1 and E1C1.	45
3.17	Scenarios for UCA E1A1.	46
3.18	Scenarios for UCA E1A2.	47
3.19	Scenarios for UCA E1A3.	48
3.20	Scenarios for UCA E1A4.	48
3.21	Scenarios for UCA for discard on all commands (D1A1 through D1A4).	49
5.1	Mission Activity Specification for Patrol Base Operations. Adapted from the U.S. Army Ranger Handbook 2017 [9].	69

List of Acronyms

ACL access-control logic.

ADT algebraic data type.

CAST Causal Analysis based on Systems Theory.

CIA confidentiality, integrity, and availability.

CSBD Certified Security by Design.

HOL Higher Order Logic Theorem Prover.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

ISO International Organization for Standardization.

MIT Massachusetts Institute of Technology.

NIST National Institute of Standards and Technology.

NSA National Security Agency.

ORP Objective Rally Point.

SE Systems Engineering.

SSE Systems Security Engineering.

SSM secure state machine.

STAMP System-Theoretic Accident Model and Processes.

STORM System-Theoretic Operational Risk Management.

STPA System-Theoretic Process Analysis.

STPA-Sec System-Theoretic Process Analysis for Security.

TR transition relation.

UCA unsafe control actions.

WFF well-formed formula.

Chapter 1

Introduction

Imagine this scene from the movie titled "The Kingdom" [10]. 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 news stories reporting real terrorist attacks on civilians 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 security in computer driven cars, bank accounts, medical records, pace makers, etc. Security 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 is vulnerable to compromise or attack, security is the foremost consideration.

This chapter introduces the reader to a methodology for designing secure systems. It briefly touches upon how this methodology fits in with current standards in systems engineering and systems security engineering.

1.1 In Context of Systems Security Engineering

This master thesis discusses the System-Theoretic Operational Risk Management (STORM) methodology for designing secure systems. STORM applies state-of-the-field practices to design systems that are consistent with ISO/ IEC/IEEE standards and the NIST Systems Security Engineering Framework. These standards ensure that security is the forefront consideration of all security-sensitive systems.

The published standards on Systems Engineering (SE) are ISO/IEC/IEEE¹ 24748-5 and [11] 15288 [12]. They describe 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 a critical component of the systems engineering process.

¹ISO = International Organization for Standardization (ISO); IEC = International Electrotechnical Commission (IEC); and IEEE = Institute of Electrical and Electronics Engineers (IEEE)

Systems Security Engineering (SSE) is a subspecialty of systems engineering. The Systems Security Engineering Framework is published by the National Institute of Standards and Technology (NIST) (Special Publication 800-160 (vol 1 and vol 2) [1]). It provides guidelines for designing trustworthy systems that engineer security into the system from the start.

1.2 System-Theoretic Operational Risk Management (STORM)

STORM is comprised of two components that each focus on some aspect of the Systems Security Engineering Framework. The first component defines the security problem and finding solutions. It derives from a new paradigm in accident analysis called System-Theoretic Accident Model and Processes (STAMP). This paradigm forms the basis for a hazard analysis methodology called System-Theoretic Process Analysis (STPA). On top of STPA is System-Theoretic Process Analysis for Security (STPA-Sec), which adds a security perspective to the hazard analysis.

The second STORM component demonstrates trustworthiness. Certified Security by Design (CSBD) defines security as confidentiality, integrity, and availability (CIA). It uses an access-control logic and computer-aided theorem proving to reason about, verify, and document security properties.

1.3 Extending The Range of Applicability

Until now, STORM has been applied to automated systems. This master thesis extends the range of STORM applicability by demonstrating it on a non-automated,

human-centered system. As an example of such a system, STORM is applied to the patrol base operations. These operations are described in the U.S. Army Ranger Handbook [9].

1.4 This Master Thesis

The main body of this thesis is divided into two parts: STPA and CSBD. The first part is comprised of chapters 2 and 3. The second part is comprised of chapters 4 through 7.

Chapter 2 describes STORM and STPA/STPA-Sec. The next chapter applies the STPA/STPA-Sec analysis to the patrol base operations.

Chapter 4 discusses CSBD, the access-control logic (ACL) and its implementation in the Higher Order Logic Theorem Prover (HOL) Interactive Theorem Prover. Chapter 5 describes a modularized, hierarchical model of the patrol base operations as secure state machines. Chapter 6 describes secure state machines and their HOL implementation. Chapter 7 describes the HOL implementation of several patrol base operations secure state machines (described in chapter 5). Chapter 8 discusses observations and findings of the overall project. Chapter 9 discusses future work and implications.

Appendix A provides supplemental information on formal methods and functional programming. Appendices B through E contain pretty-printed HOL-generated code and script code for all theories and modules used in this master thesis. Appendix F provides a map of the folder structure for the files included with this master thesis as well as a description of how to compile them.

The next chapter brings the reader into the realm of systems security engineering by STORM.

Chapter 2

Background And Introduction to STORM

This chapter provides an introduction to System-Theoretic Operational Risk Management (STORM). STORM falls within the field of Systems Engineering (SE) and the sub-discipline of Systems Security Engineering (SSE). To familiarize (or refresh) the reader, this chapter begins with some background on SE and SSE. The SE concept of system life-cycle phases and the SSE Framework are introduced. Following this, STORM and its components (STPA-Sec and CSBD) are introduced. The chapter concludes with a discussion of the patrol base operations as a sample systems to which STORM is applied.

2.1 Systems Engineering

"Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting

requirements, then proceeding with design synthesis and system validation while considering the complete problem..."

International Council on Systems Engineering

A system is a set of interacting and interdependent components that act as a whole, through various mechanisms, to perform some function. Systems engineering is a multidisciplinary approach to solving problems related to large and complex man-made systems. It focuses on stakeholder needs and assets while minimizing asset losses. It focuses on building the right product. It avoids building the wrong product or building the right product incorrectly.

According to Holstein and Bode's, who summarized systems engineering for the Encyclopedia Britannica, systems engineering probably developed from the overlapping of engineering concepts from different fields. They note that, for example, both chemical and mechanical engineering focus on heat transfer, and cybernetics and computer theory are both sub-disciplines of electrical and electronics engineering. But, at its core is the scientific methodology and the use of mathematical modeling. Holstein and Bode link the more recent emergence of systems engineering to the communications industry (telephony, in particular) and Britain's post-World War II operations research.

The emergence of systems engineering and its growing popularity is linked to the growing complexity of systems. Systems are composed more and more of numerous interacting parts. These interacting parts often exhibit emergent properties. These are properties that arise from the interactions of many components. These properties are not attributable to any one component and thus require a systems-thinking perspective. Holstein and Bode site the Nike Ajax U.S. Air Defense System as one of the earlier examples of these system-level emergent properties. Tactical range of the Ajax requires consideration of multiple factors including aerodynamic design, maneuverability provided by the control systems, warhead size and weight, etc. Other factors include

feedback from radar and the autopilot. Achieving tactical range emerges from analysis of the many interacting components.

Systems Engineering is often employed in the communications and computing industries. But, it works in any field that works with large complex systems. The applicability of systems engineering to other fields is promoted, in part, by the increased capacity to manage complex systems that arise from the increased computational power available. Computational power and thus complexity will only grow in the future, and consequently, so will the need for competent systems engineers.

The relevant SE standards are defined in ISO/IEC/IEEE¹ 24748-5 [11] and ISO/IEC/IEEE 25288² [12]. These standards define the concept of system life-cycle phases: concept, development, production, utilization, support, and retirement. The concept and development phases are the focus of STORM because safety and security should be considered during the design phase. However, the support phase is also relevant for re-envisioning and updating the system.

2.2 Systems Security Engineering

"The ultimate objective is to obtain trustworthy secure systems that are fully capable of supporting critical missions and business operations while protecting stakeholder assets, and to do so with a level of assurance that is consistent with the risk tolerance of those stakeholders."

Ron Ross (NIST) (Quoted from NIST SP 800-160 [1])

Systems Security Engineering (SSE) is a sub-discipline of Systems Engineering (SE). It

¹ISO = International Organization for Standardization; IEC = International Electrotechnical Commission; and IEEE = Institute of Electrical and Electronics Engineers

²These standards reference other relevant standards.

applies scientific, mathematical, and related technical concepts and procedures to engineer trustworthy systems that meet stakeholder needs.

Systems Security Engineering probably developed alongside systems engineering. But, the need for security engineering is becoming more and more on the forefront of everyone's minds in today's technology-dependent society. With recent high-profile security breaches such as the Target security leak (\$18.5 million legal settlement [13]) and the disastrous Snowden leaks (cost of leak unknown), companies can no longer afford to take a relaxed posture on security. This need for security is stated no better than by experts in the field

"With the continuing frequency, intensity, and adverse consequences of cyber-attacks, disruptions, hazards, and other threats to federal, state, and local governments, the military, businesses, and the critical infrastructure, the need for trustworthy secure systems has never been more important to the long-term economic and national security interests of the United States.

Engineering-based solutions are essential to managing the growing complexity, dynamically, and interconnectedness of today's systems, as exemplified by cyber-physical systems and systems-of-systems, including the Internet of Things." [1]

NIST Special Publication 800-160 vol 1 [1].

Systems Security Engineering is applied whenever security is a component of systems engineering. As a multi-disciplinary, subspecialty of systems engineering, SSE focuses on stakeholder assets and loss avoidance or mitigation. The Systems Security Engineering Framework defines the context in which STORM applies.

2.2.1 Systems Security Engineering Framework

The authoritative document on SSE is the NIST Special Publication 800-160 Volumes 1 and 2. In particular, volume 1 defines the SSE Framework. This framework defines SSE activities that ensure the stakeholder's definition of security and asset valuation informs the system's security design. A diagram of the framework is shown in figure 2.1.

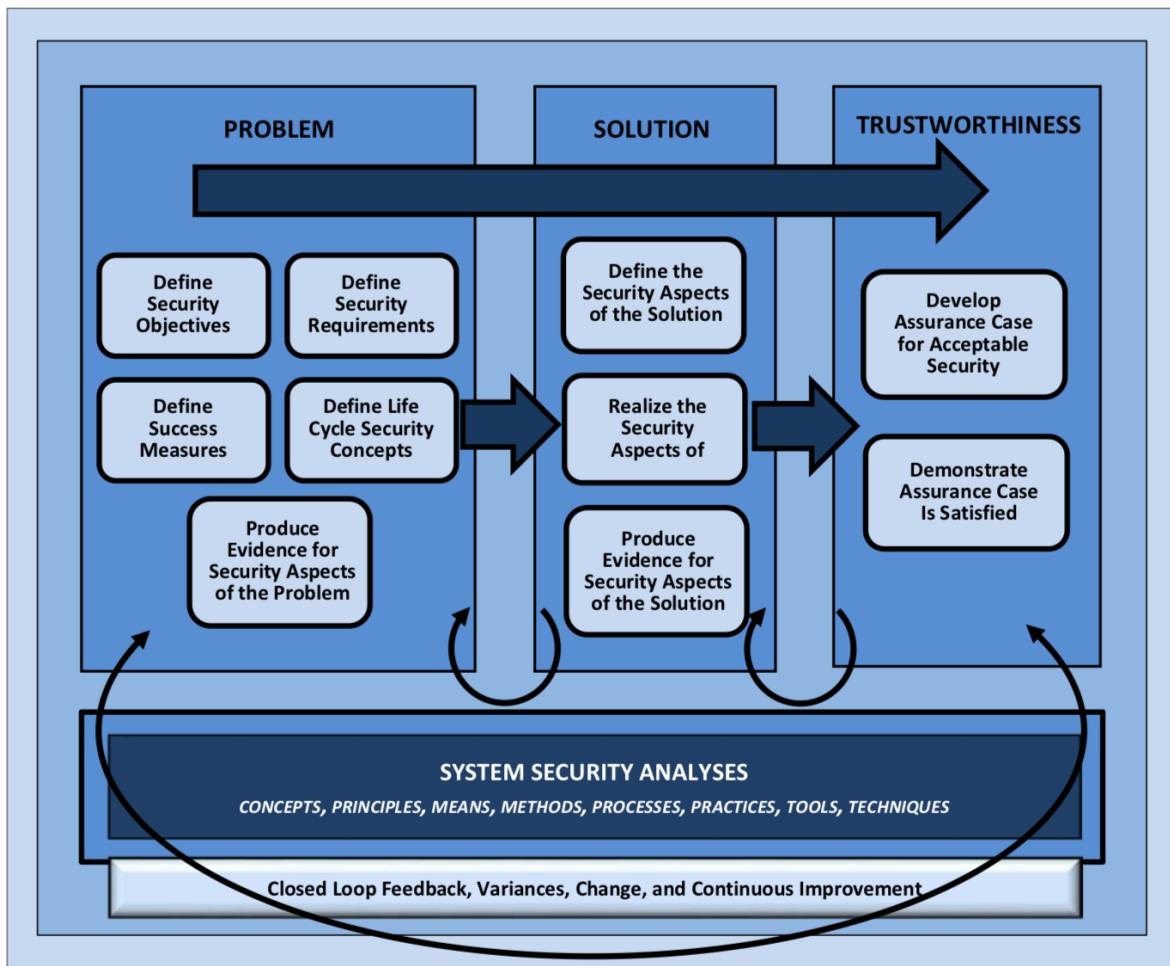


Figure 2.1: Systems security engineering Framework. (Image from NIST Special Publication 800-160 Vol 1: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.[1])

The framework describes the problem, solution, and trustworthiness activity contexts. These three contexts are further refined.

Problem The problem phase focuses on defining security from the stakeholder's needs, purpose, and mission. It is divided into four sub-contexts (activities). The first activity defines the security objective. What does it mean to be "*adequately secure*"? What are the assets and asset losses? Which losses are unacceptable?

The second activity defines measures of success. What level of asset protection is required? What level of assurance or protection is required?

The third activity defines the system life-cycle security concepts. What is the system security context throughout the life-cycle of the system? What processes, methods, and procedures throughout the system's life cycle need security?

The fourth activity defines the security requirements. The stakeholder security requirements are determined from the previous three activities.

Solution With the problem defined, the solution activity transforms the problem into security design requirements. This has three activities. The first activity defines the security aspects of the solution. Six aspects are listed in NIST SP 800-160: protection strategy; security design requirements; security architecture view and viewpoints; security design; security procedural aspects, capabilities, and limitations in the system life-cycle; and how to verify and measure security performance.

The next activity realizes the security aspects. This is the security implementation phase.

The last activity produces evidence for the security aspects of the solution. This can be obtained from a variety of SSE verification methods. This evidence is compared with claims measured against the stakeholder's requirements.

Trustworthiness Once the solution is generated, the trustworthiness context develops assurance cases and demonstrates that they are satisfied. The first activity develops and maintains assurance cases for acceptable security. NIST SP 800-160 defines assurance case as "*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.*" Assurance cases also provide auditable artifacts supporting the system security claims.

The second trustworthiness activity demonstrates that the assurance case is satisfied. A subject-matter expert should evaluate the assurance case to determine if the evidence supports the security claims.

2.2.1.1 Conforming to The SSE Framework

STORMis consistent with the SSE Framework. It is up to the STORM user to verify in detail that *all* nine sub-contexts of the SSE Framework are addressed.

STORM's two components in combination satisfy all three SSE Framework contexts. The problem and solution contexts are covered by STPA/STPA-Sec. The trustworthy context is covered by CSBD. Both STPA/STPA-Sec and CSBD are described in the following sections.

2.3 STORM

System-Theoretic Operational Risk Management (STORM) is an approach to managing risk and engineering trustworthy systems that satisfy stakeholder requirements and conform to industry standards. It applies state-of-the-field analysis techniques to design products that are safe, secure, and integral. It applies formal methods to produce reproducible and auditable verification demonstrating that stakeholder claims are

satisfied.

STORM is a new approach to implementing the Systems Security Engineering Framework. It can also be used in conformance to the ISO/IEC/IEEE 15288 and 24748 standards.

STORM is the result of the collaborative effort of Professor Shiu-Kai Chin of the College of Engineering and Computer Science at Syracuse University, Erich Devendorf, PhD of the U.S. Air Force Research Laboratory, and William Young, PhD of the U.S. Air Force 53rd Electronic Warfare Group.

STPA is the culmination of years of research in safety engineering by Massachusetts Institute of Technology (MIT) professor Nancy Leveson. Whereas STPA focuses on safety engineering, STPA-Sec focuses on security engineering. STPA-Sec is the work of Dr. Young, who presented it as his PhD thesis in 2014 at MIT.

CSBD is derived from the work of Professor Shiu-kai Chin and Professor Susan Older (also of the College of Engineering and Computer Science at Syracuse University). They wrote the text on *Access Control, Security, and Trust: A Logical Approach* [6] from which CSBD is derived. Professor Shiu-Kai Chin teaches components of CSBD at Syracuse University in the College of Engineering and Computer Science.

All elements of STORM are currently employed by the U.S. Air Force. STPA-Sec is part of the U.S. Air Force doctrine. Components of STORM are taught at Syracuse University, in particular CSBD. CSBD has also been used in the design of JP Morgan Company's SWIFT protocols [14] and the F-16 Viper payload controller and secure memory loader verifier (not published). As this master thesis is being written, STORM is being presented to the National Security Agency (NSA) to strengthen their mission assurance objectives.

2.3.1 STAMP

System-Theoretic Accident Model and Processes (STORM) is the model that underlies STPA, which underlies STPA-Sec. Figure 2.2 shows the relationship between STPA-Sec, STPA, and STAMP.

STAMP is a way of thinking about how accidents occur. It assumes both a traditional and a system-theoretic view. In the traditional view, accidents are caused by unsafe chains of events. In the system theoretic view, accidents are also caused by dynamic and complex interactions.



Figure 2.2: Relationship between STPA-Sec, STPA, and STAMP. (Image from Dr. William Young and Reed Prada 2017 STAMP Conference presentation in Boston, MA on March 27, 2017 [2])

STAMP is the subject of yearly conferences world wide promoted by the Partnership for Systems Approaches to Safety and Security (PSASS)³. When compared to other accident models, STAMP is more effective.⁴ It also cost less to implement [3].

The need for a new hazard analysis paradigm arises near the turn of the century from the culmination of years of research into safety engineering by MIT professor Nancy

³See <https://psas.scripts.mit.edu/home/other-stamp-meetings/>.

⁴See, for example, Paul Stukus' thesis dissertation on how STAMP outdid other techniques when analyzing a U.S. Coast Guard Buoy Tender Integrated Control System [15]

Leveson. This research reveals a need for new tools in safety analysis. Current tools, at one time effective, are proving ineffective in managing the complexities associated with technologies of the modern era. The current practice of analyzing individual component reliability and chain-of-causality failures does not capture the hazards associated with today's complex systems. These systems require a system-theoretic approach to hazard analysis.

System theory focuses on the system as a whole rather than as a collection of subcomponents. The need for a system theory approach is epitomized in the notion that the whole is greater than the sum of its parts. From the complex interactions of individual components arise emergent properties. These properties can be thought of as a higher order that is not predictable from the behavior of the individual components.

STAMP forms the foundation of the new, modern era systems-theoretic safety analysis techniques. Several approaches to safety engineering are built upon the STAMP foundation. System-Theoretic Process Analysis (STPA) is one these analysis techniques. In addition to STPA (the topic of the next section), several other analysis techniques are built on top of the STAMP foundation. These include Causal Analysis based on Systems Theory (CAST), STPA for security (STPA-Sec), STPA for privacy (STPA-Priv), STPA-SafeSec [16], and SAFE (a refinement focused on hardware and software subsystems [16]).

At STAMP's core is a top-down model that views accidents as dynamic control problems. It focuses on safety constraints, hierarchical control structures, and process models [4].

safety constraints STAMP views constraints, rather than events, as the safety-critical processes. A lack of constraints can lead to a hazardous system state which will lead to an accident in the worst-case scenario.

There are two types of constraints (or controls), passive and active. Passive controls prevent unsafe conditions by their presence. For example, lead shielding surrounding radioactive material minimizes radioactive exposure merely by its presence.

Active controls, on the other hand, are more complicated. They require some form of interaction to prevent unsafe conditions. For example, a Geiger counter signaling a computer that radioactive levels are unsafe minimizes exposure by sounding an alarm which causes personnel to evacuate.

hierarchical control structures Figure 2.3 is an example of a hierarchical control model. The flow of control is from top to bottom; higher-level control structures control lower-level control structures. Downward pointing arrows indicate instructions to lower level controllers or controlled processes. Upward pointing arrows indicate feedback from lower components.

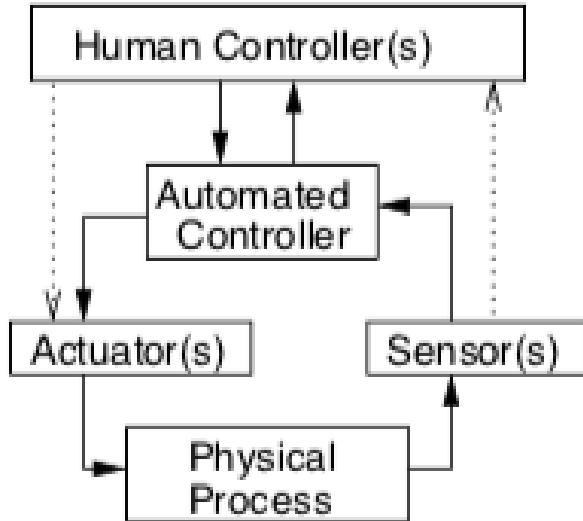


Figure 2.3: Example of a control model. (Image captured from the STPA Handbook [3].)

The cause of accidents is inadequate control at higher levels, which trickle down to lower levels causing hazardous conditions.

STAMP considers four types of inadequate control on constraints: insufficient or missing constraints, wrong constraints, out-of-order or under-timed constraints, and constraints that are imposed for too long.

process models Figure 2.4 shows a controller with a process model. Each controller has its own process model. The process model describes the controller's understanding of how the system works as well as knowledge about the state of the system.

If the controller's process model is somehow flawed or not consistent with the system's state, it could cause a hazardous condition which could cause unacceptable losses in the worst-case scenario.

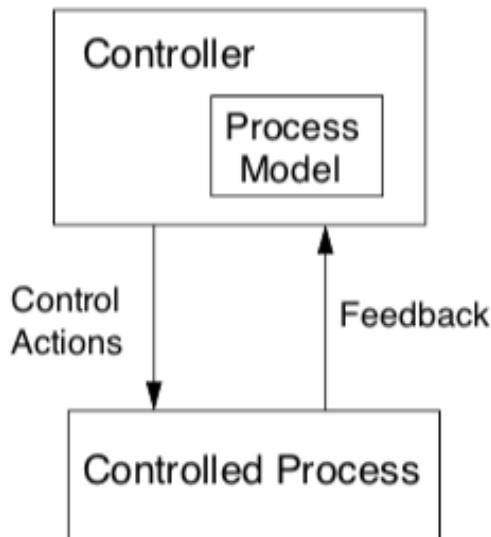


Figure 2.4: Controller with process model. (Image captured from the *Engineering a Safer World* Handbook [4].)

2.3.2 STPA/STPA-Sec

System-Theoretic Process Analysis (STPA) is a systems engineering methodology that helps the engineer identify and mitigate stakeholder losses. It does this using a four step process that is founded on the System-Theoretic Accident Model and Processes

(STAMP). Whereas STAMP is a way of thinking about the causes of accidents, STPA is a method to analyze systems in order to prevent or mitigate losses caused by these accidents.

STPA follows the guidelines described in ISO/IEC/IEEE 15288 for System Life-Cycle Processes. Its focus is on the Technical Processes (glsiso/IEC/IEEE 15288), the concept and development stage of the system life cycle (defined in ISO/IEC/IEEE 24748), and the problem and solution contexts of the Systems Security Engineering Framework (NIST SP 800-160).

STPA, like STAMP, is the work of Professor Nancy Leveson from MIT. STPA-Sec is William Young's (PhD and Col. in USAF) PhD thesis dissertation. It is a modification of STPA that adds on a security component.

In a 2013 paper, Professor Nancy Leveson and Dr. William Young define safety and security as

Safety protecting a system against *unintentional* disruptions.

Security protecting a system against *intentional* disruptions.

The primary difference between STPA and STPA-Sec is that STPA thinks about how *accidents* arise from system-level *hazards* whereas STPA-Sec thinks about how *losses* arise from system-level *vulnerabilities*. Application of the four steps is similar for both STPA and STPA-Sec.

2.4 STPA/STPA-Sec Overview: Four Steps

STPA is a four step process. Figure 2.5 diagrams these four steps. The four steps are briefly described below. They are applied to the patrol base operations in the next

chapter.

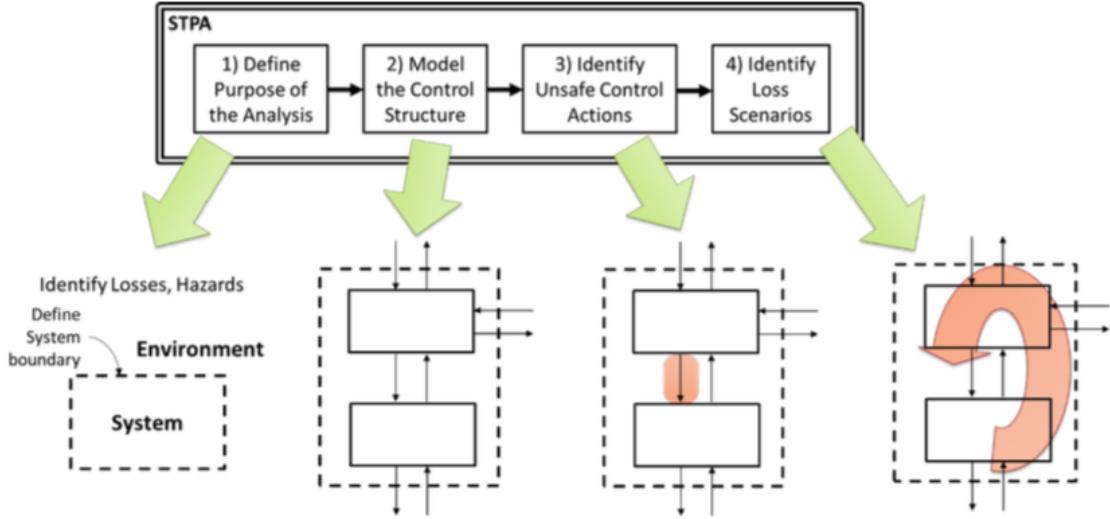


Figure 2.5: Four step process of STPA. (Image captured from the STPA Handbook [3].))

Step 1: Define The Purpose of The Analysis The first step defines the purpose of the analysis and the system to be produced. Unacceptable losses/accidents are defined by the stakeholders. Then, system-level vulnerabilities/hazards are defined and linked to the losses/accidents they would cause in the worse-case scenario. Next, system-level constraints are defined and linked to vulnerabilities/hazards.

Step 2: Model The Control Structure The second step models the system's functional control structure. This describes the flow of control (or information) in the system. Figure 2.3 shows an a generalized control structure.

The control structure shines light on system-level hazards. Figure 2.6 shows a control structure with the potential vulnerabilities/hazards labeled.

(1) and (3) type vulnerabilities/hazards are caused by representation flaws. These include missing or incorrect information or feedback and flaws involving the controller's model of the system. (2) type vulnerabilities/hazards are caused by algorithm flaws (algorithms implement the controller's process model). (4) type vulnerabilities/hazards

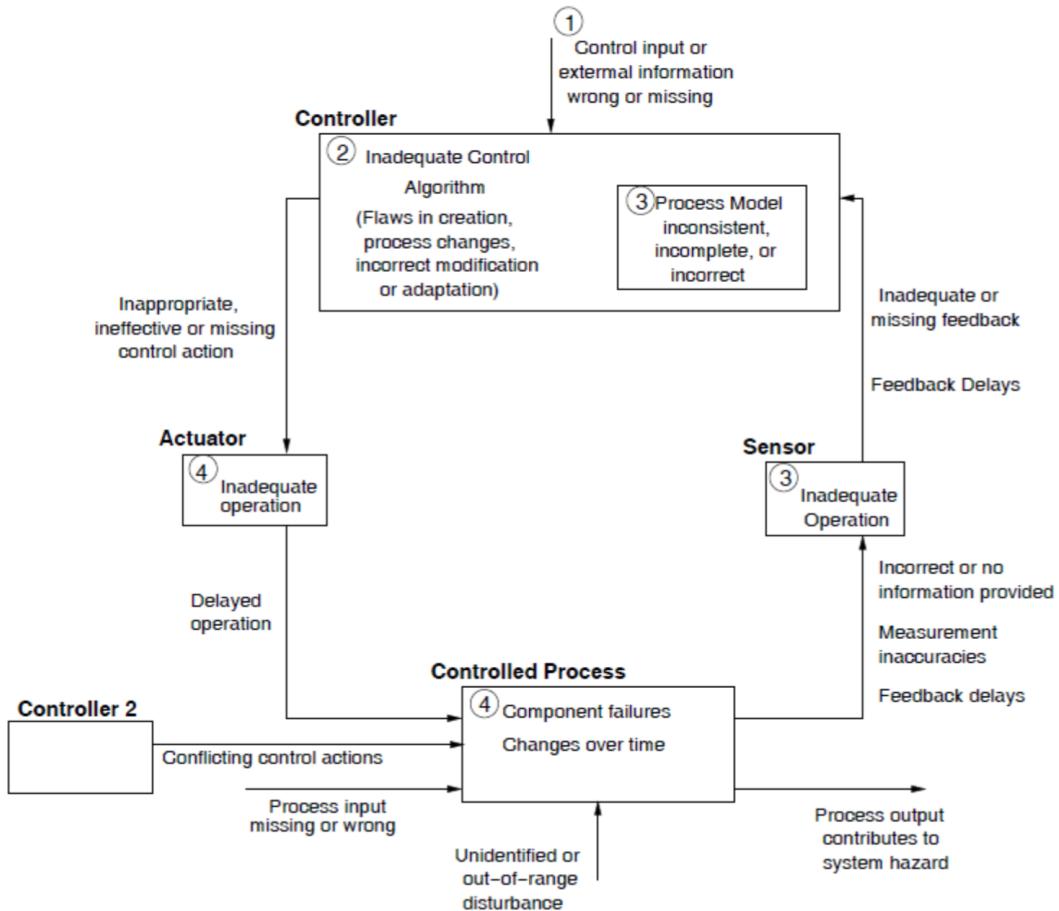


Figure 2.6: Control structure with potential system-level vulnerabilities/hazards and security considerations. (Image captured from *Systems thinking for safety and security* [5].)

are caused by component failures.

The control structure consists of two controllers, one at the top and one near the bottom and to the left. The lower control structure could represent an adversary sending conflicting instructions to the controlled process (bottom of diagram).

Step 3: Identify Unsafe Control Actions The next step identifies unsafe control actions (UCA) for each control action described in step 2. Control actions are instructions (commands, etc.) output by one control structure and input to another, typically from a higher to a lower control structure. There are four types of UCAs.

- Not applying the control action
- Applying the wrong control action
- Applying the control action in the wrong order
- Applying the control action for too long or not long enough.

In this step, each control action is analyzed for all four types of UCAs to check for vulnerabilities/hazards. UCAs are further analyzed in step 4.

Step 4: Identify Loss Scenarios The last step identifies scenarios that could cause the UCAs discovered in step 3. Each scenario describes how a UCA could cause a system-level vulnerability/loss in the worst-case scenario. Then, associated causal factors are analyzed for each scenario. This analysis guides system security engineering decisions.

2.5 CSBD

CSBD can be used to demonstrate trustworthiness of a system within the System Security Engineering Framework. CSBD is the second major component of STORM.

CSBD is a method for formally verifying and documenting the security properties of a system. 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. It uses computer-aided reasoning such as the Higher Order Logic (HOL) Interactive theorem prover to formally verify and document these security properties. .

CSBD has its own chapter (chapter 4) and is briefly described above to explain how it fits into STORM.

2.5.1 Confidentiality, Integrity, and Availability (CIA)

At the core of security are the concepts of confidentiality, integrity, and availability (CIA). Confidentiality refers to limiting access to objects (including information) by ensuring that only the right individuals (etc.⁵) have access. Confidentiality is the realm of authentication and authorization. Authentication verifies an individual's identity. Authorization describes a individual's access rights.

Integrity, on the other hand, refers to the whole of the object (or information, etc.). Integrity controls who or what can modify the object. This also describes an individual's authority over an object.

Availability is a measure of operability or degree to which the system is mission capable [17].

Confidentiality, integrity, and Availability are guarded by the principle of complete mediation.

2.5.2 Complete Mediation

The seminal paper on the principle of complete mediation is "The Protection of Information in Computer Systems" by Saltzer and Schroeder [18]. Complete mediation means:

Every access to every object must be checked for authority... It forces a system-wide view of access control... It implies that a foolproof method of identifying the source of every request must be devised... If a change in authority occurs, such remembered results must be systematically updated.

⁵or any other "entity" that can request access to an object.

In summary, this means 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).

CSBD uses an access-control logic and computer-aided reasoning to demonstrate that a system satisfies the principle of complete mediation.

2.5.3 Secure State Machines as Transition Systems

Secure state machines are used by the CSBD method to enforce constraints on the system. (However, CSBD can also verify complete mediation without secure state machines.)

Secure state machines are transition systems that constrain system behavior and enforce complete mediation. They define allowable system states and allowable transitions among states. Transitions are overseen by a monitor which enforces complete mediation by only allowing transitions if they are requested by an authenticated and authorized principal.

Secure state machines have their own chapter (chapter 6).

2.6 Patrol Base Operations

For the purpose of this master thesis, patrol base operations are an example of a non-automated, human-centered system wherein security is critical for mission success. The operations are described in the U.S. Army Ranger Handbook [9].

The patrol base operations are described throughout this master thesis within the appropriate context. They are first described in the next chapter in the context of the STPA/STPA-Sec analysis. They are described once again in chapter 5 where they are modeled as secure state machines. They are also described in chapter 7 as HOL-implemented secure state machines.

This chapter provides sufficient background for the reader to understand the systems engineering and systems security engineering basis of STORM. It also describes STORM and its components. The next chapter applies STPA/STPA-Sec on the patrol base operations.

Chapter 3

STPA/STPA-Sec on Patrol Base

Operations

This chapter applies the STPA/STPA-Sec analysis to the patrol base operations. The patrol base operations are analyzed at a high level of abstraction and only the actions of the Platoon Leader (PL) and the platoon as a whole are considered. The limited scope of the analysis serves to demonstrate the STPA component of STORM on the system of interest. A complete analysis is beyond the scope of this master thesis.

This chapter describes each of the four steps of the STPA/STPA-Sec analysis. The results are discussed in the text and presented in tables. For a review on the four steps read section 2.3.2 of the previous chapter.

3.1 Step 1: Define The Purpose of The Analysis

This section describes the first step in the STPA/STPA-Sec analysis. It begins with a definition of the stakeholders. It follows with a description of the goals of the most

immediate organizations. It describes the functional mission analysis for the patrol base operations. It then describes assumptions about the system and relevant entities. Next, it describes accidents/losses for the system. Finally, it describes the system-level hazards/vulnerabilities and constraints.

Stakeholders At the highest level, the stakeholders are the U.S. Government and the Citizens of the United States. Below that is the U.S. Army and the U.S. Army Rangers. Depending on the mission, other nations and their citizens may also hold a stake in the activities. The enemy is also a stakeholder with an adversarial agenda.

3.1.1 Organization FMA

The two most immediate organization for the patrol base operations are the U.S. Army Rangers and the U.S. Army. Their goals are stated in table 3.1.

Organization Goals	
U.S. Army Rangers	U.S. Army
"The U.S. Army's mission is to fight and win our Nation's wars by providing prompt, sustained land dominance across the full range of military operations and spectrum of conflict in support of combatant commanders." [19]	"The Rangers' primary mission is to engage the enemy in close combat and direct-fire battles. This mission includes direct action operations, raids, personnel and special equipment recovery, in addition to conventional or special light-infantry operations." [9]

Table 3.1: Organization goals.

3.1.2 Patrol Base Operations FMA

The functional mission analysis for the patrol base operations is shown in table 3.2.

This describes what the patrol base operations are, how they are conducted (broadly), and why they are conducted.

Patrol Base Operations FMA	
What	establish a security perimeter when a squad or platoon halts for an extended period of time
How	planning, reconnaissance, security, control, and common sense
Why	avoid detection: <ul style="list-style-type: none">• 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 common base from which to execute several consecutive or concurrent operations.

Table 3.2: Patrol Base Operations Functional Mission Analysis from the U.S. Ranger Handbook [9].

3.1.3 Assumptions about The System

This analysis relies on the following assumptions about the model of patrol base operations.

- Mission is decided by higher-ups and is provided as an input to the patrol base operations.
- Mission is not changeable by the patrol base itself (exceptions are mission-specific).
- Soldiers are deemed fit for duty before the mission commences. (There may be exceptions.)
- Soldiers are battle ready, but may not be fully mission capable (i.e., soldiers may require additional equipment and preparation that is mission-specific and determined after the mission is received).
- The patrol base operations begin when the platoon (or patrol) leader receives order that there will be a mission.
- The patrol base operations end when the patrol returns to the larger unit and has completed any debriefing required by the mission.
- Other inputs to the system are intelligence from the larger unit (higher-up HQ), intelligence from other units (if applicable), and intelligence gathered during the mission from the mission itself.
- Feedback is received from the platoon regarding carrying out orders and from the operations themselves regarding completion of tasks, etc.
- Adversary also may input information to the platoon leader (by interfering with communications from higher-up HQ), to the platoon soldiers (by interfering with

intra-patrol communications), and to the operations themselves (by disrupting the operations in various forms, including contact).

3.1.4 System Entities

The following entities are defined in the U.S. Ranger Handbook for the patrol base operations.

- HQ: PL, PSG, Medic, FO, RTO, HWSQ
- Squad Leaders
- Fire Team Leaders
- Buddy Team
- Soldiers
- Adversary (enemy)

The platoon leader (Platoon Leader –PL) is the only entity specifically referred to in this limited-scope analysis. The platoon sergeant (PSG) is the assistant to the platoon leader. The patrol base head quarters (HQ) consists of the PL, PSG, Medic, forward observer (FO), radio telephone operator (RTO), and the heavy weapons squad leader (HWSQL). Below the HQ are four squad leaders.¹ Each squad is comprised of two fire teams. Each fire team is composed of two or more buddy teams. Each buddy team is composed of two soldiers (buddies).

¹This is only true for platoon sized patrol base operations. Squad-sized patrol base operations consist of only one squad.

3.1.5 Accidents/Losses

Accidents/losses for the patrol base operations are shown in table 3.3.

Accidents/Losses	
	Description
L1	MIA, KIA, WIA, CIA
L2	Wrong mission
L3	Mission Failure
L4	Negative publicity/exposure/unwanted attention
L5	Equipment loss/damage/capture by enemy
L6	Civilian casualties/disruption to local population
L7	Insufficient communications with high-up HQ

Table 3.3: System accidents/losses.

Note the following definitions: MIA = missing in action; KIA = killed in action; WIA = wounded in action; CIA = captured in action.

3.1.6 System-level Hazards/Vulnerabilities And Constraints

Table 3.4 describes the system-level hazards/vulnerabilities and associated system-level constraints. The system-level hazards/vulnerabilities are described in the second column. The associated accidents/losses for each hazard are shown in the next column. These are linked to system-level constraints in the last column.

Hazards/Vulnerabilities and constraints				
System-level Hazards/Vulnerabilities		Accidents/Losses	System-level Constraints	
H1	Insufficient planning	L1, L2, L3, L4, L5, L6, L7	SC1	Plan sufficiently
H2	Insufficient reconnaissance/intelligence	L1, L2, L3, L4, L5, L6, L7	SC2	Reconnoiter sufficiently
H3	Insufficient security	L1 ,L3, L4, L5, L6	SC3	Provide sufficient security
H4	Insufficient control	L1, L2, L3, L4, L5, L6 ,L7	SC4	Maintain adequate control
H5	Insufficient common sense	L1, L2, L3, L4, L5, L6, L7	SC5	Use common sense
H6	Insufficient leadership	L1, L2, L3, L4, L5, L6, L7	SC6	Establish sufficient leadership
H7	Insufficient communication with higher-HQ	L1, L2, L3, L4, L5, L6, L7	SC7	Maintain sufficient communications with higher-HQ
H8	Insufficient haste	L1, L3	SC8	Maintain appropriate pace
H9	Disregard for Rules of Engagement	L4, L6	SC9	Follow Rules of Engagement

Table 3.4: System-level hazards/vulnerabilities and constraints.

A more in-depth analysis would return to this table and generate refined hazards/vulnerabilities and constraints. For example, H1 could be refined to H1.1 insufficient reconnaissance, H1.2 mission objectives not clear, H1.3 insufficient time for completing plan, etc.. For the constraints, SC1.1 reconnoiter efficiently, SC1.2 confirm mission objectives, SC1.3 use time efficiently, etc..

3.2 Step 2: Model The Control Structure

This section first divides the patrol base operations into phases. It then describes the system controllers and their process models. It then describes the control actions for each controller. Finally, it presents a diagram of the functional control model.

3.2.1 Phases of The Patrol Base Operations

Figure 3.5 shows the patrol base operations divided into six phases. This is a simplified first approximation of the patrol base operations and is very abstract. For the purposes of this analysis, transitions among phases of the operations are indicated by the Platoon Leader who is responsible for the success of the mission as a whole.

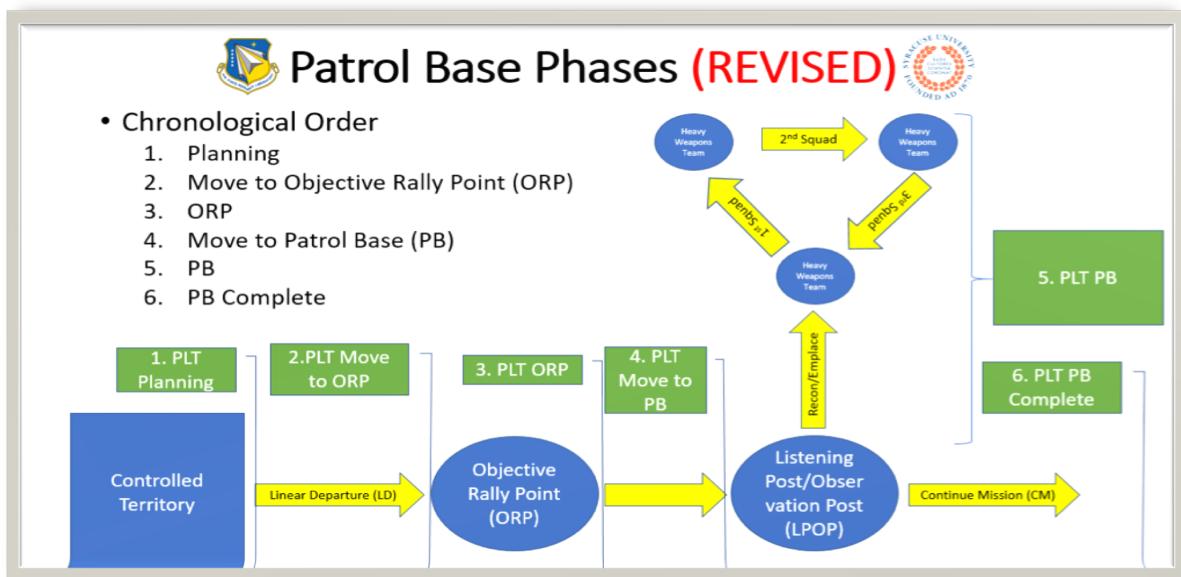


Table 3.5: Scenarios for UCA E1A3.

3.2.2 Controllers And Process Models

The controllers and their process models are shown in table 3.6. There are three

Controllers And Process Model			
Controller	Model	Variables	Values
Platoon Leader	Patrol Base Status	State	<ul style="list-style-type: none"> • PLAN_PB • MOVE_TO_ORP • CONDUCT_ORP • MOVE_TO_PB • CONDUCT_PB • COMPLETE_PB • ABORT_PB
	Policy	Authorized	<ul style="list-style-type: none"> • True • False
Platoon	Authentication Model	Authenticated	<ul style="list-style-type: none"> • True • False
Adversary	Threat Model	To be determined	to be determined

Table 3.6: Controllers and process model.

controllers: the Platoon Leader, the platoon, and the adversary (enemy). The Platoon Leader is the top controller. She issues orders to the subordinates (the platoon) based on her knowledge of the status and policy of the operations. The platoon authenticates the Platoon Leader and either executes the orders or discards the commands.

Each controller has one or more models of the patrol base operations that governs her decisions and actions. Each model has a variable that can take on one of several values. For example, the Platoon Leader has a model of the Patrol Base Status. The status is the State of the system. For example, the State could be PLAN_PB or

MOVE_TO_ORP.

The Platoon Leader's Policy model defines which conditions and which commands are authorized. For example, for most instances of the patrol base operations, the planning phase should be completed before moving to the objective rally point (ORP).² Therefore, the Platoon Leaders policy would include the condition that *planning phase complete implies move to the ORP*. The Authorization variable for the Policy model would be true if the phase is PLAN_PB and this phase is complete.

The platoon's Authentication Model defines how the platoon authenticates commands. For example, the platoon may authorizes commands by visual confirmation of the leader issuing the command. Then, whenever the Platoon Leader issues the command in the line of sight of the platoon, the Authentication variable would take the value True.

The adversary's process model is the threat model. It is determined from METT-TC³ analysis. It may or may not be known or complete. The more known and complete the model, the more capable the platoon is at anticipating threats from the enemy. But, knowledge of a specific enemy should not guide the system design. The design should be defensible, minimizing risk from any enemy.

²The most likely variation is for the platoon to move to the ORP and then receive orders. Regardless, it is unlikely that the platoon will move to the patrol base operations before completing planning.

³mission, enemy, terrain and weather, troops and support available-time available, and civil considerations factors.

3.2.3 Control Actions

Each controller has its own set of actions (commands, etc.) that it can issue. The control actions for each principal are listed below.

- Platoon Leader
 - PL says planComplete (same as crossLD)
 - PL says conductORP
 - PL says moveToPB
 - PL says completePB
 - PL says reactToContact
 - PL says returnToBase
 - PL says changeMission
- Platoon
 - exec(crossLD) (same as planComplete)
 - exec(conductORP)
 - exec(moveToPB)
 - exec(completePB)
 - exec(reactToContact)
 - exec(returnToBase)
 - exec(changeMission)
 - discard(anyCommand)

For example, the Platoon Leader may determine (based on policy, etc.), that the planning phase is complete. She may then indicate this with the statement *PL says*

planComplete.⁴ Similarly, the platoon may respond to this command with *exec(planComplete)*. If the platoon is in the PLAN_PB phase, it would then move to the next phase MOVE_TO_ORP.

The *discard(anyCommand)* indicates that the command, whatever it is, should be discarded.

3.2.4 Functional Control Structure

The functional control structure diagrams the command and control structure with feedback. It is shown in figure 3.1.

The blue background encapsulates the command and control structure of the operations. The yellow background encapsulates the access-control structures. The green is the adversary's realm with both command and control and access-control capabilities.

Downward pointing arrows indicate the direction of commands and upward pointing arrows indicate feedback. The adversary is an exception. These arrows are directed out towards the controllers (PL and platoon) and the controlled actions (patrol base operations). Feedback to the adversary is not modeled. However, modeling feedback may shine light on avenues of attack or counter-attack on the adversary.

The Platoon Leader is the legitimate controller in the command and control structure. She sends commands to the platoon. The commands are shown in the grey box. The Platoon Leader receives feedback from the platoon (i.e, discard command, confirm orders, etc.). The Platoon Leader also receives orders or intelligence from higher-up headquarters, the operations, and potentially from other sources.

⁴crossLD = cross line of discrimination. Depending on the model, this would occur after planning. It would transition the patrol into the MOVE_TO_ORP phase of the operations. A description of this works is reserved for chapter 5.

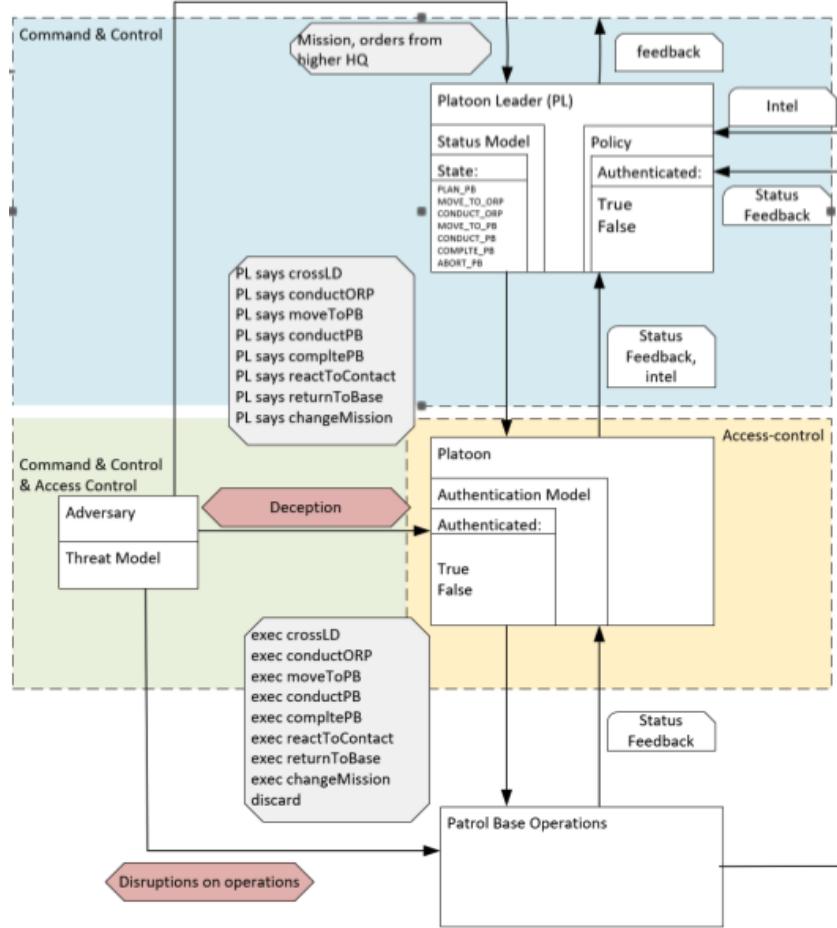


Figure 3.1: Control structure for patrol base operations.

The platoon receives orders from the Platoon Leader and either executes them or discards them. Execution of an order implies that the Platoon Leader's command was authenticated. Discarding means that the command was not authenticated. The platoon also receives information from the operations as to what phase it is in, etc..

The adversary can interfere with the operations in numerous ways. It can attack the Platoon Leader or platoon. It can send conflicting messages, such as send unauthenticated commands in the hope that the platoon or Platoon Leader will authenticate it. The adversary can attack or sabotage the platoon. The capabilities of the adversary to interrupt the operations should be analyzed as soon as possible to develop an accurate threat model.

3.3 Step 3: Identify Unsafe Control Actions (UCAs)

This section determines unsafe control actions (UCA) for the patrol base operations. It uses the Thomas Method to identify UCAs. This method considers all possible combinations of values of the process model variables.

All possible process model values are analyzed for each control action. This means that for any control action the possible combinations of process model values is 7 (seven states) $\times 2$ (Authorized: True or False) $\times 2$ (Authenticated: True or False) = 28 .

There are 7 possible control actions (commands) for the Platoon Leader and 2×7 possible control actions for the platoon. In total, there are $7 + 2 \times 7 = 21$ possible control structures. To analyze each control structure requires $21 \times 28 = 588$ possible UCAs.

Fortunately, many UCAs are similar or can be grouped. Nevertheless, to limit this analysis for its demonstrative purposes only three control actions are analyzed: *PL says planComplete*, *exec(planComplete)*, and *discard(anyCommand)*. Notice that the last control action covers all 7 discard commands because discarding any command is the same for all commands.

The UCAs are analyzed using the tables described next. In each of these tables, each control action is analyzed for each combination of process model value and for each of the four types of UCAs. The four UCAs described in section 2.3.2 of the previous chapter are repeated in the list below.

- Not applying the control action
- Applying the wrong control action
- Applying the control action in the wrong order

- Applying the control action for too long or not long enough.

To make the tables more manageable, for each combination of control action and process model value the table indicates if there is an UCA by marking the appropriate box with a code. The code will be used in step 4 to describe scenarios. If the box is empty, then no UCA is determined for those set of values.

3.3.1 Unsafe Control Actions (UCAs): Thomas Model

PL says planComplete Table 3.7 analyzes the UCAs for the control action *PL says planComplete.*

Control Actions			Process Model Values			Unsafe Control Actions			
Control Actions	Source	Destination	Authorized	Authenticated	State	Providing	Not Providing	Wrong Timing/ Order	Stopped Too Soon/ Applied Too Long
PL Says <u>planComplete</u>	Platoon Leader	Platoon	yes	yes	PLAN_PB	P1A1	P1B1	P1C1	
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
					CONDUCT_PB				
					COMPLETE_PB				
					ABORT_PB				
			yes	no	PLAN_PB	P1A2			
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
					CONDUCT_PB				
					COMPLETE_PB				
					ABORT_PB				
			no	yes	PLAN_PB	P1A3			
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
					CONDUCT_PB				
					COMPLETE_PB				
					ABORT_PB				
			no	no	PLAN_PB	P1A4			
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
					CONDUCT_PB				
					COMPLETE_PB				
					ABORT_PB				

Table 3.7: Unsafe control actions UCAs for control action "PL says planComplete."

`exec(planComplete)` Table 3.8 analyzes the UCAs for the control action

`exec(planComplete).`

Control Actions			Process Model Values			Unsafe Control Actions			
Control Actions	Source	Destination	Authorized	Authenticated	State	Providing	Not Providing	Wrong Timing/ Order	Stopped Too Soon/ Applied Too Long
<code>exec planComplete</code>	Platoon	Patrol Base Ops	yes	yes	PLAN_PB	E1A1	E1B1	E1C1	
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
					CONDUCT_PB				
					COMPLETE_PB				
			yes	no	ABORT_PB	E1A2			
					PLAN_PB				
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
					CONDUCT_PB				
			no	yes	COMPLETE_PB	E1A3			
					ABORT_PB				
					PLAN_PB				
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
			no	no	CONDUCT_PB	E1A4			
					COMPLETE_PB				
					ABORT_PB				
					PLAN_PB				
					MOVE_TO_ORP				
					CONDUCT_ORP				

Table 3.8: Unsafe control actions UCAs for control action "exec(planComplete)."

discard(anyCommand) Table 3.9 analyzes the UCAs for the control action *discard(anyCommand)*.

Control Actions			Process Model Values			Unsafe Control Actions			
Control Actions	Source	Destination	Authorized	Authenticated	State	Providing	Not Providing	Wrong Timing/ Order	Stopped Too Soon/ Applied Too Long
discard	Platoon	Patrol Base Ops	yes	yes	PLAN_PB	D1A1			
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
					CONDUCT_PB				
					COMPLETE_PB				
			yes	no	ABORT_PB	D1B1			
					PLAN_PB				
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
					CONDUCT_PB				
			no	yes	COMPLETE_PB	D1A2			
					ABORT_PB				
					PLAN_PB				
					MOVE_TO_ORP				
					CONDUCT_ORP				
					MOVE_TO_PB				
			no	no	CONDUCT_PB	D1B2			
					COMPLETE_PB				
					ABORT_PB				

Table 3.9: Unsafe control actions UCAs for control action "discard(anyCommand)." "

3.4 Step 4: Identify Loss Scenarios

This section describes the scenarios associated with each UCA identified in the previous step.

3.4.1 Scenarios

The scenarios are described in a series of tables. The top row of each table indicates which control action is analyzed. The next row describes the process model variable values. It may also contain other useful information.

The first column, labeled "Scenarios", identifies the UCA by its code (from the tables in the previous section. It also describes the unsafe scenario. The next column, labeled "Associated Causal Factors", describes factors that could cause the unsafe scenario. The third column describes the rationale for mitigating or avoiding the causal factors.

The last two columns contain lists of alternating black and blue entries. The alternating entries in the second column correspond to the alternating black and blue entries in the third column.

P1 says planComplete

The next set of tables describes scenarios for the *PL says planComplete* control action.

P1B1/P1C1: state = PLAN_PB, Authorized = Yes, Authenticated = Yes

Tables 3.10 and 3.11 present the scenarios for the P1B1 and P1C1 UCAs.

UCA: control action: PL says planComplete		
state = PLAN_PB, Authorized = Yes, Authenticated = Yes. PL fails to issue command to complete the plan when she is authorized to do so.		
Scenarios	Associated Causal Factors	Rationale
P1B1. Failure to act when action is required. H1,H4,H6,H8.	<ul style="list-style-type: none"> PL overloaded/too much work for one person. PL not effectively delegating tasks to subordinates. Mutiny/rogue leadership. PL incapacitated. 	<ul style="list-style-type: none"> Limit span of control. Assign tasks to subordinates early in mission. Discourage mutiny! Foster confidence in leadership. Vet leaders. Establish SOP for transfer of command. GOTWA.
P1B1. Communication is sent but somehow is corrupted in transmission. H1,H2,H4, H5,H6,H8.	Ineffective communications* equipment. <ul style="list-style-type: none"> Equipment not checked before mission. Wrong equipment. Battery dead, equipment fails in service. Equipment damaged or not functioning while on mission. Lack of operational know how. Platoon not all on same channel. Terrain interference on transmission. Communications equipment cannot otherwise be used. 	<ul style="list-style-type: none"> Check equipment prior to mission (functioning?, battery, antenna). Choose communication devices that will satisfy mission needs. Know operating distances, etc., of equipment. Bring back-up communication equipment, batteries, antenna. Tools and know-how to troubleshoot and repair equipment. Train soldiers on equipment use. Communicate communication channel to subordinates. Move to a better position. Contingency communications plan for equipment failure.

Table 3.10: Scenarios for UCA P1B1.

<p>UCA: control action: PL says planComplete</p> <p>state = PLAN_PB, Authorized = Yes, Authenticated = Yes.</p> <p>PL fails to issue command to complete the plan when she is authorized to do so.</p>		
Scenarios	Associated Causal Factors	Rationale
P1B1 and P1C1. Adversary interferes with communications (i.e., blocks radio signals).	<ul style="list-style-type: none"> Enemy interfering with communications 	<ul style="list-style-type: none"> Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles , flares).
P1C1. PL does not complete plan with sufficient haste. H1,H5,H6,H8.	<ul style="list-style-type: none"> PL does not plan to complete mission with sufficient haste. Circumstances require planning to be completed at another time. Enemy disruptions to operations cause plan completion to be delayed or aborted. METT-TC considerations interfere with completion of plan. Insufficient intel/recon to complete plan. 	<ul style="list-style-type: none"> Establish standards for length of time needed to plan, train leaders, rehearse. Have a contingency plan for what to do until plan is complete. Analyze enemy. Establish plan for early contact. Gather intel for METT-TC before mission. Establish contingency plan. Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ.

Table 3.11: Scenarios for UCA P1B1/P1C1.

P1A1: state = PLAN_PB, Authorized = Yes, Authenticated = Yes Table

3.12 presents the scenarios for the P1A1 UCA.

UCA: control action: PL says planComplete		
state ≠ PLAN_PB, Authorized = Yes, Authenticated = Yes. (Command should not be issued)		
Scenarios	Associated Causal Factors	Rationale
P1A1. PL never began or completed the plan. H1,H4,H6.	<ul style="list-style-type: none"> Circumstances require planning to be completed at another time. Enemy disruptions to operations cause plan completion to be delayed or aborted. METT-TC considerations interfere with completion of plan. Insufficient intel/recon to complete plan. PL incapacitated. Change of leadership without sufficient transfer of command. 	<ul style="list-style-type: none"> Have a contingency plan for what to do until plan is complete. Analyze enemy. Establish plan for early contact. Gather intel for METT-TC before mission. Establish contingency plan. Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ. Establish SOP for transfer of command. GOTWA. Establish SOP for transfer of command. GOTWA.
P1A1. Need to re-plan H3,H4,H6.	<ul style="list-style-type: none"> Contingency plan not established early in mission. Higher-up HQ presents new intel or new mission objectives. Enemy disruptions to operations make previous plan obsolete. METT-TC considerations interfere with completion of plan. New intel/recon requires significant change in plan. Change of leadership without sufficient transfer of command. 	<ul style="list-style-type: none"> Establish contingency plan early in mission. Contingency plan (training, SOP) for re-planning during mission. Analyze enemy. Establish plan for early contact. Gather intel for METT-TC before mission. Establish contingency plan. Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ. Establish SOP for transfer of command. GOTWA.

Table 3.12: Scenarios for UCA P1A1.

P1A2: state = any state, Authorized = Yes, Authenticated = No Table 3.13

presents the scenarios for the P1A2 UCA.

UCA: control action: PL says planComplete		
state = any state, Authorized = Yes, Authenticated = No (Command should not be issued)		
Scenarios	Associated Causal Factors	Rationale
P1A2. Adversary issues command. H1,H3,H4,H5,H6.	<ul style="list-style-type: none"> • Adversary has commandeered (lost or stolen) communications equipment. • Adversary is otherwise attempting to issue commands to platoon. 	<ul style="list-style-type: none"> • Secure equipment. • Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles , flares).
P1A2. Platoon does not recognize PL because of communications equipment failure. H1,H3,H4,H5,H6.	<ul style="list-style-type: none"> • Equipment not checked before mission. • Wrong equipment. • Battery dead, equipment fails in service. • Equipment damaged or not functioning while on mission. • Lack of operational know how. • Platoon not all on same channel. • Terrain interference on transmission. • Communications equipment cannot otherwise be used. 	<ul style="list-style-type: none"> • Check equipment prior to mission. • Choose communication devices that will satisfy mission needs. Know operating distances, etc., of equipment. • Bring back-up communication equipment, batteries, antenna. • Tools and know-how to troubleshoot and repair equipment. • Train soldiers on equipment use. • Communicate communication channel to subordinates. • Move to a better position. • Contingency communications plan for equipment failure.
P1A2. Platoon does not recognize PL for other reasons. H1,H3,H4,H5,H6.	<ul style="list-style-type: none"> • PL incapacitated. • Change of leadership without sufficient transfer of command. • Mutiny! • No line-of-sight for visual communications signals. 	<ul style="list-style-type: none"> • Establish SOP for transfer of command. GOTWA. • Establish SOP for transfer of command. GOTWA. • Discourage mutiny! Foster confidence in leadership. Vet leaders. • Move within line-of-sight. Contingency communications for no line-of-sight. Animal calls, radios, etc.

Table 3.13: Scenarios for UCA P1A2.

P1A3: state = any state, Authorized = No, Authenticated = Yes Table 3.14

presents the scenarios for the P1A3 UCA.

UCA: control action: PL says planComplete		
state = any state, Authorized = No, Authenticated = Yes (Command should not be issued)		
Scenarios	Associated Causal Factors	Rationale
P1A3. PL indicates planning is complete when it is not. H1,H2,H4.	<ul style="list-style-type: none"> Inadequate leadership. Change of leadership without sufficient transfer of command. Mutiny/rogue leadership. 	<ul style="list-style-type: none"> Vet leadership. Establish SOP for transfer of command. GOTWA. Discourage mutiny! Foster confidence in leadership. Vet leaders.

Table 3.14: Scenarios for UCA P1A3.

P1A4: state = any state, Authorized = No, Authenticated = No Table 3.15

presents the scenarios for the P1A4 UCA.

UCA: control action: PL says planComplete		
state = any state, Authorized = No, Authenticated = No (Command should not be issued)		
Scenarios	Associated Causal Factors	Rationale
P1A4. Adversary issues command. H1,H3,H4,H5,H6.	<ul style="list-style-type: none"> Adversary has commandeered (lost or stolen) communications equipment. Adversary is otherwise attempting to issue commands to platoon. 	<ul style="list-style-type: none"> Secure equipment. Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares).

Table 3.15: Scenarios for UCA P1A4.

exec(planComplete)

The next set of tables describes scenarios for the *exec(planComplete)* control action.

E1B1/E1C1: state = PLAN_PB, Authorized = Yes, Authenticated = Yes

Table 3.16 present the scenarios for the E1B1 and E1C1 UCAs.

UCA: control action: exec planComplete		
state = PLAN_PB, Authorized = Yes, Authenticated = Yes. Command should be given. Not given is an UCA.		
Scenarios	Associated Causal Factors	Rationale
E1B1. Failure to act when action is required. H4,H6,H8.	<ul style="list-style-type: none"> Mutiny. Platoon does not follow orders. PL overloaded/too much work for one person. Platoon is incapacitated and unable to follow orders. Perceived conflict with ROE. Assigned tasks are not clear. METT-TC conditions interfere with execution of orders. 	<ul style="list-style-type: none"> Discourage mutiny. Foster confidence in leadership. Limit tasks to a manageable number. Contingency plan for handling incapacitated soldiers and re-assigned responsibilities. Enforce ROE. Establish procedures for ROE problems (i.e., change of command, etc.). Foster confidence in leadership. Assign tasks, clarify, and rehearse. Gather intel for METT-TC before mission. Establish contingency plan.
E1B1 and E1C1. Adversary interferes with operations (i.e., engages platoon, etc.). H1,H3,H4,H5.	<ul style="list-style-type: none"> Contact with enemy. Sabotage of plans. 	<ul style="list-style-type: none"> Conduct enemy analysis. Develop contingency plan for operations, contact with enemy, etc. Conduct enemy analysis. Contingency plan.
E1C1. exec planComplete not provided soon enough. H3,H4,H6,H8.	<ul style="list-style-type: none"> METT-TC conditions interfere with execution of orders. PL overloaded/too much work for one person. Lack of discipline. 	<ul style="list-style-type: none"> Gather intel for METT-TC before mission. Establish contingency plan. Limit tasks to a manageable number. Train soldiers to be disciplined.

Table 3.16: Scenarios for UCAs E1B1 and E1C1.

E1A1: state ≠ PLAN_PB, Authorized = Yes, Authenticated = Yes Table

3.17 present the scenarios for the E1A1 UCAs.

UCA: control action: exec planComplete		
state ≠ PLAN_PB, Authorized = Yes, Authenticated = Yes. (Command should not be issued) This is a problem if the planning should be completed before commencing with actions. But, in some cases, for example, the platoon may move to the ORP before receiving the mission. In this case, the PLAN_PB state would follow the MOVE_TO_ORP state. Regardless, indication that the plan is complete when not in the planning phase is the UCA considered.		
Scenarios	Associated Causal Factors	Rationale
E1A1. Planning never began or completed. H1,H4,H5,H6.	<ul style="list-style-type: none"> Circumstances require planning to be completed at another time. Enemy disruptions to operations cause plan completion to be delayed or aborted. METT-TC considerations interfere with completion of plan. Insufficient intel/recon to complete plan. PL incapacitated. Change of leadership without sufficient transfer of command. 	<ul style="list-style-type: none"> Have a contingency plan for what to do until plan is complete. Analyze enemy. Establish plan for early contact. Gather intel for METT-TC before mission. Establish contingency plan. Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ. Establish SOP for transfer of command. GOTWA. Establish SOP for transfer of command. GOTWA.
E1A1. Need to re-plan H3,H4,H6.	<ul style="list-style-type: none"> Contingency plan not established early in mission. Higher-up HQ presents new intel or new mission objectives. Enemy disruptions to operations make previous plan obsolete. METT-TC considerations interfere with completion of plan. New intel/recon requires significant change in plan. Change of leadership without sufficient transfer of command. 	<ul style="list-style-type: none"> Establish contingency plan early in mission. Contingency plan (training, SOP) for re-planning during mission. Analyze enemy. Establish plan for early contact. Gather intel for METT-TC before mission. Establish contingency plan. Contingency plan for planning without sufficient intelligence. Request intel from higher-up HQ. Establish SOP for transfer of command. GOTWA.

Table 3.17: Scenarios for UCA E1A1.

E1A2: state = any state, Authorized = Yes, Authenticated = No Table 3.18

present the scenarios for the E1A2 UCAs.

UCA: control action: exec planComplete		
state = any state, Authorized = Yes, Authenticated = No (Command should not be issued).		
Scenarios	Associated Causal Factors	Rationale
E1A2. Adversary issues command. H1,H3,H4,H5,H6.	<ul style="list-style-type: none"> Adversary has commandeered (lost or stolen) communications equipment. Adversary is otherwise attempting to issue commands to platoon. 	<ul style="list-style-type: none"> Secure equipment. Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles , flares).
E1A2. Platoon does not recognize PL because of communications equipment failure. H1,H3,H4,H5,H6.	<ul style="list-style-type: none"> Equipment not checked before mission. Wrong equipment. Battery dead, equipment fails in service. Equipment damaged or not functioning while on mission. Lack of operational know how. Platoon not all on same channel. Terrain interference on transmission. Communications equipment cannot otherwise be used. 	<ul style="list-style-type: none"> Check equipment prior to mission (functioning?, battery, antenna). Choose communication devices that will satisfy mission needs. Know operating distances, etc., of equipment. Bring back-up communication equipment, batteries, antenna. Tools and know-how to troubleshoot and repair equipment. Train soldiers on equipment use. Communicate communication channel to subordinates. Move to a better position. Contingency communications plan for equipment failure.
E1A2. Platoon does not recognize PL for other reasons. H1,H3,H4,H5,H6.	<ul style="list-style-type: none"> PL incapacitated. Change of leadership without sufficient transfer of command. Mutiny! No line-of-sight for visual communications signals. 	<ul style="list-style-type: none"> Establish SOP for transfer of command. GOTWA. Establish SOP for transfer of command. GOTWA. Discourage mutiny! Foster confidence in leadership. Vet leaders. Move within line-of-sight. Contingency communications for no line-of-sight. Animal calls, radios, etc.

Table 3.18: Scenarios for UCA E1A2.

E1A3: state = any state, Authorized = No, Authenticated = Yes Table 3.19

present the scenarios for the E1A3 UCAs.

UCA: control action: exec planComplete		
state = any state, Authorized = No, Authenticated = Yes (Command should not be issued). This problem occurs at the Platoon Leader control level...and trickles down to the platoon.		
Scenarios	Associated Causal Factors	Rationale
E1A3. PL indicates planning is complete when it is not. Platoon executes command because PL is authenticated. H1,H2,H4.	<ul style="list-style-type: none"> Inadequate leadership. Change of leadership without sufficient transfer of command. Mutiny/rogue leadership. Platoon does not recognize unauthorized requests. 	<ul style="list-style-type: none"> Vet leadership. Establish SOP for transfer of command. GOTWA. Discourage mutiny! Foster confidence in leadership. Vet leaders. For the most part, soldiers follow orders. However, platoon should be trained on expectations regarding the operations and leader's duties and commands. Use common sense.

Table 3.19: Scenarios for UCA E1A3.

E1A4: state = any state, Authorized = No, Authenticated = No Table 3.20

present the scenarios for the E1A4 UCAs.

UCA: control action: exec planComplete		
state = any state, Authorized = No, Authenticated = No (Command should not be issued).		
Scenarios	Associated Causal Factors	Rationale
E1A4. Adversary issues command. H1,H3,H4,H5,H6.	<ul style="list-style-type: none"> Adversary has commandeered (lost or stolen) communications equipment. Adversary is otherwise attempting to issue commands to platoon. 	<ul style="list-style-type: none"> Secure equipment. Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares).

Table 3.20: Scenarios for UCA E1A4.

discard(anyCommand)

Table 3.21 describes scenarios for the *discard(planComplete)* control action.

SCENARIOS	UCA: control action: discard	
state = any state, Authorized = Yes, Authenticated = Yes. Providing is an UCA. Discarding means that the platoon refuses to execute the command for some reason.		
Scenarios	Associated Causal Factors	Rationale
D1A1. Command discarded when it should be executed. H3,H4,H6,H8.	<ul style="list-style-type: none"> • Mutiny. • Lack of discipline. 	<ul style="list-style-type: none"> • Discourage mutiny! Foster confidence in leadership. Vet leaders. • Train soldiers to be disciplined.
state = any state, Authorized = Yes, Authenticated = No. Not providing is an UCA.		
D1B1. Unauthenticated command not discarded. H3,H4,H6.	<ul style="list-style-type: none"> • Authentication procedures not clear. 	<ul style="list-style-type: none"> • Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares). Rehearse.
state = any state, Authorized = No, Authenticated = Yes. Providing is an UCA. Unauthorized commands are a problem with the Platoon Leader control structure...trickles down to the platoon. But, from the process model of the platoon, this command should not be discarded because it is authorized.		
D1A2. Command discarded when it should be executed. H3,H4,H6,H8.	<ul style="list-style-type: none"> • Mutiny. • Lack of discipline. 	<ul style="list-style-type: none"> • Discourage mutiny! Foster confidence in leadership. Vet leaders. • Train soldiers to be disciplined.
state = any state, Authorized = No, Authenticated = No. Not providing is a UCA.		
D1B2. Unauthenticated command not discarded. H3,H4,H6	<ul style="list-style-type: none"> • Authentication procedures not clear. 	<ul style="list-style-type: none"> Establish authentication techniques: passwords, non-radio communication signals (i.e., arm signals, whistles, flares). Rehearse.

Table 3.21: Scenarios for UCA for discard on all commands (D1A1 through D1A4).

3.5 Discussion And Conclusions

This chapter describes the STPA/STPA-Sec analysis on an abstract model of the patrol base operations. From this analysis, numerous recommendations regarding safety and security are derived (step 4). These recommendations ("Rationale" in the tables) would be passed to the systems engineers to inform the security aspects of the system design.

The reader who is familiar with the patrol base operations and military procedures in general may recognize the implementation of some of the rationale. For example, contingency plans are a standard for many phases of the operations. The Ranger Handbook explains basic repair techniques for communications equipment. METT-TC analysis is standard analysis for the patrol base operations. GOTWA is standard when a leader leaves the platoon. Some of the recommendation are common sense. For example, discourage mutiny and build confidence in leadership. Train soldiers to be disciplined. Secure equipment.

A more thorough analysis should find more recommendations that are specific to particular actions. For example, the troop leading procedures comprise an eight step planning phase. Each step should be analyzed using STPA/STPA-Sec to identify unsafe control actions that arise from system-level interactions.

In a thorough analysis of the patrol base operations, the system-level hazards/vulnerabilities would be furthered refined. The system-level constraints would also be further refined. The degree of refinement depends on the purpose of the analysis and the safety and security needs of the system. The process can be lengthy. But, it is manageable with tables and the repetitive nature of many of the unsafe conditions. There is also software available that automates some of the analysis. This software is new and continues to be refined and improved.

The next chapters explain the second component of STORM and its application to the

patrol base operations

Chapter 4

Certified Security by Design (CSBD)

&

Access-Control Logic (ACL)

4.1 Certified Security by Design (CSBD)

"Providing satisfactory security controls in a computer system is in itself a system design problem."

Rand Corporation, 1970 [20]

Certified Security by Design (CSBD) is a method for formally verifying and documenting the security properties of a system. It satisfies the principle of complete mediation. It uses an access-control logic (ACL) to reason about access to security sensitive objects. 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 are consistent with the Systems Security Engineering (SSE) Framework described in NIST SP 800-160 [1]. They serve as a reproducible and auditable documentation of trustworthiness.

This chapter first describes CSBD's access-control logic (ACL) used to prove complete mediation. It then describes ACL's implementation in the Higher Order Logic (HOL) Interactive Theorem prover. Properties of transition systems are described in chapter 6. The HOL code is included with this thesis in the folder /MasterThesis/HOL/ACL and pretty-printed code is included in appendix B.

4.2 Access-Control Logic (ACL)

The ACL is a logic for reasoning about security in the sense of CIA. It develops formal proofs based on a formal propositional logic to verify complete mediation. This documentation is reproducible and consistent with the SSE Framework.

4.2.1 ACL: A Command and Control (C2) Calculus

In the jargon of the day ACL is a command and control (C2) calculus with a set of axioms and logical rules used to reason about access-control. At its most basic, the ACL is composed of principals that make requests to access objects. These principals are given authority or jurisdiction over objects. These principals must be authenticated before they can access or modify objects over which they have authority. The following sections describe the components and logical rules of the ACL.

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, Jurisdiction, and Delegation

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 this master thesis and doing something", where the propositional variable "doing something" could be replaced with "I am reading now" or "I am drinking coffee", etc..

Principals can make requests. In the ACL, principals make requests using the *says* operator. Requests have the form P *says* φ , where P represents some principal and φ

represents some assertion (proposition, propositional variable). 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. The ACL conveys authority 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 states that the Platoon Leader has the authority over 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. The *speaks for* operator is typically used to reason about certifications issued by some authority. For example, a driver's license speaks for the Department of Motor Vehicles.

Principals can also delegate authority. Principals do this using the *reps* rule. The *reps* rule assigns authority given to one principal to another principal. For example, the Platoon Leader *reps* the President of the United States on platoon leading commands.

The concept of assigning integrity and security levels (such as classification) to principals and objects is also covered in the ACL. But, they are not discussed in this master thesis because they are beyond the scope of this work.

4.2.4 Well-formed Formulas

Well-formed formulas (WFFs) define the syntax (or grammatical structure) of the propositional logic. They are syntactically valid statements in the ACL. All ACL statements must be formulated as WFFs. The syntax is defined as follows

$$\begin{aligned} \text{Form} ::= & \text{PropVar} / \neg \text{Form} / (\text{Form} \vee \text{Form}) / \\ & (\text{Form} \wedge \text{Form}) / (\text{Form} \supset \text{Form}) / (\text{Form} \equiv \text{Form}) / \\ & (\text{Princ} \Rightarrow \text{Princ}) / (\text{Princ says Form}) / (\text{Princ controls Form}) / \\ & (\text{Princ} \text{ reps } \text{Princ} \text{ on Form}^1) \end{aligned}$$

This is a recursive definition. ***PropVar*** is a propositional variable. The symbols \neg , \vee , \wedge , \subset , and \equiv are the standard set and logical symbols. They represent "not", "or", "and", "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 [21].

Whereas WFFs define the syntax of the propositional logic, Kripke semantics describe the semantics. Semantics refers to the meaning of statements.

¹The last line is from [7].

4.2.5.1 Kripke Structures

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, 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.
- $I : \text{PropVar} \rightarrow \mathcal{P}(W)$ is an interpretation function that maps each propositional variable to a set of worlds.
- $J : \text{PName} \rightarrow \mathcal{P}(W \times W)$ 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. 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.

4.2.5.2 Kripke Semantics

The Kripke semantics define the meanings of WFFs 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.

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

Figure 4.1: Kripke semantics. Image taken from *Access Control, Security, and Trust: A Logical Approach*[6]

4.2.5.3 Satisfies

The "satisfies" condition applies to a particular Kripke structure $\mathcal{M} = \langle W, I, J \rangle$. It is said that \mathcal{M} satisfies some proposition φ if the evaluation function $\mathcal{E}_{\mathcal{M}}[\varphi] = W$ (the set 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 $\mathcal{M} \not\models \varphi$.

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 hypotheses, 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 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*[6].

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

Figure 4.2: The ACL inference rules. Image taken from *Access Control, Security, and Trust: A Logical Approach*[6]

4.2.7 Complete mediation

Fundamental to security is the concept of complete mediation. This means that each principal must be authenticated and authorized on each request. ACL does this primarily using the *Controls* inference rule shown 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*[6]

This rule has two hypotheses and one conclusion. The left hypothesis is an authorization.² The principal P controls (is authorized on) some assertion φ . The right hypothesis is a request.³ The principal P requests some assertion φ . The conjunction of the authorization and the request of P on φ results in the assertion φ . That is, if P controls φ and P says φ then φ is true.

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

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

Figure 4.4: The *Reps* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[6]

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 assertion φ . And also, P represents Q on that

²or a *control* in the C2 calculus

³or a *command* in the C2 calculus

assertion. In the ACL, this means that $P \text{ Reps } Q \text{ on } \varphi$. Note that $P \text{ says } \varphi$ is not the same thing as $P / Q \text{ says } \varphi$ (short for, $P \text{ says } Q \text{ says } \varphi$). 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.

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

Figure 4.5: The *Derived Speaks For* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[6]

The *Derived Speaks For* rule usually applies to jurisdiction. This rule is shown in figure 4.5. 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 say that $Q \text{ says } \varphi$.

4.3 ACL in HOL

The material in this section is adapted from *Access Control, Security, and Trust: A Logical Approach*[6] and *Certified Security by Design Using Higher Order Logic*[7]. 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. Pretty-printed HOL-generated output for all theories discussed in this section can be found in appendix B.

4.3.1 Principals

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

Princ is an algebraic data type. The concept of types are important in HOL. They are

```

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*[7]

discussed in the background chapter in section A.3.

The definitions here correspond to the **Princ** defined in section 4.2.2. The first line corresponds to **PropVar**, 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.

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

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 in HOL are:

Name PlatoonLeader, or

(Name PlatoonLeader) ` meet ` (Name PlatoonSergeant), or

(Name PlatoonLeader) ` quoting ` (Name PlatoonSergeant).

4.3.2 Well-Formed Formulas

The HOL representation of WFFs is 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*[7]

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 versions 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)`.

4.3.3 Kripke structures

Kripke structures are necessary to prove the “satisfies” and “soundness” properties of the ACL.

```
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*[7]

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

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 reader should 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 says</i> $\langle jump \rangle$	Name Alice says prop jump
<i>Alice & Bob says</i> $\langle stop \rangle$	Name Alice meet Name Bob says prop stop
<i>Bob Carol says</i> $\langle run \rangle$	Name Bob quoting Name Carol says prop run
<i>Bob controls</i> $\langle walk \rangle$	Name Bob controls prop walk
<i>Bob reps Alice on</i> $\langle jump \rangle$	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*[6]

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

```
prop command
```

A request in HOL is coded as:

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

Parentheses are used when necessary.⁴ 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.

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 B in the section on aclsemantics. The reader should note that the definition is defined for all Kripke structures as $\forall O_i O_s M$.

4.3.6 Satisfies And Soundness

The implementation of the "satisfies" property is shown in figure 4.11. Satisfies in this implementation is the same as soundness.

The definition reads "f is true for the Kripke structure if and only if it evaluates to all worlds." The fact that is is defined for all Kripke structures $\forall M O_i O_s$ makes it sound.

⁴The need for parentheses is well-defined. But, the author finds the best approach to parentheses is trial and error and generosity.

[Efn_def]

```

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

```

Figure 4.10: The HOL implementation of the Kripke semantics (evaluation function). Image from *Certified Security by Design Using Higher Order Logic*[7]

[sat_def]

```

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

```

Figure 4.11: The HOL implementation of the "satisfies" property. Image from *Certified Security by Design Using Higher Order Logic*[7]

The next chapter describes the patrol base operations as a modularized hierarchy of secure state machines.

Chapter 5

Patrol Base Operations

The patrol base operations are modeled as a modularized hierarchy of secure state machines. This model is a collaborative effort between the author of this thesis and a subject matter expert Jesse Nathaniel Hall from the U.S. Army. But, the details with regards to translating the Ranger Handbook into the model is primarily the work of Jesse Hall. He deserves a lot of credit for his work on this project.

To demonstrate properties of complete mediation on the patrol base operations, they are modeled as a hierarchy of secure state machines (SSMs). The hierarchy and modularization manage the complexity of the operations. The secure state machines (SSMs) constrain the behavior of the operations such that authentication and authorization can be formally verified and documented with an access-control logic and computer-aided reasoning.

This chapter describes the structure and details of the model. Chapter 6.2 describes SSMs in more detail.

5.1 Patrol Base Operations

Patrol base operations are described in the United States Army Ranger Handbook [9] in chapter 7 (2017 edition). The mission activity specification for the patrol base operations are shown in table 5.1

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 [9].

5.2 Overview of The Model

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.

The diagram describes a chronological order of abstract phases (modeled as states) of the patrol base operations. (1) The operations begin with the planning phase. (2) Next,

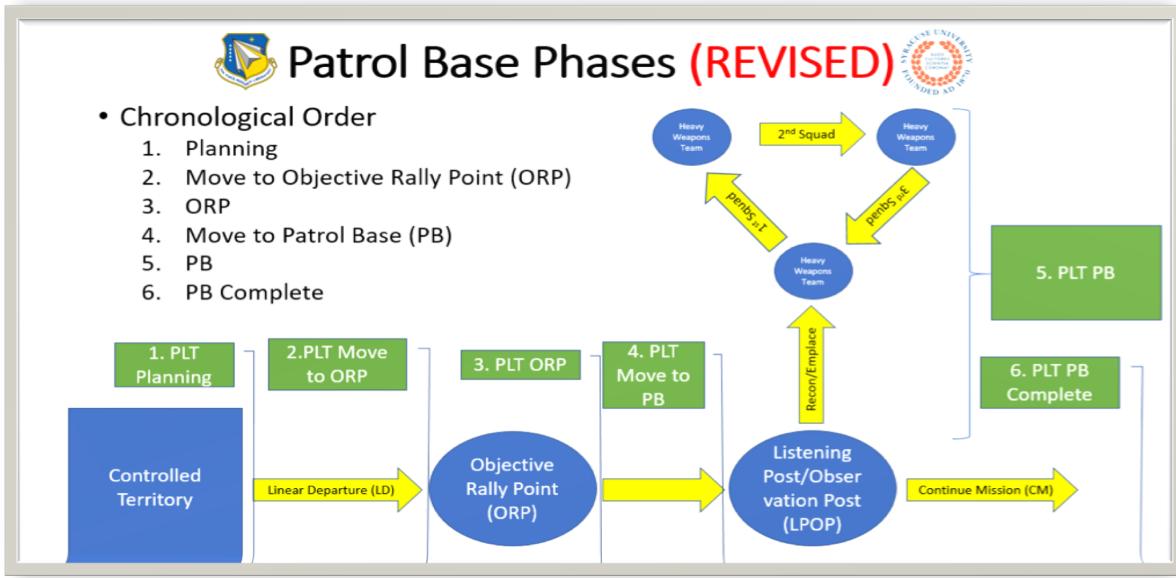


Figure 5.1: A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel Hall.

they move to the Objective Rally Point (ORP). (3) At the ORP, operations commence. (4) When these are complete, the patrol base operations move to the actual patrol base. (5) At the patrol base, operations proceed. (6) Finally, the patrol base operations are complete. The last phase is an end phase (actual withdraw, etc. of the operations are included in the 5th phase).

The next level of abstraction in the hierarchy of SSM represents a horizontal slice through the patrol base operations. 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 5.3.6.1) which detail activities conducted during the planning phase of the patrol base operations.

At yet another lower level of abstraction is the sub-sub (3rd) level. This expands on the states in the level above it. This is the last level of the hierarchy that is modeled in detail.¹

¹It is not necessary to model the entire system. To do so, is beyond the scope of this work

To demonstrate complete mediation on all levels of the hierarchy, a vertical slice is also modeled. Each SSM expands upon only one state in the level above it, starting from the top level Move to ORP state and ending in a lower level state.

Modularization begins at the sub-level. The sub-level consists of five SSMs, one for each of the phases in the top level. Each phase in the top level is its own module (SSM) at the lower level.

In addition to the horizontal and vertical slices, an escape level is also modeled. This level models actions that require the patrol base operations to abort. This is a floating module and can be reached from any state in the model.

5.3 Hierarchy of Secure State Machines

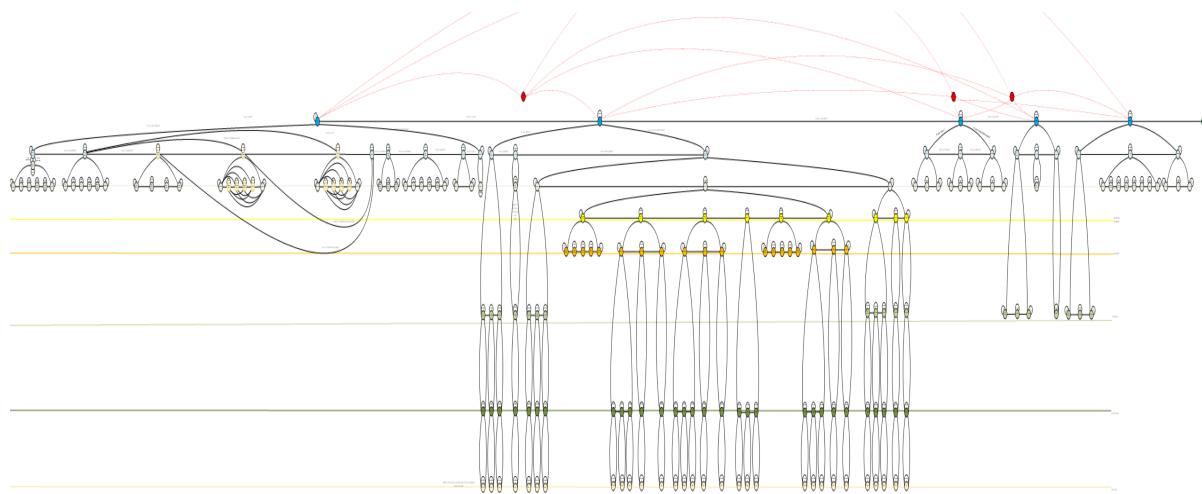


Figure 5.2: Diagrammatic description of patrol base operations as a modularized hierarchy of secure state machines. (Generated by Jesse Nathaniel Hall.)

5.3.1 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).

Levels of The Hierarchy The straight, colored lines that span the diagram in figure 5.2 delineate levels of the hierarchy of SSM. 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.

Each level represents a different degree of abstraction with a set of principal responsible for actions at that level.

- Red—escape level:

Unacceptable losses that require the mission to abort. This is a floating module that can be reached by any level.

- Dark blue—top level:

Describes the patrol base operations as five phases (plus an end phase). The Platoon Leader (PL) is responsible at this level.

- Blue grey—sub-level:

This level takes the previous level's states to a lower level of abstraction. The Platoon Leader and Platoon Sergeant (PSG) are responsible for this level.

- Grey green—sub-sub-level:

This level takes the previous level's states to a lower level of abstraction. The responsible principals are the patrol base headquarters: Platoon Leader, Platoon Sergeant, Radio Telephone Operator (RTO), Forward Observer (FO), Medic, and Heavy Weapons Squad Leader (HWSQL).

- Bright yellow—4th level:

This level takes the previous level's states to a lower level of abstraction. The responsible principals are the squad leaders (SQL).

- Orange–5th level:

This level takes the previous level’s states to a lower level of abstraction. The responsible principals are the Team Leaders (TL).

- Olive–6th level:

This level takes the previous level’s states to a lower level of abstraction. The responsible principals are the Squads. That is, the squad behavior is described at this level and the squad indicates readiness.

- Dark green–7th level:

This level takes the previous level’s states to a lower level of abstraction. The responsible principals are the Fire Team (FT), Recon Team (RT), Buddy Team (BT) and Security Team (ST).

- Beige–8th level:

This level takes the previous level’s states to a lower level of abstraction. The responsible principals are the individual soldiers themselves.

The diagram is only a partial model of the patrol base operations. The horizontal slice is the second to last row that expands across the diagram. The vertical slice is in the middle of the diagram and expands several modules to the lowest level of abstraction.

States and Transitions in The Hierarchy The small, colored dots in figure 5.2 represent states (phases) of the patrol base operations. The labels for these states are not readable in this diagram because of limited space, but each dot contains a description of the state. The dots are color coded, with a different color for dots (states) at each level.

The lines connecting one dot to another represent transition requests. These lines are annotated, but are also unreadable in this diagram.

The following sections describe modules from this diagram. The descriptions of the states and transitions are discussed and a readable diagram for each model is provided.

5.3.2 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.² 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*.

There are two approaches to the policy. The first allows transition within each module with no regard to completion of lower level modules. For example, transition from the planning phase to the move to ORP phase in the top level does not require any specific information about completion of the planning module in the level below.

The second approach requires confirmation of completion of the lower level module before transitions at the top level are allowed. For example, before transitioning from the planning phase to the move to ORP phase, the planning phase must confirm that

²The patrol base operations are modeled as sequential activities. However, not all operations follow a definite sequential pattern. The planning phase discusses non-linear transitions. Linear sequences are modeled because it is easier to work with given the complexity. But, linear sequences are not necessary.

the planning module is complete. This isn't necessarily true of the real patrol base operations. For example, the patrol may move to the ORP before receiving its mission. But, modeling it this way demonstrates integration modules at different levels of the hierarchy.

For this second approach, an "all knowing" principal (essentially a signal relay) communicates when a lower level module is complete. This principal is called OMNI. OMNI allows for encapsulation of each module by communicating information about one module to another and eliminating the need for principals in one module to "know" what is happening in another module. This is necessary to reason with the ACL in HOL. It is a feature of the model and not the operations.

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. In SSMs that integrate modules, this could be thought of as feedback from OMNI.

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

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 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.

5.3.3 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 SSM, 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 SSM is complete.

5.3.4 Escape

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

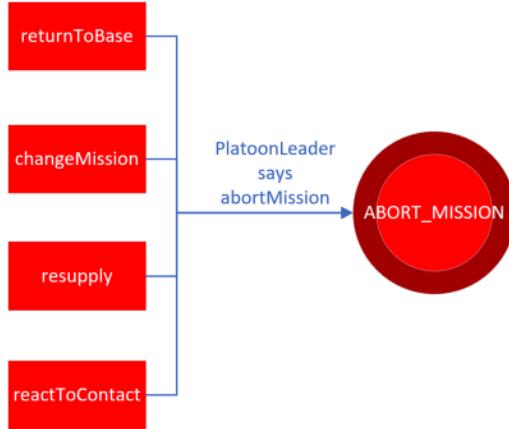


Figure 5.3: Escape level diagram.

The square boxes in the diagram represent communication from outside the module. They represent external signals from OMNI. The only state is the ABORT_MISSION state.

The abortable conditions are *returnToBase*, *changeMission*, *resupply*, and *reactToContact*.³

The security policy for the escape level contains the clauses *OMNI controls returnToBase*, *OMNI controls changeMission*, etc. Given a request *OMNI says returnToBase* and the security policy, *returnToBase* is believed.

The security policy also contains a clause that allows the Platoon Leader to transition to the ABORT_MISSION state if *returnToBase* (*changeMission*, etc.) is believed and the Platoon Leader requests the transition.

³React to contact in this description refers to unwanted contact that requires the mission to be aborted. It does not refer to planned contact in combat missions or contact that does not require mission abortion.

5.3.5 Top Level

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

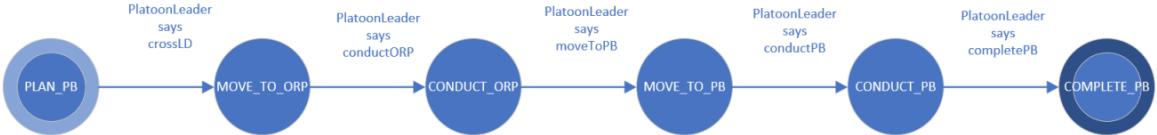


Figure 5.4: Top level diagram.

This SSM represents the phases shown in figure 5.1. There are six states: PLAN_PB, MOVE_TO_ORP, CONDUCT_ORP, MOVE_TO_PB, CONDUCT_PB, and COMPLETE_PB (an end state).

Commands to transition from one state to another are named after the next state. For example, to transition from MOVE_TO_ORP to CONDUCT_ORP, the command is conductORP. The transition from the PLAN_PB state to the MOVE_TO_ORP state is the only exception. This command is crossLD.

Transitions are initiated by a request from and authenticated and authorized principal. These requests have the form *PlatoonLeader says crossLD*.

This SSM integrates with the lower level modules. This means that transitions are allowed when the appropriate lower level module is complete. This requires the aid of the OMNI principal.

The Platoon Leader is the only authenticated principal at this level. Authentication is assumed to be visual or other recognition (i.e., does not require anything complicated such as a cryptographic key.)

The security policy is state dependent. It requires a signal from OMNI that the lower

level module is complete. Once this is believed, the Platoon Leader is authorized to make a transition to the next state.

5.3.6 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. These SSMs are described next.

5.3.6.1 ssmPlanPB

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

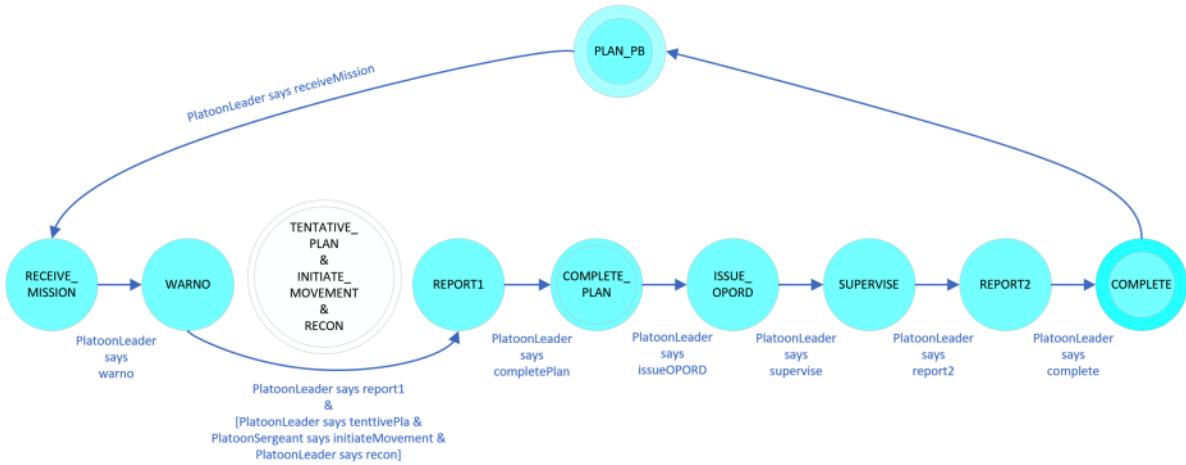


Figure 5.5: Horizontal slice: PlanPB diagram.

This is the largest SSM and models the eight steps of the troop leading procedures.

There are states: RECEIVE_MISSION, WARNO, TENTATIVE_PLAN, INITIATE_MOVEMENT, RECON, REPORT1, COMPLETE_PLAN, ISSUE_OPORD, SUPERVISE, REPORT2, and COMPLETE.

Transition commands follow the same pattern as in the other SSMs. The exception is discussed below.

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 order. 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. The security policy enforces this transition by including the clause *tentativePlan andf initiateMovement andf recon impf PlatoonLeader controls report1*.

ssmPlanPB has two principals: PlatoonLeader and PlatoonSergeant. Only the PlatoonLeader is authorized to make transitions among states. The PlatoonSergeant controls the InitiateMovement command (but, it is not a state and therefore there is no transition).

This SSM does not integrate and thus does not require the OMNI principal. The security policy authorizes the Platoon Leader to make transitions as described in previous sections, with the exception as described above.

5.3.6.2 ssmMoveToORP

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

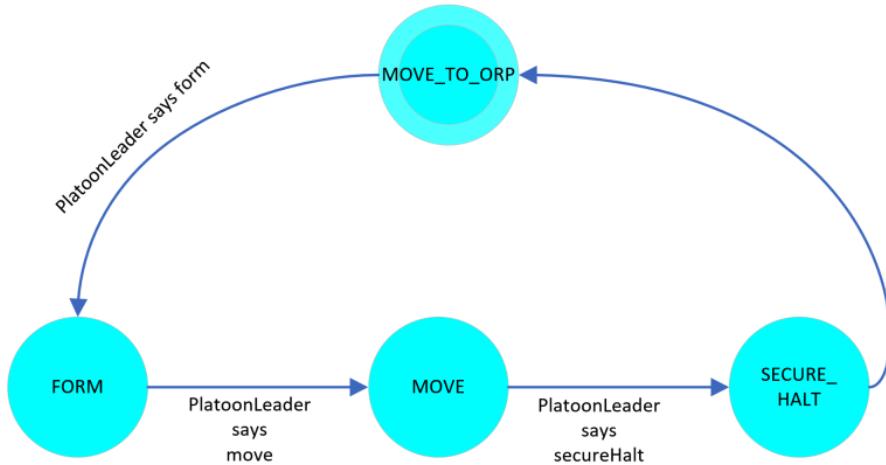


Figure 5.6: Horizontal slice: MoveToORP diagram.

This SSM has three states: FORM, MOVE, and SECURE_HALTI.

The Platoon Leader is the only authorized principal.

This is an integrating SSM, and requires a signal from OMNI that the lower level SSM is complete.

The security policy allows for transitions if requested to do so from the Platoon Leader.

5.3.6.3 ssmConductORP

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

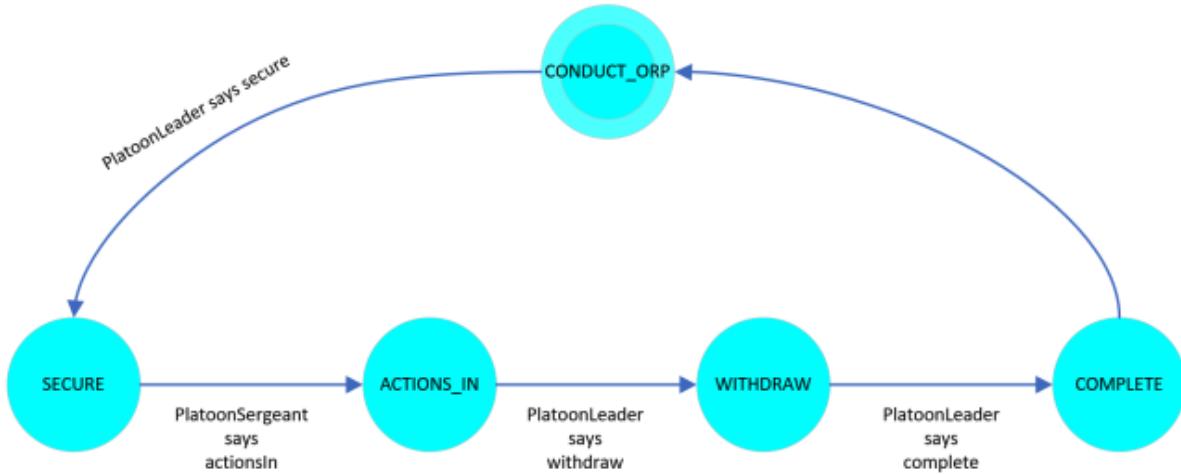


Figure 5.7: Horizontal slice: ConductORP diagram.

This SSM has four states: SECURE, ACTIONS_IN, WITHDRAW, and COMPLETE.

There are two principals. The Platoon Leader is authorized on all commands except for the actionsIn command. The PlatoonSergeant is only authorized on the actionsIn command.

In contrast to ssmMoveToORP, this is not an integrating SSM, therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader or the PlatoonSergeant.

5.3.6.4 ssmMoveToPB

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

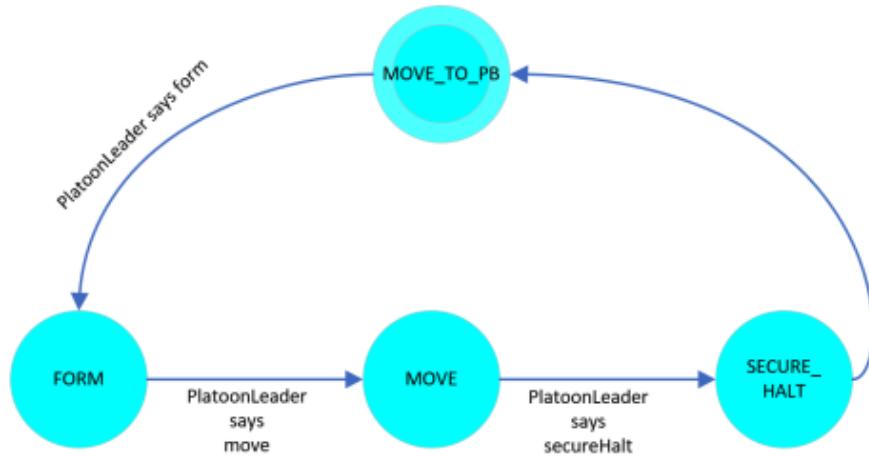


Figure 5.8: Horizontal slice: MoveToPB diagram.

This SSM is the same as ssmMoveToORP. It has three states: FORM, MOVE, and SECURE_HALTI.

The Platoon Leader is the only authorized principal.

In contrast to ssmMoveToORP, this is not an integrating SSM, therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.

5.3.6.5 ssmConductPB

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

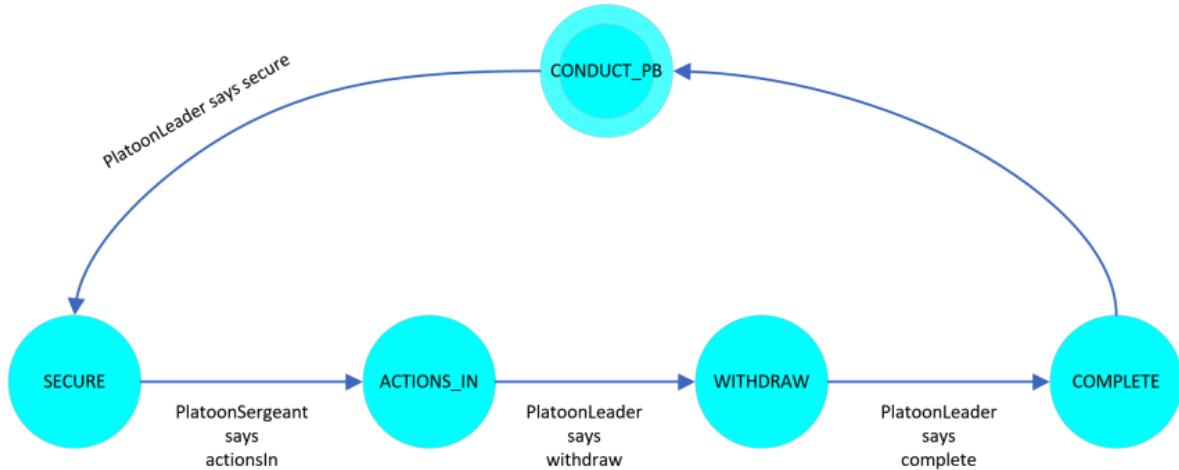


Figure 5.9: Horizontal slice: ConductPB diagram.

This SSM is the same as ssmConductORP. It has four states: SECURE, ACTIONS_IN, WITHDRAW, and COMPLETE.

There are two principals. The Platoon Leader is authorized on all commands except for the actionsIn command. The PlatoonSergeant is only authorized on the actionsIn command.

In contrast to ssmMoveToORP, this is not an integrating SSM, therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader or the PlatoonSergeant.

5.3.7 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. In this sub level, the state SECURE_HALT is expanded into the ssmSecureHalt SSM. In this sub-sub-level SSM, the state ORP_RECON is expanded into the ssmORPRecon SSM. In this SSM, the state MOVE_TO_ORP is expanded into the ssmMoveToORP4L SSM. In this SSM, the state FORM_RT is expanded into the ssmFormRT SSM.

5.3.7.1 ssmSecureHalt

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

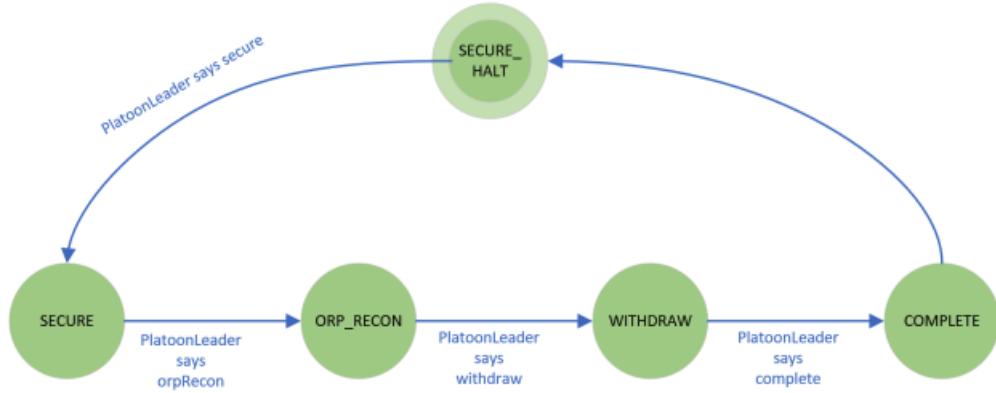


Figure 5.10: Vertical slice: SecureHalt diagram.

This SSM has four states: SECURE, ORP_RECON, WITHDRAW, and COMPLETE.

The Platoon Leader is the only authorized principal.

This is not an integrating SSM. Therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.

5.3.7.2 ssmORPRecon

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

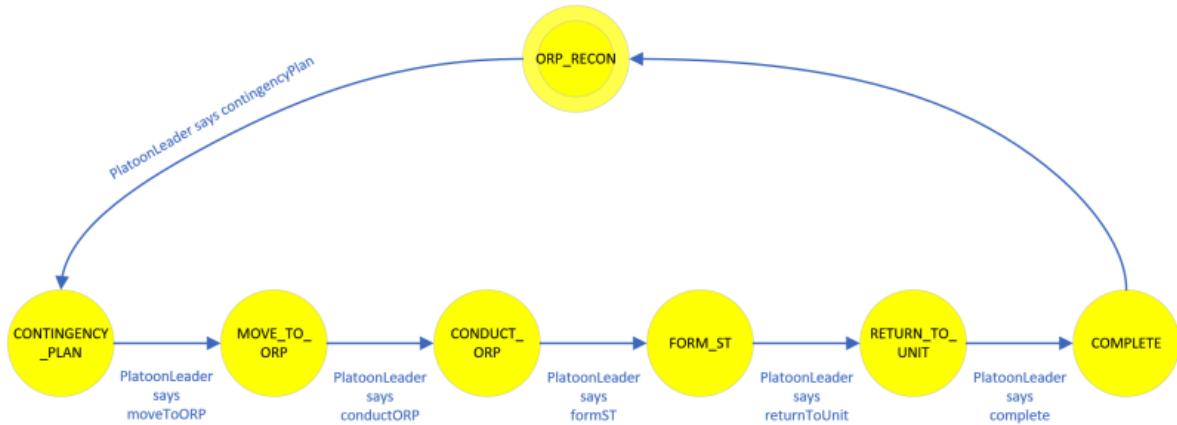


Figure 5.11: Vertical slice: ORPRecon diagram.

This SSM has six states: CONTINGENCY_PLAN, MOVE_TO_ORP, CONDUCT_ORP, FORM_ST, RETURN_TO_UNIT, and COMPLETE.

The Platoon Leader is the only authorized principal.

This is not an integrating SSM. Therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.

5.3.7.3 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.

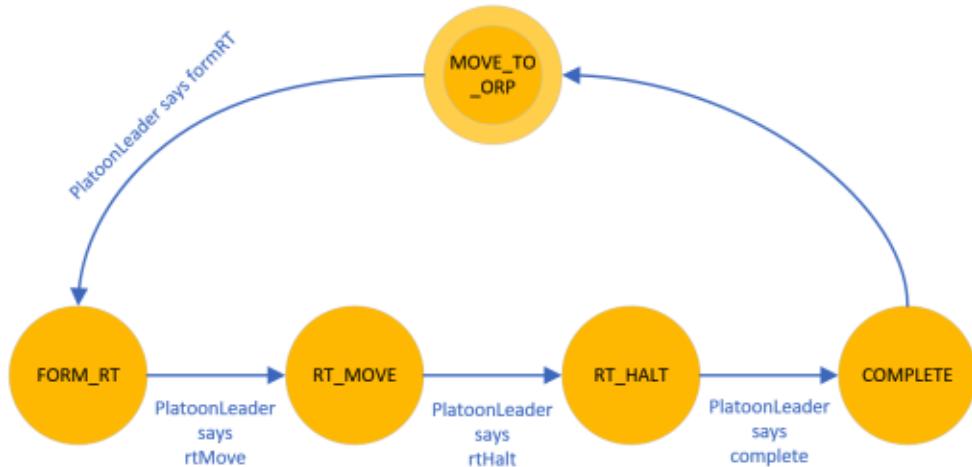


Figure 5.12: Vertical slice: MoveToORP4L diagram.

This SSM has four states: FORM_RT, RT_MOVE, RT_HALT, and COMPLETE.

The Platoon Leader is the only authorized principal.

This is not an integrating SSM. Therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.

5.3.7.4 ssmFormRT

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

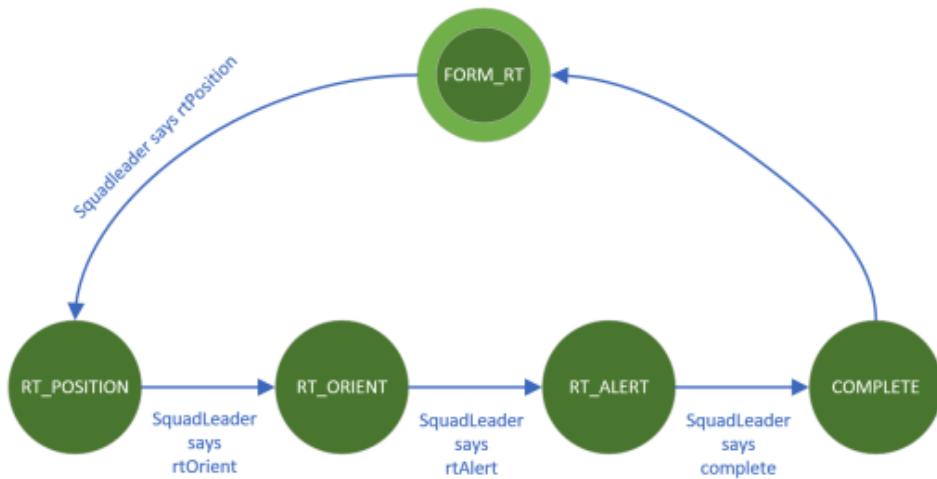


Figure 5.13: Vertical slice: FormRT diagram.

This SSM has four states: RT_POSITION, RT_ORIENT, RT_ALERT, and COMPLETE.

The Platoon Leader is the only authorized principal.

This is not an integrating SSM. Therefore there is no OMNI principal.

The security policy allows for transitions if requested to do so from the Platoon Leader.

The next chapter describes secure state machines in detail and discusses their implementation in HOL.

Chapter 6

Secure State Machine Model

This chapter introduces the reader to secure state machines (SSMs). It begins with a description of state machines and then describes *secure* state machines. It concludes with a description of the HOL implementation of the SSM. After reading this chapter, the reader should be prepared to follow the description of the patrol base operations implementation in HOL (described in the next chapter).

6.1 State Machines

State machines are models of systems. They use *states* and *transitions* among states to describe the system's behavior. The result is a well-defined set of system states and system behaviors that are readily automizable.

There are several models of state machines.¹ The state machine model described in this master thesis changes states and outputs based on input. An input will cause the state machine to change states and produce an output.

¹A discussion of these models is beyond the scope of this master thesis.

6.1.1 States

States can be nearly anything. For example, the state for a military base could be described by the number of foreign civilians on base. If a foreign civilian is permitted to enter the base, then the state of the base increases from n foreign civilians to $n + 1$. Or, the state of that foreign civilian requesting access to a base could be "not granted" or "granted."

States, in this master thesis, 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 what is referred to as the top level secure state machine.

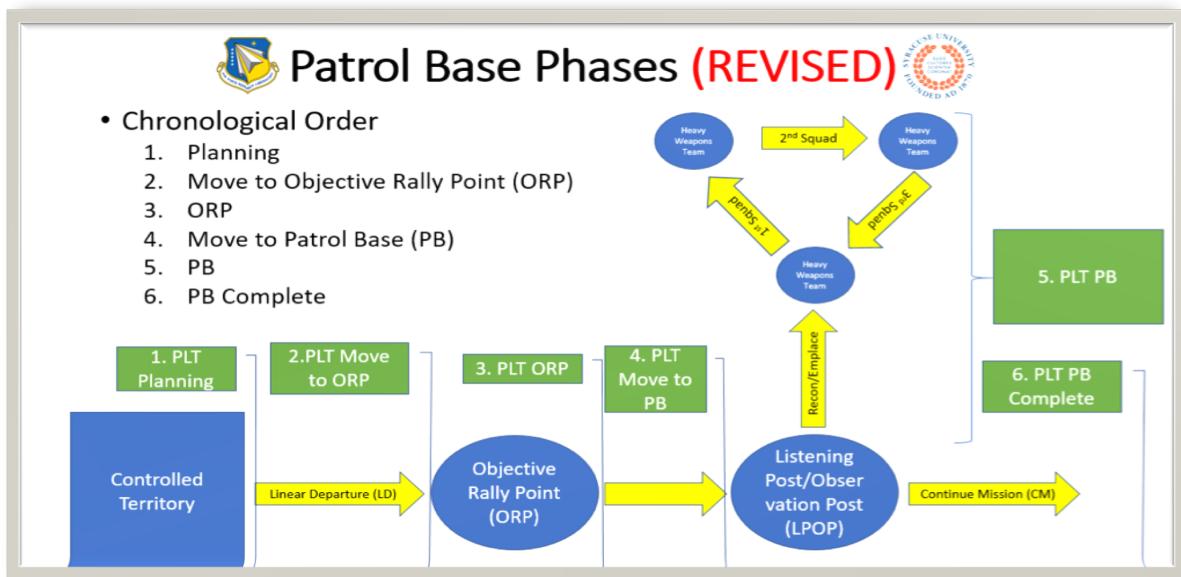


Figure 6.1: A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel Hall.

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 the n *foreign civilians* state to the $n + 1$ *foreign civilians* state could be "access granted." This same input could be used to change a foreign civilian's state from "not granted" to "granted."

In this master thesis, inputs are commands to transition from one state to another. 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 (*NS*) describes the next state of the state machine given the current state and input. In HOL (and functional programming languages in general), the parameters follow the name of the function.

$$NS \ CURRENT_STATE \ input$$

- . The function is defined with it's result.

$$NS \ CURRENT_STATE \ input = NEXT_STATE$$

For example, the next-state function to change from the n *foreign civilians* state to the $n + 1$ *foreign civilians* state using the "accessGranted" input would look like this:

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

Similarly, for the foreign civilian to change from the *Not Granted* state to the *Granted* state:

$$NS \ NOT_GRANTED \ accessGranted = GRANTED$$

In this master thesis, to transition from the *MOVE_TO_ORP* state to the *CONDUCT_ORP* state using the command "conductORP" looks like this:

$$NS \text{ } MOVE_TO_ORP \text{ } conductORP = CONDUCT_ORP$$

6.1.4 Next-output Function

The next-output function (*NOut*) describes the next output of the state machine given the current state and input. This is defined similarly to the next-state function.

$$NOut \text{ } CURRENT_STATE \text{ } input = NextOutput$$

For example, the next-output function to change from the *n foreign civilians* state to the *n + 1 foreign civilians* state using the "accessGranted" input would look like this:

$$NOut \text{ } N \text{ } accessGranted = BasePopulationIncreasedByOne$$

Similarly, for the foreign civilian to change from the *Not Granted* state to the *Granted* state::

$$NOut \text{ } NOT_GRANTED \text{ } accessGranted = NowOnBase$$

In this master thesis, the next output for the transition from the *MOVE_TO_ORP* state to the *CONDUCT_ORP* state using the command "conductORP" looks like this:

$$NOut \text{ } MOVE_TO_ORP \text{ } conductORP = ConductORP$$

6.1.5 Configuration

A configuration describes the state machine using input and output streams [7]. 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 behavior.

6.1.6 TR Relations

Transition relations (TRs) define the behavior of the state machine based on its input, configuration, and the next configuration. A TR takes an input, an initial configuration, and a final configuration. In terms of logic, a TR is a propositional statement that is either true or false. If the final configuration follows from the initial configuration and the input, then the TR is true. Otherwise, it is false. The trueness or falsity of the TR follows from the next-state and next-output functions. The pattern looks like this:

TR *input*

CFG

input::*inputList*

CURRENT_STATE

outList

CFG

inputList

(*NS* *input* *CURRENT_STATE*)

(*NOut* *input* *CURRENT_STATE*)::*outList*

CFG is a type constructor² for a configuration. What follows are the three components of the *CFG* data type. The first is the input stream, a list of inputs, denoted

²see appendix A.3 for a description of types in HOL.

input::*inputList*.³ The first component of this list is the same as the input to the TR.

The next component of the configuration is the current state, denoted

CURRENT_STATE. The third component is the output list, denoted *outList*.⁴

The second *CFG* is also composed of three components. The first is the input list with the first element removed (because we used this element as input to the TR). The next component is the state which results from applying the *input* and *CURRENT_STATE* to the next-state function. The third component is the output which results from applying the *input* and *CURRENT_STATE* to the next-state function. This result is "consed" onto *outList*.⁵

The input stream in the first configuration is a list (denoted as *headOfList*::*remainderOfList*). 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*.

TRs form the basis for the HOL representation of state machines, which is a precursor to the secure state machines implemented in this master thesis.

³In functional programming languages, functions are applied to list elements one element at a time. The notation *headOfList*::*remainderOfList* allows access to the first component of the list.

⁴Note that only the input (not the output) is needed for the TR to evaluate to true or false. Therefore, the output is not denoted as *headOfList*::*remainderOfList*.

⁵"Consed" is functional programming language jargon for "appended to the front of the list."

6.2 Secure State Machines

The secure state machine (SSM) adds security to the state machine model. It implements complete mediation using a monitor that verifies the authentication and authorization of any request (input) to transition from one state (or configuration) to another.

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.

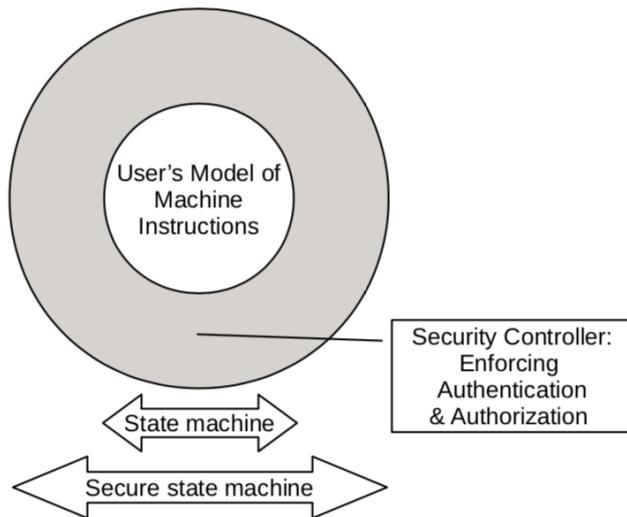


Figure 6.2: State machine versus secure state machine with a monitor. Image taken from *Certified Security by Design Using Higher Order Logic* [7].

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 requests to access or change the state of the SSMs. In a real world analogy, the monitor is the sentry at the gate who checks IDs to verify a person's identity and checks with policy or procedures to determine if that person should be granted access.

6.2.3 Transition Types

The monitor assigns a transition type to each command. The transition types are: execute (*exec*), trap, (*trap*), and discard (*discard*).⁶ These correspond to executing a command, trapping a command, or discarding a command, respectively. For example, *exec accessGranted* indicates that the command to grant access for the foreign civilian to the base should be executed (or allowed). *trap accessGranted* means that the request to grant access is not authorized. *discard grantAccess* means that the command to grant access is not authenticated (i.e., the requestor's identity is not confirmed).

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

⁶The names are derived from their use in virtual machines. Commands (inputs) in virtual machines are either executed, trapped, or discarded. Each has a different behavior in the machine.

to each transition type and command (request).

6.2.5 Principals And Requests

Principals are unique to the secure state machine. The SSM monitor checks authentication and authorization for each request. This means the identity of "who" is making the request must be first verified and then authorized to control actions on the SSM. Therefore, a "who" must be defined. The "who" in the access-control logic is the principal.

Transitions in the SSM are requested by a principal as *Principal says changeStates*. The next example shows how requests are made in a SSM for the top level patrol base operations.

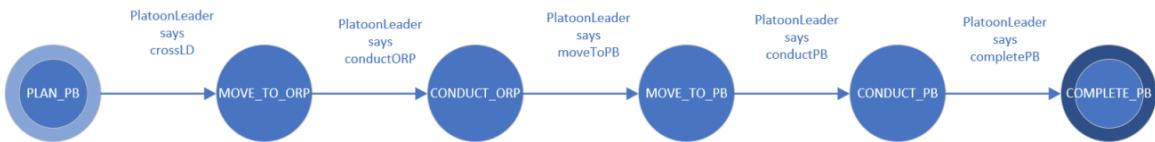


Figure 6.3: Top level diagram.

Figure 7.1 shows the top level SSM. The transition from PLAN_PB to MOVE_TO_ORP in a state machine only requires the command *crossLD*. But, in the secure state machine, this transition request looks like this:

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

exec crossLD

This is passed to the next-state and next-output functions, which define the behavior for the command *exec crossLD*. Note that the next-state and next-output functions must also define the SSM behavior for *trap crossLD* and *discard crossLD*.

6.2.6 Authentication

Authentication refers to verification of identity.⁷ Authentication is verified by the monitor for each request.

In this master thesis, authentication is verified by visual confirmation of a principal's identity. Either the monitor recognizes the principal making the request or not.⁸ The implementation of this in HOL authenticates principals on particular commands (or sets of commands). Thus, the monitor authenticates the request *SomePrincipal says someCommand* rather than just the *SomePrincipal*.

6.2.7 Authorization

Authorization controls access-rights by way of a security policy. Authorization in the SSM is defined by two functions. One function describes access-control rules that are dependent on state and input. The other function describes access-control rules that are dependent on only input only.

Rules are of the form *SomePrincipal controls someCommand*. or *if (state =*

⁷Authentication is a very broad topic and a thorough discussion is beyond the scope of this master thesis.

⁸SSMs also work with cryptographic functions. There already exists cryptographic versions of the SSM described in this and the next section.

SOME_STATE then SomePrincipal controls someCommand.

With authentication and authorization rules defined, the SSM monitor can use the *Controls* rule described in section 4.2.6 of chapter 4. Thus, the monitor reasons about a request and a policy as such:

SomePrincipal says someCommand

and

SomePrincipal controls someCommand

implies

someCommand

More complicated statements that reduce to this are also allowed.

6.2.8 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*.

6.2.9 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.

6.2.10 TR Relations

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

TR (trType input)

CFG

authenticationTest

stateInterpretation

securityContext

input::inputList

CURRENT_STATE

outList

CFG

authenticationTest

stateInterpretation

securityContext

inputList

(NS (trType input) CURRENT_STATE)

(NOut (trType input) CURRENT_STATE)::output

In addition to the three additional components (authenticationTest, stateInterpretation, and securityContext functions), the transition type (trType) is added to the front of the input.

6.2.11 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. The configuration interpretation function *ConfigInterp* interprets the authentication and authorization of the request based on the current configuration.

ConfigInterp

(M, O_i, O_s)

CFG

AuthenticationTest *inputOK*

StateInterpretation

SecurityContext

input::inputList

CURRENT_STATE

outList

\iff

(M, O_i, O_s) satisfies (*securityContext*, *input*, and *stateInterpretation*)

Like TR, this is higher order logic predicate that is either true or false. It is a biconditional (if and only if). If the first part of the bioconditional follows from the second and vis-a-vis, then the *ConfigInterp* evaluates to true. Otherwise, it evaluates to false. It also requires use of the Kripke structure because it uses the ACL inference

rules to evaluate.

The first part of the bioconditional takes the current *CFG*. Note that in this *CFG* the input is in the form of a request (not *trType* command).

The second part of the biconditional is the "satisfies list." This is a conjunction of the "satisfies" property applied to the *securityContext*, *input*, and *StateInterpretation* function. The *SecurityContext* and *StateInterpretation* functions are user defined and specific for each state machine.

6.3 Secure State Machines in HOL

This following sections describe the HOL implementation of the SSM. It focuses on the parametrizable SSM that forms the basis for the patrol base operations SSMs described in the next chapter. The parametrizable SSM is denoted as "ssm".

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

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 $P \text{ says } prop \text{ (SOME cmd)}$.⁹ 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 takes one input of the form $P \text{ says } prop \text{ (SOME cmd)}$ and extracts the cmd part.

[`extractCommand_def`]

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

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.

$\vdash \forall x.$

$\text{commandList } x = \text{MAP extractCommand } x$

⁹See next section.

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

[`extractPropCommand_def`]

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

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 $\text{prop } (\text{SOME } cmd)$ elements.

$\vdash \forall x.$

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

extractInput takes one input of the form $P \text{ says prop } x$ and extracts the x part.

Note that x can have two forms: *NONE* or *SOME cmd*.

[`extractInput_def`]

```
⊢ extractInput (P says prop x) = x
```

inputList takes an input list consisting of list elements of the form $P \text{ says prop } x$. 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 *SOME command* or *NONE*.

For example, *P says prop (SOME cmd)* or *P says prop NONE*. Option types are common in functional programming languages because a function must return something. The *NONE* option allows for that something to be nothing.

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 SSM. 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 constructors¹⁰: *ESCC* and *SLC*. The second part is the name of the datatype variable¹¹ or datatype.

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

```
escCommand = returnToBase
```

```
    | changeMission
```

¹⁰see the background section A.3

¹¹Both of these are datatype variables because they define other datatypes.

```
| 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.

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. These are not data type variables, they are datatypes, because they are not further defined.

```
plCommand = crossLD  
| conductORP
```

```

| moveToPB
| conductPB
| completePB
| incomplete

```

omniCommand refers to commands that the OMNI level principal¹² is authorized to make.

```

omniCommand = ssmPlanPBComplete
| 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.

¹²See section 5.3.3 for a discussion of the OMNI level principal.

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* rather than *((plCommand slCommand) command)Option*, for example. This is because most of the authentication and authorization definitions require a type *slCommand*, which includes both *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.

$$trType = \text{discard } 'cmdlist \mid \text{trap } 'cmdlist \mid \text{exec } 'cmdlist$$

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 with the later of the pair in the form of a list

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

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. The transition from the WARNO to the REPORT1 state in ssmPlanPB requires four commands.

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

$$SOME \ (SLc \ (PL \ crossLD)); SOME \ (SLc \ (PL \ crossLD))]$$

List elements are separated with a semicolon.

6.3.4 Authentication

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

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

This function takes an `elementTest` function and an input list as parameters. The `elementTest` function is named `inputOK` in the patrol base operations SSMs 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 `FOLDRs`¹³ the `elementTest` function over the input list `x` with the conjunction function and `True` as the accumulator. 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 context-dependent. This means that each SSM defines its own security context. The parametrizable SSM allows for two ways to define the security context and

¹³This 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.

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.

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 NOut, respectively.

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

```
(('command option, 'principal, 'd, 'e) Form -> bool)
```

2. state interpretation function.

The type signature is

```
('state -> ('command option, 'principal, 'd, 'e) Form list ->
   ('command option, 'principal, 'd, 'e) Form list)
```

3. security context function.

The type signature is

```
(('command option, 'principal, 'd, 'e) Form list ->  
('command option, 'principal, 'd, 'e) Form list)
```

4. input list stream.

The type signature is

```
(('command option, 'principal, 'd, 'e) Form list list)
```

5. state.

The type signature is

```
'state
```

6. output stream.

The type signature is

```
('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 SSM).

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 takes 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 state interpretation function.

`satList` is a list of elements that satisfy the property of soundness as discussed in chapter 4 section 4.2.5.4¹⁴. `satList` is defined in `satListTheory`.

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

`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 B.

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

$$\vdash \text{CFGInterpret}$$

¹⁴Note that in the ACL implementation of the "satisfies" and "soundness" properties, "satisfies" serves as "soundness" when it is generalized for all Kripke structures.

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

6.3.9 TR Rules

The transition inference rules for execute, trap, and discard are defined in figure 6.4.

SSM behavior is defined inductively by three rules.

$$\begin{array}{l}
\text{Execute} \quad \frac{(\text{authenticationTest elementTest } x) \quad (\text{CFGInterpret } (M, O_i, O_s) \text{ Config})}{\text{Config} \xrightarrow{\text{exec (inputList } x\text{)}} \text{Config}_e} \\
\\
\text{Trap} \quad \frac{(\text{authenticationTest elementTest } x) \quad (\text{CFGInterpret } (M, O_i, O_s) \text{ Config})}{\text{Config} \xrightarrow{\text{trap (inputList } x\text{)}} \text{Config}_t} \\
\\
\text{Discard} \quad \frac{\neg(\text{authenticationTest elementTest } x)}{\text{Config} \xrightarrow{\text{discard (inputList } x\text{)}} \text{Config}_d}
\end{array}$$

where,

$$\begin{aligned}
\text{Config} &= \text{CFG elementTest stateInterp context } (x :: ins) s \text{ outs} \\
\text{Config}_e &= \text{CFG elementTest stateInterp context } ins \\
&\quad (NS s (\text{exec (inputList } x))) (Out s (\text{exec (inputList } x)) :: outs) \\
\text{Config}_t &= \text{CFG elementTest stateInterp context } ins \\
&\quad (NS s (\text{trap (inputList } x))) (Out s (\text{trap (inputList } x)) :: outs) \\
\text{Config}_d &= \text{CFG elementTest stateInterp context } ins \\
&\quad (NS s (\text{discard (inputList } x))) (Out s (\text{discard (inputList } x)) :: outs)
\end{aligned}$$

Figure 6.4: Transition rules in HOL. Image taken from Certified Security by Design Using Higher Order Logic [7]

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

The *Execute* rule takes the authenticationTest function (with two parameters) and the CFGInterpret function (with two parameters). If these are true, then *Execute* concludes the TR $Config \xrightarrow{\text{exec (inputList } x\text{)}} Config_e$. *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 $Config \xrightarrow{\text{discard (inputList } x\text{)}} Config_d$ 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]

```

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

```

rule1 [TRrule1]

```

 $\vdash \text{TR } (M, Oi, Os) \ (\text{trap } (\text{inputList } x))$ 
 $(\text{CFG } elementTest \ stateInterp \ context \ (x :: ins) \ s \ outs)$ 
 $(\text{CFG } elementTest \ stateInterp \ context \ ins$ 

```

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

rule2 [rule2... (same as) TR_discard_cmd_rule]

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

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 of a command.

[TR_exec_cmd_rule]

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

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

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

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

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

TR_trap_cmd_rule is similar to TR_exec_cmd_rule.

[TR_trap_cmd_rule]

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

```


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


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


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


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


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


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


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


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


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


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


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


```

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

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

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

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

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

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

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 modularized hierarchy of secure state machines (SSMs) consists of many SSMs. The code for all of them can be found in the appendices. Appendix D contains the pretty-printed HOL output. Appendix E contains the HOL script (raw) code. These files can also be found in the accompanying files and folders in the folder `MasterThesis/HOL/`.

Many of the SSMs are similar and differ only in the names of the states, commands, and principals. A representative sample of these is described in this chapter. The first SSM, `ssmPB`, is a typical example from the hierarchy that uses the OMNI principal and one other principal. All transitions are sequential. The next example, `ssmConductORP`, uses two principals and the OMNI principal. The third example is `ssmPlanPB`, which contains the non-sequential conjunction of states. It does not use the OMNI principal. All other SSMs fit into one of these three patterns.

7.1 ssmPB: A Typical Example from the Hierarchy

ssmPB is the top level SSM. It is an example of a typical one-principal¹ SSM that also uses the Omni principal. A diagram of ssmPB is shown in figure 7.1.

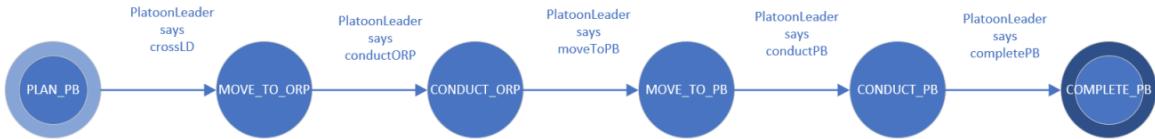


Figure 7.1: Top level diagram.

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 transition request is a list and has the form²

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 form³

stateComplete impf

PlatoonLeader controls moveToNextState

¹For the purposes of transitions, the Omni principal isn't thought of as a principal, but a relay. However, for the purposes of defining principals in HOL, Omni is included as a principal.

²*stateComplete* and *moveToNextState* are place holders for the actual commands which are defined later.

³*impf* is the ACL-HOL equivalent of "implies"

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

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

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 OMNITYrpeScript.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
```

unAuthorized and *unAuthenticated* pertain to *trap* and *discard* transition types,

respectively.

7.1.4 Commands

OMNITyprScript.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]

```

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

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

The next-state function for *ssmPB* uses pattern matching and if-then-else statements. The function first matches the state. Then, it uses the *getPlCom* function to extract the *plCommand* from the command list *x*. It compares this command to the command on the right. If they are equal, then the transition occurs. Otherwise, the state does not change.

For the *trap* and *discard* transition types (last two lines in the definition), no transition occurs regardless of the command. Therefore, the command is not checked. The

original state s is returned.

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 output. The *trap* and *discard* transition types return *unAuthorized* and *unAuthenticated*, respectively, for any command list x .

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*⁴ in the SSMs.

In HOL, *inputOK* uses the wild card denoted by an underscore "`_`". This causes HOL to generate a lot of additional code when it evaluates the definition. Both the HOL definition and the HOL-generated output for this function are 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 Omni) says prop (cmd:( (slCommand command) option )))  
:(( slCommand command) option , stateRole , 'd , 'e)Form) = T) /\  
(inputOK _ = F)`
```

The HOL definition for *inputOK* uses pattern matching. The first call to *inputOK* matches the input to

$((Name \text{PlatoonLeader}) \text{ says } prop \text{ (cmd:((slCommand command)option))).}$

If it matches, the function returns T for true. The second call to *inputOK* matches the input to

$((Name \text{Omni}) \text{ says } prop \text{ (cmd:((slCommand command)option))).}$

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`]

⁴The function *elementTest* should be thought of as a parameter where *inputOK* specializes that parameter.

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

$$\begin{aligned}
 (\text{inputOK } (v_{25} \text{ eqs } v_{26}) \iff F) \wedge (\text{inputOK } (v_{27} \text{ eqn } v_{28}) \iff F) \wedge \\
 (\text{inputOK } (v_{29} \text{ lte } v_{30}) \iff F) \wedge (\text{inputOK } (v_{31} \text{ lt } v_{32}) \iff F)
 \end{aligned}$$

It is straight forward to prove that any command that is not issued by a principal is rejected. This follows 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 authorization in the parametrizable ssm. One is state-dependent, the other is not.

State-dependent Authorization In ssmPB, the state-dependent authorization 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`]

$$\begin{aligned}
 \vdash (\forall xs. & \\
 & \text{secContext PLAN_PB } xs = \\
 & \quad \text{if getOmniCommand } xs = \text{ssmPlanPBComplete} \text{ then} \\
 & \quad \quad [\text{prop } (\text{SOME } (\text{SLc } (\text{OMNI ssmPlanPBComplete}))) \text{ impf} \\
 & \quad \quad \quad \text{Name PlatoonLeader controls} \\
 & \quad \quad \quad \text{prop } (\text{SOME } (\text{SLc } (\text{PL crossLD})))] \\
 & \quad \text{else } [\text{prop NONE}]) \wedge \\
 & (\forall xs. &
 \end{aligned}$$

```

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]) ∧
(∀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 (*impf*) of the form described in

section 7.1.

HOL Definition for getOmniCommand *getOmniCommand* also uses the wildcard "_". 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. This 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*. Thus, if no valid command is present then *invalidOmniCommand* is returned, otherwise the valid *cmd* is returned.

State-independent Authorization The state-independent authorization function *secAuthorization* 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](#)]

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

[[secHelper_def](#)]

```

 $\vdash \forall cmd .$ 
  secHelper cmd =
    [Name Omni controls prop (SOME (SLC (OMNI cmd)) ) ]

```

7.1.9 Proved Theorems

These theorems with their helper lemmas prove the property of complete mediation for each transition in the SSM.

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 first example is described in detail and should be used as a reference for other proofs.

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*]

```

 $\vdash \forall elementTest context stateInterp x ins s outs .$ 
  ( $\forall M Oi Os .$ 

```

```

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

```

thPlanPB 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 *x*
- an output list: substitutes for *outs*

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 deconstructed 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 (in brackets). REWRITE_TAC⁵ rewrites the lemma using the appropriate definition shown in brackets. The definitions in brackets are defined earlier in this and other chapters. 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 ACL in chapter 4 section 4.3.

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

[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)))]
```

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))
```

⁵See <https://hol-theorem-prover.org/kananaskis-11-helpdocs/help/HOLindex.html> for details on HOL tactics and functions.

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*.

```
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]

```

 $\vdash \forall 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]

```

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

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

 $(\text{exec}$ 
 $\quad [\text{SOME}\ (\text{SLc}\ (\text{OMNI}\ \text{ssmPlanPBComplete}));$ 
 $\quad \text{SOME}\ (\text{SLc}\ (\text{PL}\ \text{crossLD}))])$ 

 $(\text{CFG}\ \text{inputOK}\ \text{secContext}\ \text{secAuthorization}$ 
 $\quad ([\text{Name}\ \text{Omni}\ \text{says}$ 
 $\quad \text{prop}\ (\text{SOME}\ (\text{SLc}\ (\text{OMNI}\ \text{ssmPlanPBComplete})));$ 
 $\quad \text{Name}\ \text{PlatoonLeader}\ \text{says}$ 
 $\quad \text{prop}\ (\text{SOME}\ (\text{SLc}\ (\text{PL}\ \text{crossLD}))))] :: ins)\ \text{PLAN\_PB}\ outs)$ 
 $(\text{CFG}\ \text{inputOK}\ \text{secContext}\ \text{secAuthorization}\ ins$ 
 $\quad (NS\ \text{PLAN\_PB}$ 
 $\quad (\text{exec}$ 
 $\quad [\text{SOME}\ (\text{SLc}\ (\text{OMNI}\ \text{ssmPlanPBComplete}));$ 
 $\quad \text{SOME}\ (\text{SLc}\ (\text{PL}\ \text{crossLD}))))$ 

 $(Out\ \text{PLAN\_PB}$ 
 $\quad (\text{exec}$ 
 $\quad [\text{SOME}\ (\text{SLc}\ (\text{OMNI}\ \text{ssmPlanPBComplete}));$ 
 $\quad \text{SOME}\ (\text{SLc}\ (\text{PL}\ \text{crossLD}))))] :: outs) \iff$ 
 $\text{authenticationTest}\ \text{inputOK}$ 
 $\quad [\text{Name}\ \text{Omni}\ \text{says}$ 
 $\quad \text{prop}\ (\text{SOME}\ (\text{SLc}\ (\text{OMNI}\ \text{ssmPlanPBComplete})));$ 
 $\quad \text{Name}\ \text{PlatoonLeader}\ \text{says}$ 
 $\quad \text{prop}\ (\text{SOME}\ (\text{SLc}\ (\text{PL}\ \text{crossLD}))))] \wedge$ 
 $\text{CFGInterpret}\ (M, Oi, Os)$ 
 $\quad (\text{CFG}\ \text{inputOK}\ \text{secContext}\ \text{secAuthorization}$ 
 $\quad (\text{[Name}\ \text{Omni}\ \text{says}$ 
 $\quad \text{prop}\ (\text{SOME}\ (\text{SLc}\ (\text{OMNI}\ \text{ssmPlanPBComplete})));$ 

```

```

Name PlatoonLeader says

prop (SOME (SLc (PL crossLD))) ] :: ins) PLAN_PB

outs) ∧

(M, Oi, Os) satList

[prop (SOME (SLc (OMNI ssmPlanPBComplete)));;

prop (SOME (SLc (PL crossLD))) ]

```

With few exceptions, all proofs justifying execution of a command in the 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 as pretty-printed HOL-generated output.

[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]

```

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

```

```

[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))  $\iff$ 
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_PLAN_PB_trap_justified_thm]

$$\begin{aligned}
& \vdash \text{omniCommand} \neq \text{ssmPlanPBComplete} \Rightarrow \\
& (s = \text{PLAN_PB}) \Rightarrow \\
& \forall NS \ Out \ M \ Oi \ Os. \\
& \text{TR } (M, Oi, Os) \\
& (\text{trap} \\
& [\text{SOME } (\text{SLc } (\text{OMNI } \text{omniCommand})) ; \\
& \text{SOME } (\text{SLc } (\text{PL } \text{crossLD})))])
\end{aligned}$$

```

(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, 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_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 *s1Commands* and *inputOK* authenticates both principals for *s1Commands*. What this means more generally is that both the PlatoonLeader and Omni are authenticated on any state-level command.

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]
```

```

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

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 showing them here. For completeness, however, they are included in the appendices and file folders as described in the introduction to this chapter.

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.

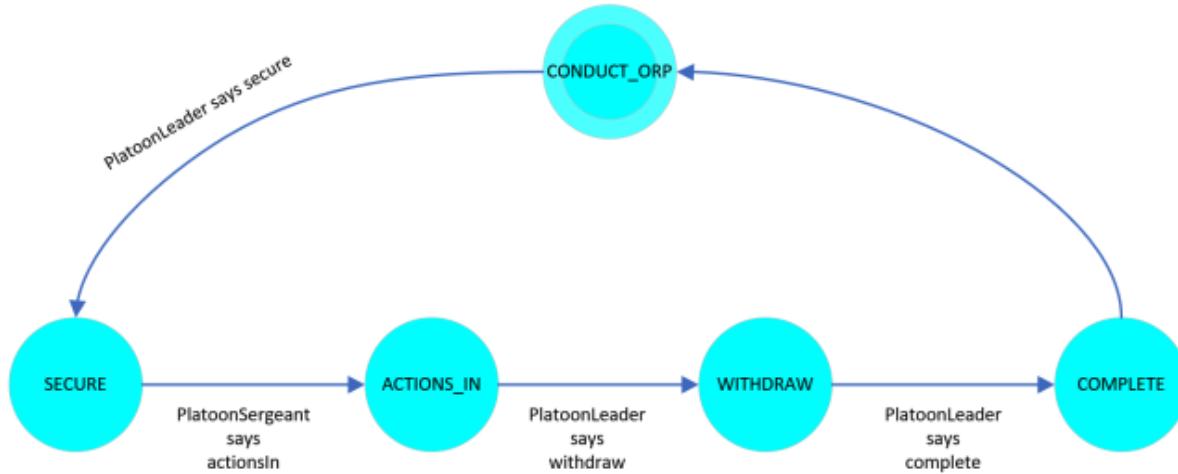


Figure 7.2: Horizontal slice: ConductORP diagram.

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

7.2.1 Principles

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*. This is the transition from SECURE to ACTIONS_IN.

[conductORPNS_def]

```

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

```

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 "`_`" as a wild card. The HOL definition for 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]

```
⊢ (conductORPOut CONDUCT_ORP (exec x) =
  if getPlCom x = secure then Secure else ConductORP) ∧
(conductORPOut SECURE (exec x) =
  if getPsgCom x = actionsIn then ActionsIn else Secure) ∧
(conductORPOut ACTIONS_IN (exec x) =
  if getPlCom x = withdraw then Withdraw else ActionsIn) ∧
(conductORPOut WITHDRAW (exec x) =
  if getPlCom x = complete then Complete else Withdraw) ∧
(conductORPOut s (trap x) = unAuthorized) ∧
(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 an 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 follows 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]

```
 $\vdash (\forall xs.$   

 $\quad \text{secContext CONDUCT\_ORP } xs =$   

 $\quad [\text{Name PlatoonLeader controls}$   

 $\quad \text{prop} (\text{SOME} (\text{SLc} (\text{PL secure}))))]) \wedge$   

 $(\forall xs.$   

 $\quad \text{secContext SECURE } xs =$   

 $\quad \text{if getOmniCommand } xs = \text{ssmSecureComplete} \text{ then}$   

 $\quad \quad [\text{prop} (\text{SOME} (\text{SLc} (\text{OMNI ssmSecureComplete}))) \text{ impf}$   

 $\quad \quad \text{Name PlatoonSergeant controls}$ 
```

```

prop (SOME (SLc (PSG actionsIn))) ]
else [prop NONE] )  $\wedge$ 

( $\forall xs.$ 

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] )  $\wedge$ 

 $\forall xs.$ 

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]

```

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*. It is repeated here as a reference.

[secAuthorization_def]

```
|- ∀ xs. secAuthorization xs = secHelper (getOmniCommand xs)
```

[secHelper_def]

```
|- ∀ 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 of the proof is omitted. The pretty-printed HOL-generated output is included here.

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]

```
|- ∀ 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]

```

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

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

```

```

 $outs) \wedge$ 
 $(M, Oi, Os) \text{ satList}$ 
 $\text{[prop (SOME (SLc (OMNI ssmSecureComplete))) ;}$ 
 $\text{prop (SOME (SLc (PSG actionsIn)))]}$ 

```

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]

```

 $\vdash \forall M \ Oi \ Os.$ 
 $\text{CFGInterpret } (M, Oi, Os)$ 
 $\text{(CFG inputOK secContext secAuthorization}$ 
 $\text{([Name Omni says}$ 
 $\text{prop (SOME (SLc (OMNI ssmActionsIncomplete))) ;}$ 
 $\text{Name PlatoonLeader says}$ 
 $\text{prop (SOME (SLc (PL withdraw))) ] ::ins) ACTIONS\_IN}$ 
 $outs) \Rightarrow$ 
 $(M, Oi, Os) \text{ satList}$ 
 $\text{propCommandList}$ 
 $\text{[Name Omni says}$ 
 $\text{prop (SOME (SLc (OMNI ssmActionsIncomplete))) ;}$ 
 $\text{Name PlatoonLeader says prop (SOME (SLc (PL withdraw))) ]}$ 

```

[PlatoonLeader_ACTIONS_IN_exec_justified_lemma]

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

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

```

(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)) ) ⇔
authenticationTest inputOK

```

```

[Name Omni says
prop (SOME (SLc (OMNI ssmActionsIncomplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))) ] ∧
CFGInterpret ( $M, O_i, O_s$ )
(CFG inputOK secContext secAuthorization
([Name Omni says
prop (SOME (SLc (OMNI ssmActionsIncomplete)));
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))) ] ::ins) ACTIONS_IN
outs) ∧
( $M, O_i, O_s$ ) satList
propCommandList
[Name Omni says
prop (SOME (SLc (OMNI ssmActionsIncomplete)));
Name PlatoonLeader says prop (SOME (SLc (PL withdraw))) ]

```

[PlatoonLeader_ACTIONS_IN_exec_justified_thm]

$$\vdash \forall NS\ Out\ M\ O_i\ O_s.$$

```

TR ( $M, O_i, O_s$ )
(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)
(CFG inputOK secContext secAuthorization ins

```

```

(NS ACTIONS_IN
  (exec
    [SOME (SLc (OMNI ssmActionsIncomplete)) ;
     SOME (SLc (PL withdraw)) ]))

(Out ACTIONS_IN
  (exec
    [SOME (SLc (OMNI ssmActionsIncomplete)) ;
     SOME (SLc (PL withdraw)) ] ) :: outs) )  $\iff$ 
authenticationTest inputOK

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

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

( $M, O_i, O_s$ ) satList
[prop (SOME (SLc (OMNI ssmActionsIncomplete))) ;
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_trap_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]

```

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

```

```

(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) )   $\iff$ 

authenticationTest inputOK

[Name Omni says prop (SOME (SLc (OMNI omniCommand))) ;
Name PlatoonLeader says
prop (SOME (SLc (PL withdraw))) ]  $\wedge$ 

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)  $\wedge$  (M, Oi, Os) sat prop NONE

```

[PlatoonLeader_ACTIONS_IN_trap_justified_thm]

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

```

TR (M, Oi, Os)

(trap

  [SOME (SLc (OMNI omniCommand)) ;
   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

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

  (Out ACTIONS_IN

  (trap

    [SOME (SLc (OMNI omniCommand)) ;
     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

```

As with ssmPlanPB, the other theorems are included in the appendices and files.

7.3 ssmPlanPB: Non-sequential Transitions

The ssmPlanPB SSM does not use the Omni principal but does use a "sort-of" non-sequential progression through states. To minimize the complexity of the model, the three states that are non-sequential are represented as "virtual states" whose completion is noted as input to the command to transition from the state before them to the state after them. Figure 7.3 is a diagram of this SSM.

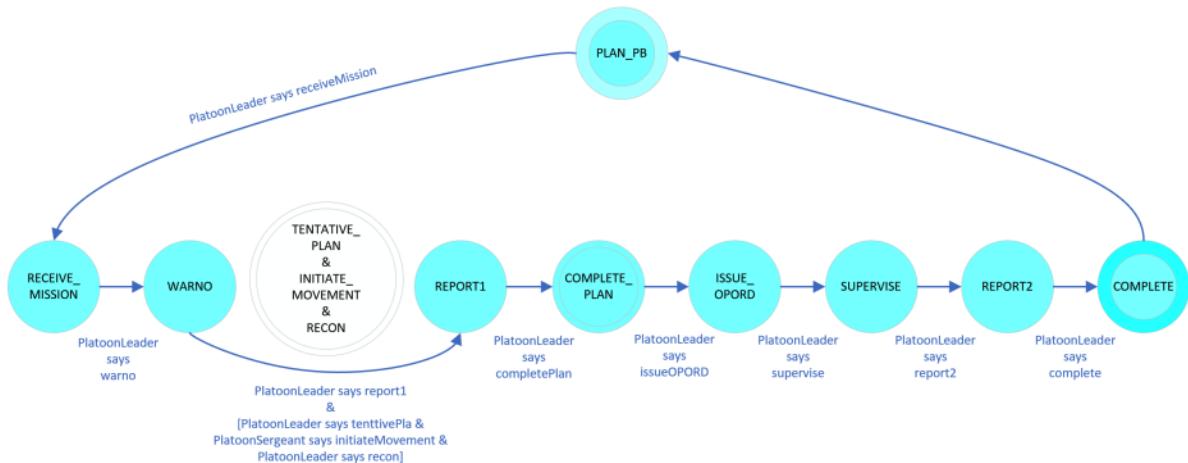


Figure 7.3: Horizontal slice: PlanPB diagram.

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 next state REPORT1.

To handle this, the three non-sequential states are combined into one "virtual state." 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 (a list of inputs) for this transition is

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 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

```

7.3.4 Commands

The *slCommand* datatype for this SSM is defined below.

$$slCommand = PL\ plCommand \mid PSG\ psgCommand$$

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

$$\begin{aligned}
plCommand = & \text{receiveMission} \\
& \mid \text{warno} \\
& \mid \text{tentativePlan} \\
& \mid \text{recon} \\
& \mid \text{report1} \\
& \mid \text{completePlan} \\
& \mid \text{opoid} \\
& \mid \text{supervise} \\
& \mid \text{report2}
\end{aligned}$$

```

| complete
| plIncomplete
| invalidPlCommand

```

```

psgCommand = initiateMovement
| psgIncomplete
| invalidPsgCommand

```

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

7.3.5 Next-State Function

The next-state function is defined similarly to those defined for previously described SSMs.

[planPBNS_def]

```

 $\vdash (\text{planPBNS} \text{ WARNO } (\text{exec } x) =$ 
if
  ( $\text{getRecon } x = [\text{SOME } (\text{SLc } (\text{PL recon}))]) \wedge$ 
  ( $\text{getTenativePlan } x = [\text{SOME } (\text{SLc } (\text{PL tentativePlan}))]) \wedge$ 
  ( $\text{getReport } x = [\text{SOME } (\text{SLc } (\text{PL report1}))]) \wedge$ 
  ( $\text{getInitMove } x = [\text{SOME } (\text{SLc } (\text{PSG initiateMovement}))])$ 
then
  REPORT1
else WARNO)  $\wedge$ 
( $\text{planPBNS PLAN\_PB } (\text{exec } x) =$ 
if  $\text{getPlCom } x = \text{receiveMission}$  then RECEIVE_MISSION

```

```

else PLAN_PB)  $\wedge$ 
(planPBNS RECEIVE_MISSION (exec x) =
 $\text{if}$  getPlCom x = warno then WARNO else RECEIVE_MISSION)  $\wedge$ 
(planPBNS REPORT1 (exec x) =
 $\text{if}$  getPlCom x = completePlan then COMPLETE_PLAN
else REPORT1)  $\wedge$ 
(planPBNS COMPLETE_PLAN (exec x) =
 $\text{if}$  getPlCom x = opoid then OPOID else COMPLETE_PLAN)  $\wedge$ 
(planPBNS OPOID (exec x) =
 $\text{if}$  getPlCom x = supervise then SUPERVISE else OPOID)  $\wedge$ 
(planPBNS SUPERVISE (exec x) =
 $\text{if}$  getPlCom x = report2 then REPORT2 else SUPERVISE)  $\wedge$ 
(planPBNS REPORT2 (exec x) =
 $\text{if}$  getPlCom x = complete then COMPLETE else REPORT2)  $\wedge$ 
(planPBNS s (trap v0) = s)  $\wedge$  (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)`

```

7.3.6 Next-Output Function

The next-output function is defined similarly to those defined for previously described SSMs.

[planPBOut_def]

```

 $\vdash (\text{planPBOut} \text{ WARNO } (\text{exec } x) =$ 
 $\quad \text{if}$ 
 $\quad (\text{getRecon } x = [\text{SOME } (\text{SLc } (\text{PL recon}))]) \wedge$ 
 $\quad (\text{getTenativePlan } x = [\text{SOME } (\text{SLc } (\text{PL tentativePlan}))]) \wedge$ 
 $\quad (\text{getReport } x = [\text{SOME } (\text{SLc } (\text{PL report1}))]) \wedge$ 
 $\quad (\text{getInitMove } x = [\text{SOME } (\text{SLc } (\text{PSG initiateMovement}))])$ 
 $\quad \text{then}$ 
 $\quad \text{Report1}$ 
 $\quad \text{else unAuthorized}) \wedge$ 
 $(\text{planPBOut} \text{ PLAN_PB } (\text{exec } x) =$ 
 $\quad \text{if getPlCom } x = \text{receiveMission} \text{ then ReceiveMission}$ 
 $\quad \text{else unAuthorized}) \wedge$ 
 $(\text{planPBOut} \text{ RECEIVE_MISSION } (\text{exec } x) =$ 
 $\quad \text{if getPlCom } x = \text{warno} \text{ then Warno} \text{ else unAuthorized}) \wedge$ 
 $(\text{planPBOut} \text{ REPORT1 } (\text{exec } x) =$ 
 $\quad \text{if getPlCom } x = \text{completePlan} \text{ then CompletePlan}$ 
 $\quad \text{else unAuthorized}) \wedge$ 
 $(\text{planPBOut} \text{ COMPLETE_PLAN } (\text{exec } x) =$ 
 $\quad \text{if getPlCom } x = \text{oqid} \text{ then Oqid} \text{ else unAuthorized}) \wedge$ 
 $(\text{planPBOut} \text{ OPOID } (\text{exec } x) =$ 
 $\quad \text{if getPlCom } x = \text{supervise} \text{ then Supervise}$ 
 $\quad \text{else unAuthorized}) \wedge$ 
 $(\text{planPBOut} \text{ SUPERVISE } (\text{exec } x) =$ 
 $\quad \text{if getPlCom } x = \text{report2} \text{ then Report2} \text{ else unAuthorized}) \wedge$ 
```

```

(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

Authentication for this SSM is the same as for the previously described SSMs.

```

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)` 

```

It is straight forward to prove 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

The two authorization functions are defined below.

```
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] ) /\ \
     (getTentativePlan   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 *PL_WARNO_Auth* and *PL_notWARNO_Auth*. These are defined below.

```

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``)`

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

Theorems proving complete mediation follow the same format as those for previously described SSMs.

**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 lemmas and a theorem and follows the same pattern as the other three-part proofs. The pretty-printed HOL-generated output is shown below.

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma]

```

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

```

propCommandList
  [Name PlatoonLeader says
    prop (SOME (SLC (PL plCommand)))]
  [PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_
lemma]

```

$\vdash s \neq \text{WARNO} \Rightarrow$

$plCommand \neq \text{invalidPlCommand} \Rightarrow$

$plCommand \neq \text{report1} \Rightarrow$

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

TR (M, Oi, Os)

(exec

(inputList

[Name PlatoonLeader says

prop (SOME (SLC (PL *plCommand*))))])

(CFG inputOK secContext secContextNull

([Name PlatoonLeader says

prop (SOME (SLC (PL *plCommand*))))] :: *ins*) s *outs*)

(CFG inputOK secContext secContextNull *ins*

(*NS* s

(exec

(inputList

[Name PlatoonLeader says

prop (SOME (SLC (PL *plCommand*))))])

(*Out* s

(exec

(inputList

[Name PlatoonLeader says

prop (SOME (SLC (PL *plCommand*))))] ::

outs)) \iff

```

authenticationTest inputOK

[Name PlatoonLeader says

prop (SOME (SLc (PL plCommand))) ]  $\wedge$ 

CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secContextNull

([Name PlatoonLeader says

prop (SOME (SLc (PL plCommand))) ] ::ins) s outs)  $\wedge$ 

(M, Oi, Os) satList

propCommandList

[Name PlatoonLeader says

prop (SOME (SLc (PL plCommand))) ]

```

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]

$\vdash s \neq \text{WARNO} \Rightarrow$

$plCommand \neq \text{invalidPlCommand} \Rightarrow$

$plCommand \neq \text{report1} \Rightarrow$

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

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) ∧
(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 *slCommands*. (Only unauthenticated requests are discarded.) This theorem has no lemmas.

[PlatoonLeader_psgCommand_notDiscard_thm]

```

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

[PlatoonLeader_trap_psgCommand_lemma]

```

⊢ ∀M Oi Os.
CFGInterpret (M, Oi, Os)
(CFG inputOK secContext secContextNull

```

```

([Name PlatoonLeader says
prop (SOME (SLc (PSG psgCommand))) ] :: ins) s outs) =>
(M, Oi, Os) sat prop NONE

```

[PlatoonLeader_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) ∧
(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

```

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 and a theorem.

[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) ⇒
  (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)) ⇔

authenticationTest inputOK

[Name PlatoonLeader says prop (SOME (SLc (PL recon))));

Name PlatoonLeader says

prop (SOME (SLc (PL tentativePlan))));

Name PlatoonSergeant says

prop (SOME (SLc (PSG initiateMovement))));

Name PlatoonLeader says

prop (SOME (SLc (PL report1))))] ∧

CFGInterpret ( $M, O_i, O_s$ )

(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]

```

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

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 ( $M, O_i, O_s$ )
(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)))]
```

PlatoonSergeant_trap_plCommand_justified_thm This final theorem proves that the PlatoonSergeant is trapped on all *plCommands*. This is for the same reason that the PlatoonLeader is trapped on all *psgCommands*.

[PlatoonSergeant_trap_plCommand_lemma]

```

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

```

[PlatoonSergeant_trap_plCommand_justified_lemma]

```

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

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

 $(\text{trap}$ 
 $\quad (\text{inputList}$ 
 $\quad [\text{Name PlatoonSergeant says}$ 
 $\quad \quad \text{prop (SOME (SLc (PL } plCommand) ))]))$ 
 $\quad (\text{CFG inputOK secContext secContextNull}$ 
 $\quad (\text{[Name PlatoonSergeant says}$ 
 $\quad \quad \text{prop (SOME (SLc (PL } plCommand) ))] :: ins) \ s \ outs)$ 
 $\quad (\text{CFG inputOK secContext secContextNull ins}$ 
 $\quad (NS \ s$ 
 $\quad (\text{trap}$ 
 $\quad (\text{inputList}$ 
 $\quad [\text{Name PlatoonSergeant says}$ 
 $\quad \quad \text{prop (SOME (SLc (PL } plCommand) ))]))$ 
 $\quad (\text{Out } s$ 

```

```

(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, O_i, O_s) sat prop NONE

In concluding this chapter, note that the model here is a first approximation to modeling the patrol base operations. Some of the proofs may not accurately reflect real patrol base operations. For example, the Platoon Leader should probably be authorized on ALL commands. But, we applied the principal of least privilege and restricted transition requests to the responsible principal rather than all who should have been permitted.

The next chapter discusses the results described in the previous chapters.

Chapter 8

Discussion

The previous chapters describe the application of STORM to the patrol base operations. This chapter elaborates on the STPA and CSBD analyses by discussing findings not covered in previous chapters. It then concludes with some discussion of challenges of the approach and how they are overcome.

8.1 STPA Analysis

8.2 CSBD Analysis

8.2.1 Authentication

Authentication for this model of the patrol base operations is assumed through visual recognition of an authority. But, in many parts of the patrol base operations, soldiers are challenged and must provide a password. CSBD manages passwords (and cryptographic functions for automated systems) using a cryptographic secure state

machine. Such a parametrizable secure state machine already exists and it would be straight forward to apply it to future patrol base operations or to other non-automated, human-centered systems.

8.2.2 Roles

Principals in this model are roles (i.e., Platoon Leader, Platoon Sergeant, etc.). But, a more accurate description would include soldiers acting in the role of a principal. CSBD manages principals acting in roles using various ACL rules and a specific secure state machine. Like cryptographic functions, such a parametrizable secure state machine already exists. It would be straight forward to apply it to future patrol base operations or to other non-automated, human-centered systems.

8.3 Soldier, Squad, and Platoon Theories

The model describes transitions among phases of the operations. Additional models could include soldiers, squads, and platoon modules. Figure 8.1 is a possible description of a high level soldier module with one level expanded. This is a simplified model to demonstrate the idea of a soldier module.

The soldier begins in the in-processing state. She is then assigned to installation or community in-processing. Then, she is assigned to a battalion or unit. From there, she is sent on a mission or sent to out-processing. From the mission state, she can be sent to out processing or proceed through the mission. The mission begins in the planning phase where she receives her WARNO, prepares and rehearses, receives her OPORD, and is then supervised. The ReceiveMission state would lead to other activities within the patrol base operations.

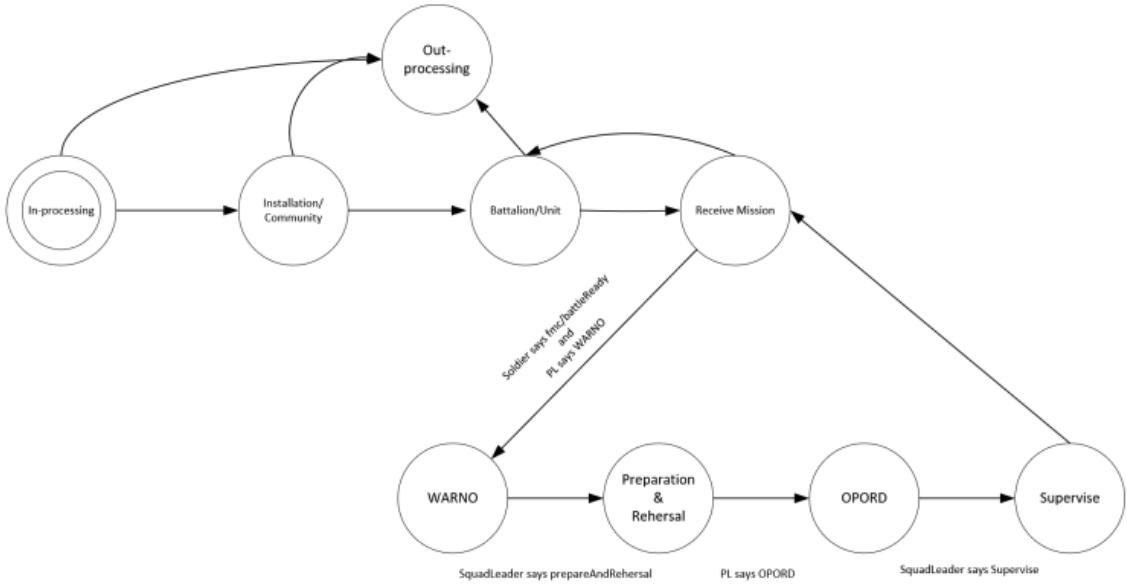


Figure 8.1: An abbreviated concept for a soldier module.

This is a basic idea of a soldier secure state machine. It requires more research into the processing and soldier steps. But, it is another way of thinking about the patrol base operations in terms of complete mediation.

The reader could imagine similar secure state machines for the platoon, squad, and team levels.

8.4 Non-sequential Variations

The secure state machines described in the previous chapters describe mostly linear, sequential transitions. But, non-linear and non-sequential secure state machines may describe some of the operations more accurately. One example of non-linear, non-sequential operations is described in the planning secure state machine at the sub-level. That secure state machine is shown again in figure 8.2.

For this state machine, three of the states could be completed in any order. But, they ALL must be completed before completing the plan. These states are

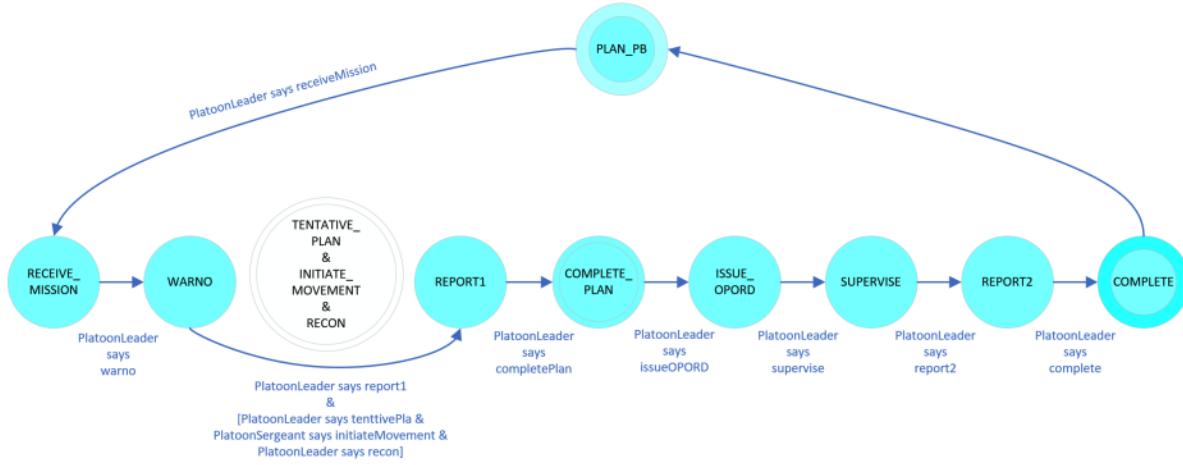


Figure 8.2: Horizontal slice: PlanPB diagram.

TENTATIVE_PLAN, INITIATE_MOVEMENT, and RECON. The secure state machine in figure 8.2 describes these as "virtual states" and simply requires their completion to be part of the input to the transition request.

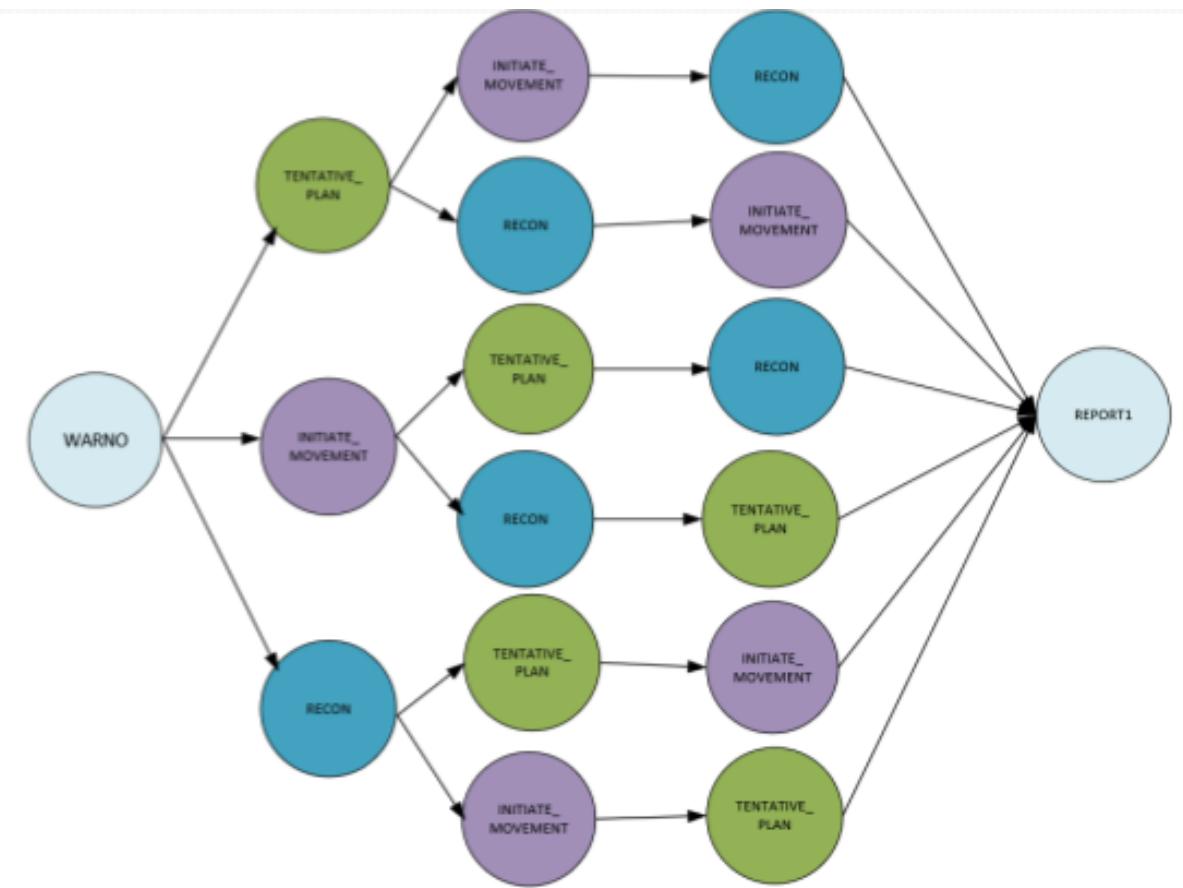


Figure 8.3: Alternative path through the three non-sequential states.

Another approach to modeling this section of the secure state machine is shown in

figure 8.3. This diagrams show all permutations of paths from the WARNO state to the REPORT1 state. From the WARNO state, the operations can move to one of the TENTATIVE_PLAN, INITIATE_PLAN, or RECON states. From there, the operations can move to any one of the two states that is has not yet occupied. Finally, the operations move to the state it hasn't occupied yet. There are a total of 6 paths from WARNO to REPORT1.

Another approach to modeling non-linear sequential states is to allow more transitions. For example, the top level progresses from the PLAN_PB state sequentially through to the COMPLETE_PB state. But, it is possible that a platoon receives orders to move to the objective rally point before receiving a mission. Figure 8.4 shows an example of how to model this as a secure state machine.

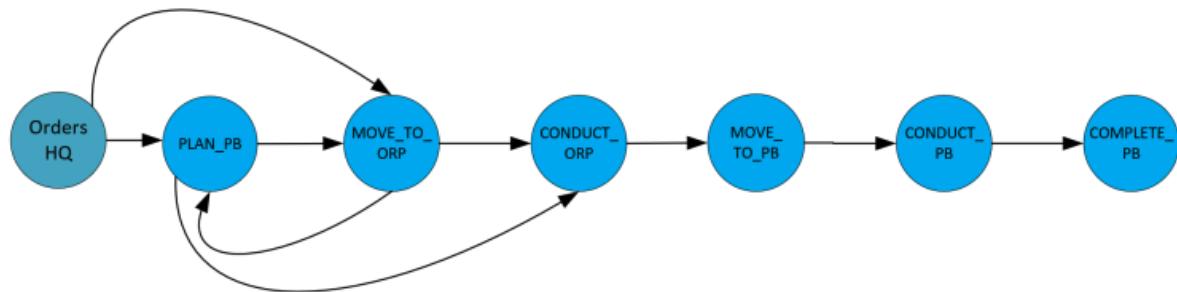


Figure 8.4: Permutation paths through the three non-sequential states..

The additional arrows allow for the platoon to receive orders from head quarters and move directly to the objective rally point before the planning phase is complete. From there, the platoon can move into the planning phase and then conduct activities at the ORP.

Additionally, the planning phase could begin while moving to the objective rally point. This would eliminate encapsulation of each module, but it would be a more accurate description and the hierarchy could remain in-tact.

8.5 Concluding Comments

STORM works. Applying it to a non-automated, human-centered system is straight forward.

The biggest challenge is managing the large size of the operations. The patrol base operations are composed of numerous activities. This challenge is dealt with using a modularized hierarchy of secure state machines.

The next biggest challenge for the STPA/STPA-Sec analysis is expertise. A subject matter expert¹ is necessary to describe the scenarios and causes of accidents and vulnerabilities. For the CSBD, a subject matter expert is necessary to interpret the Ranger Handbook [9] into secure state machines.

Another challenge is managing complexity. Some activities should be conducted in a sequential, checklist-like fashion. But, others may be conducted in a variety of orders. In addition, there is more than one type of patrol bases. The Ranger Handbook describes the three types of patrol base as combat, security, and reconnaissance. A complete model of the patrol base operations is beyond the scope of this master thesis. But, it is doable and not much more involved than the model described in this master thesis.

With the modularized hierarchy of secure state machines, large and complex systems non-automated, human-centered systems can be managed. The model described in this master thesis is just a starting point and intended to demonstrate feasibility. But, a full analysis of a system of this size and variety would start here and then adjust the transitions and details to accommodate.

One of the biggest take homes from a design perspective is thinking critically about the

¹A subject matter expert was available for the modularized hierarchy of secure state machines. But, he was not available for the STPA analysis. The author learned STPA and patrol base operations simultaneously.

system's security. The detailed analysis of the system from the perspective of hazard and security analysis and complete mediation is informative.

The next chapter discusses future work and some ideas about partial automation of the system with regards to accountability applications.

Chapter 9

Future Work

This chapter discusses plans and possibilities for future work with the patrol base operations and STORM.

9.1 Patrol Base Operations

The purpose of this master thesis is to demonstrate STORM on a non-automated, human-centered system. This work is complete. There are currently no plans to extend this particular project. However, this project left the author with a mind full of ideas on how to modify the secure state machines to manage the variety and complexity of the patrol base operations. Some modifications are discussed in the previous chapter. It would be interesting to see how far STORM could go with regards to the patrol base operations and military operations in general. For example, would STORM be effective on a battalion, mission, entire operations, and the U.S. Army as a whole?

9.2 Automation

There is no doubt that the future of non-automated, human-centered systems is, at least, partial automation. Partial automation could be an accountability system that tracks soldiers through the operations using an implant, chip embedded in dog tags, a hand-held device, etc.. It could be an accountability system that tracks equipment (an equipment module). It could also be an accountability system that tracks the progression of operations as a whole by combining tracking data from one system with tracking data and intelligence from other systems.¹ It could even be an accountability system that tracks the health of soldiers.² With advances in computational power, automated intelligent systems, and technology nearly anything imaginable is possible.

The design of the patrol base operations as a modularized hierarchy of secure state machines is readily automated. The automation follows from the system's well-defined behavior, which is constrained by the nature of the secure state machines. Furthermore, secure state machines enforce security as demonstrated by the formal application of the access-control logic.

9.2.1 STORM

The components of STORM are gaining popularity because they are proving effective and efficient for analyzing system safety and security. Already, additional components for STPA exist to analyze other systems engineering concerns such as privacy and large, software intensive systems. Such systems may also be STORM-applicable, particularly systems where privacy is a concern such as medical records or communications systems.

¹Artificial intelligence could be used to analyze the operations as a whole, make predictions and simulations, and assist decision-makers at the Pentagon, for example, in determining the best course of action for the operations. It could also be used to review and analyze the operations after the fact to make improvements in future operations.

²There already exists micro-bots that track blood pressure and whether an individual has taken his medication. This data is transmitted via wifi to the appropriate medical practitioners.

STPA-Sec could be supplemented with STPA-Priv. CSBD is already designed to reason about privacy.

Future work for STORM involves generating more examples of STORM and training individuals to use it. There is also an application for the STPA-Sec component of STORM called XSTAMPP. But, there remains some bugs to work through.³

What is unquestionable is the need to develop systems that are trustworthy. Trustworthiness guarantees the safety, security, and privacy of man-made systems and can only be realized through rigorous analysis of best practices in the field of systems security engineering. The field's Systems Security Engineering Framework guides a security analysis and is implemented through methodologies such as STORM. STORM works. It is effective and efficient, and that's the bottom line.

³Some people have got the program to work.

Appendices

Appendix A

Background

This appendix provides some supplemental information that may be helpful to the reader. It discusses formal methods, functional programming, algebraic data types in ML, the Higher Order Logic (HOL) Interactive Theorem Prover, other theorem provers, and how to compile the LaTeX and HOL files included with this master thesis.

A.1 Formal Methods

(Primary source for this section is [22])

Formal methods improve the reliability and correctness of systems [23]. 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 process¹.

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 [22]. Specification consists of describing the system in a mathematical-based modeling language [22]. 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

¹For example, there is notable progress being made in formally verifying c code. See for example [24] and [25]

explosion problem, wherein the number of states of the system grows exponentially with the complexity of the system [26].

Theorem proving, on the other hand, employs a formal logic to prove properties of the system. Formal proofs remain true for any test case [22]. 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 [27]. It often requires specialized knowledge and a sufficient degree of competency in mathematics². However, formal theorem proving techniques have been successfully taught to undergraduates³.

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 [28].

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[22].

Specification has its benefits, not only as a precursor to verification, but also as a tool for understanding. Even in cases wherein properties of the system are not proved formally, the act of formally specifying the system adds a degree of rigor to the process. This rigor 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 communicability⁴ [22].

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.

A.2 Functional Programming

(Primary source for this section is [29])

Functional programming is a style of programming that uses functions to define program

²and a good amount of patience, in the author's opinion.

³The PI Professor Shiu-Kai Chin teaches CSBD, which includes theorem proving, to both undergraduates and graduates at Syracuse University.

⁴The later ideas are not specifically stated in the main cite, but follow logically from the author's perspective.

behavior. Haskell and ML or polyML⁵ (meta language) are examples of functional programming languages. Functional programming is inherently different than procedural or object-oriented programming, which 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.

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 (polyML). HOL is thought to be very reliable and trustworthy.

A.3 Algebraic Data Types in ML

The primary source for this section is *Introduction to Programming Languages/Algebraic Data Types* [8].

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, ADT are preceded by the datatype keyword. The boolean data type is defined in figure A.1.

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

Figure A.1: An implementation of the boolean datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [8]

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

⁵ML is not considered a purely functional language. But, it's pretty close. Even Haskell, which is the archetypical functional programming language, has non-pure components, in particular IO operations.

```

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

```

Figure A.2: An implementation of a tree datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [8]

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).

In ML,⁶ a type constructor is 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 is a constructor that behaves like a function which returns something of type 'a tree [30].

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 A.3: An implementation of the weekday datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [8]

With pattern matching, it is possible to define a function *is_weekend* which returns "True" if it is a weekend and "False" otherwise. This is shown in figure A.4.

```

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

```

Figure A.4: An implementation of the function *is_weekend* in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [8]

⁶Standard ML according to the author [30].

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 that argument equals "Saturday" ML will return "True", otherwise it will check the next line. The next line contains the "`_`" 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 A.5: An implementation of a parametrized tree datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [8]

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 as in figure A.2. But, it can also produce a tree of characters with the declaration `char tree`.

`'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.

A.4 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. 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 [31] and is descended from HOL88 citeHOL. Download instructions can be found at the HOL website hol-theorem-prover.org [32].

A.5 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/HOL 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 at Syracuse University.

A.6 How to Compile The Included Files

All the files necessary to compile the theories discussed in this mater thesis are included. A diagram of the folder structure is included in the appendix F. 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 installed.

Appendix B

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 C

Parametrizable 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) ⇒

```

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

[TR_trap_cmd_rule]

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

[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`]

$\vdash \forall P'.$

$$\begin{aligned}
& (\forall P \text{ 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 \\
& (\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
\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)))

```

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

[TR_trap_cmd_rule]

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

[TRrule0]

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

[TRrule1]

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

[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]

$$\begin{aligned}
 \vdash & (\text{getPlCom } [] = \text{incomplete}) \wedge \\
 & (\forall xs cmd. \text{getPlCom } (\text{SOME } (\text{SLc } (\text{PL } cmd))::xs) = cmd) \wedge \\
 & (\forall xs. \text{getPlCom } (\text{NONE}::xs) = \text{getPlCom } xs) \wedge \\
 & (\forall xs v_4. \text{getPlCom } (\text{SOME } (\text{ESCc } v_4)::xs) = \text{getPlCom } xs) \wedge \\
 & \forall xs v_9. \text{getPlCom } (\text{SOME } (\text{SLc } (\text{OMNI } v_9))::xs) = \text{getPlCom } xs
 \end{aligned}$$

[getPlCom_ind]

$$\begin{aligned}
 \vdash & \forall P. \\
 & P [] \wedge (\forall cmd xs. P (\text{SOME } (\text{SLc } (\text{PL } cmd))::xs)) \wedge \\
 & (\forall xs. P xs \Rightarrow P (\text{NONE}::xs)) \wedge \\
 & (\forall v_4 xs. P xs \Rightarrow P (\text{SOME } (\text{ESCc } v_4)::xs)) \wedge \\
 & (\forall v_9 xs. P xs \Rightarrow P (\text{SOME } (\text{SLc } (\text{OMNI } v_9))::xs)) \Rightarrow \\
 & \forall v. P v
 \end{aligned}$$

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]

```

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

```

[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 equi 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]

```

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

```

$\forall v. \ P \ v$

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

[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  $v_1$ ))  $\wedge$ 
( $\forall v_2 v_3. P$  (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**]

$$\vdash (\text{planPBNS WARNO (exec } x) = \\ \text{if} \\ (\text{getRecon } x = [\text{SOME (SLc (PL recon))}] \wedge \\ (\text{getTenativePlan } x = [\text{SOME (SLc (PL tentativePlan))}] \wedge \\ (\text{getReport } x = [\text{SOME (SLc (PL report1))}] \wedge \\ (\text{getInitMove } x = [\text{SOME (SLc (PSG initiateMovement))}])) \\ \text{then} \\ \text{REPORT1} \\ \text{else WARNO}) \wedge$$

```

(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]

$$\vdash \forall NS \ Out \ M \ Oi \ Os . \\
 \text{TR} \ (M, Oi, Os) \\
 (\text{exec} \\
 (\text{inputList} \\
 [\text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL recon)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL tentativePlan)))}; \\
 \text{Name PlatoonSergeant says} \\
 \text{prop (SOME (SLc (PSG initiateMovement)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL report1))))})) \\
 (\text{CFG inputOK secContext secContextNull} \\
 ([\text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL recon)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL tentativePlan)))}; \\
 \text{Name PlatoonSergeant says} \\
 \text{prop (SOME (SLc (PSG initiateMovement)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL report1))))] :: ins) \ \text{WARNO outs}) \\
 (\text{CFG inputOK secContext secContextNull} \\
 (\text{NS} \ \text{WARNO} \\
 (\text{exec} \\
 (\text{inputList} \\
 [\text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL recon)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL tentativePlan)))}; \\
 \text{Name PlatoonSergeant says} \\
 \text{prop (SOME (SLc (PSG initiateMovement)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL report1))))})) \\
 (\text{Out} \ \text{WARNO} \\
 (\text{exec} \\
 (\text{inputList} \\
 [\text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL recon)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL tentativePlan)))}; \\
 \text{Name PlatoonSergeant says} \\
 \text{prop (SOME (SLc (PSG initiateMovement)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL report1))))] :: outs)) \iff \\
 \text{authenticationTest inputOK} \\
 [\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))}; \\
 \text{Name PlatoonLeader says} \\
 \text{prop (SOME (SLc (PL tentativePlan)))}; \\$$

```

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  $xs$ )  $\wedge$ 
 $(\forall xs v_{12}. \text{getInitMove } (v_{12} \text{ says } \text{TT}::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v_{12}. \text{getInitMove } (v_{12} \text{ says } \text{FF}::xs) = \text{getInitMove } xs) \wedge$ 
 $(\forall xs v134.$ 
  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]

```

 $\vdash \forall P.$ 
 $P [] \wedge$ 
 $(\forall cmd\ xs.$ 
 $P$ 
 $(\text{Name PlatoonLeader says prop (SOME (SLc (PL cmd)))::}$ 
 $xs)) \wedge (\forall xs.\ P\ xs \Rightarrow P\ (\text{TT}::xs)) \wedge$ 
 $(\forall xs.\ P\ xs \Rightarrow P\ (\text{FF}::xs)) \wedge$ 
 $(\forall v_2\ xs.\ P\ xs \Rightarrow P\ (\text{prop}\ v_2::xs)) \wedge$ 
 $(\forall v_3\ xs.\ P\ xs \Rightarrow P\ (\text{notf}\ v_3::xs)) \wedge$ 
 $(\forall v_4\ v_5\ xs.\ P\ xs \Rightarrow P\ (v_4\ \text{andf}\ v_5::xs)) \wedge$ 
 $(\forall v_6\ v_7\ xs.\ P\ xs \Rightarrow P\ (v_6\ \text{orf}\ v_7::xs)) \wedge$ 
 $(\forall v_8\ v_9\ xs.\ P\ xs \Rightarrow P\ (v_8\ \text{impf}\ v_9::xs)) \wedge$ 
 $(\forall v_{10}\ v_{11}\ xs.\ P\ xs \Rightarrow P\ (v_{10}\ \text{eqf}\ v_{11}::xs)) \wedge$ 
 $(\forall v_{12}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ \text{TT}::xs)) \wedge$ 
 $(\forall v_{12}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ \text{FF}::xs)) \wedge$ 
 $(\forall v_{134}\ xs.\ P\ xs \Rightarrow P\ (\text{Name}\ v_{134}\ \text{says}\ \text{prop}\ \text{NONE}::xs)) \wedge$ 
 $(\forall v_{146}\ xs.$ 
 $P\ xs \Rightarrow$ 
 $P$ 
 $(\text{Name PlatoonLeader says prop (SOME (ESCc}\ v_{146}))::$ 
 $xs)) \wedge$ 
 $(\forall v_{151}\ xs.$ 
 $P\ xs \Rightarrow$ 
 $P$ 
 $(\text{Name PlatoonLeader says}$ 
 $\text{prop (SOME (SLc (PSG}\ v_{151}))::xs)) \wedge$ 
 $(\forall v_{144}\ xs.$ 
 $P\ xs \Rightarrow$ 
 $P\ (\text{Name PlatoonSergeant says prop (SOME}\ v_{144})::xs)) \wedge$ 
 $(\forall v_{135}\ v_{136}\ v_{68}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{135}\ \text{meet}\ v_{136}\ \text{says prop}\ v_{68}::xs)) \wedge$ 
 $(\forall v_{137}\ v_{138}\ v_{68}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{137}\ \text{quoting}\ v_{138}\ \text{says prop}\ v_{68}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{69}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says notf}\ v_{69}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{70}\ v_{71}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ (v_{70}\ \text{andf}\ v_{71})::xs)) \wedge$ 
 $(\forall v_{12}\ v_{72}\ v_{73}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ (v_{72}\ \text{orf}\ v_{73})::xs)) \wedge$ 
 $(\forall v_{12}\ v_{74}\ v_{75}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ (v_{74}\ \text{impf}\ v_{75})::xs)) \wedge$ 
 $(\forall v_{12}\ v_{76}\ v_{77}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ (v_{76}\ \text{eqf}\ v_{77})::xs)) \wedge$ 
 $(\forall v_{12}\ v_{78}\ v_{79}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ v_{78}\ \text{says}\ v_{79}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{80}\ v_{81}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ v_{80}\ \text{speaks_for}\ v_{81}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{82}\ v_{83}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ v_{82}\ \text{controls}\ v_{83}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{84}\ v_{85}\ v_{86}\ xs.$ 
 $P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ \text{reps}\ v_{84}\ v_{85}\ v_{86}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{87}\ v_{88}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ v_{87}\ \text{domi}\ v_{88}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{89}\ v_{90}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ v_{89}\ \text{eqi}\ v_{90}::xs)) \wedge$ 
 $(\forall v_{12}\ v_{91}\ v_{92}\ xs.\ P\ xs \Rightarrow P\ (v_{12}\ \text{says}\ v_{91}\ \text{doms}\ v_{92}::xs)) \wedge$ 

```

$$\begin{aligned}
& (\forall v_{12} v_{93} v_{94} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{93} \text{ eqs } v_{94}::xs)) \wedge \\
& (\forall v_{12} v_{95} v_{96} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{95} \text{ eqn } v_{96}::xs)) \wedge \\
& (\forall v_{12} v_{97} v_{98} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{97} \text{ lte } v_{98}::xs)) \wedge \\
& (\forall v_{12} v_{99} v_{100} xs. P xs \Rightarrow P (v_{12} \text{ says } v_{99} \text{ lt } v_{100}::xs)) \wedge \\
& (\forall v_{14} v_{15} xs. P xs \Rightarrow P (v_{14} \text{ speaks_for } v_{15}::xs)) \wedge \\
& (\forall v_{16} v_{17} xs. P xs \Rightarrow P (v_{16} \text{ controls } v_{17}::xs)) \wedge \\
& (\forall v_{18} v_{19} v_{20} xs. P xs \Rightarrow P (\text{reps } v_{18} v_{19} v_{20}::xs)) \wedge \\
& (\forall v_{21} v_{22} xs. P xs \Rightarrow P (v_{21} \text{ domi } v_{22}::xs)) \wedge \\
& (\forall v_{23} v_{24} xs. P xs \Rightarrow P (v_{23} \text{ eqi } v_{24}::xs)) \wedge \\
& (\forall v_{25} v_{26} xs. P xs \Rightarrow P (v_{25} \text{ doms } v_{26}::xs)) \wedge \\
& (\forall v_{27} v_{28} xs. P xs \Rightarrow P (v_{27} \text{ eqs } v_{28}::xs)) \wedge \\
& (\forall v_{29} v_{30} xs. P xs \Rightarrow P (v_{29} \text{ eqn } v_{30}::xs)) \wedge \\
& (\forall v_{31} v_{32} xs. P xs \Rightarrow P (v_{31} \text{ lte } v_{32}::xs)) \wedge \\
& (\forall v_{33} v_{34} xs. P xs \Rightarrow P (v_{33} \text{ lt } v_{34}::xs)) \Rightarrow \\
& \forall v. P v
\end{aligned}$$

[getPsgCom_def]

$$\begin{aligned}
\vdash & (\text{getPsgCom } [] = \text{invalidPsgCommand}) \wedge \\
& (\forall xs cmd. \\
& \quad \text{getPsgCom} \\
& \quad (\text{Name PlatoonSergeant says prop (SOME (SLc (PSG } cmd))):: \\
& \quad \quad xs) = \\
& \quad cmd) \wedge (\forall xs. \text{getPsgCom (TT::xs)} = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs. \text{getPsgCom (FF::xs)} = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_2. \text{getPsgCom (prop } v_2::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_3. \text{getPsgCom (notf } v_3::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_5 v_4. \text{getPsgCom (v}_4 \text{ andf } v_5::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_7 v_6. \text{getPsgCom (v}_6 \text{ orf } v_7::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_9 v_8. \text{getPsgCom (v}_8 \text{ impf } v_9::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_{11} v_{10}. \text{getPsgCom (v}_{10} \text{ eqf } v_{11}::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_{12}. \text{getPsgCom (v}_{12} \text{ says TT}::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_{12}. \text{getPsgCom (v}_{12} \text{ says FF}::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_{134}. \\
& \quad \quad \text{getPsgCom (Name } v_{134} \text{ says prop NONE}::xs) = \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_{144}. \\
& \quad \quad \text{getPsgCom (Name PlatoonLeader says prop (SOME } v_{144})::xs) = \\
& \quad \quad \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_{146}. \\
& \quad \quad \text{getPsgCom} \\
& \quad \quad (\text{Name PlatoonSergeant says prop (SOME (ESCc } v_{146})):: \\
& \quad \quad \quad xs) = \\
& \quad \quad \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_{150}. \\
& \quad \quad \text{getPsgCom} \\
& \quad \quad (\text{Name PlatoonSergeant says prop (SOME (SLc (PL } v_{150})):: \\
& \quad \quad \quad xs) = \\
& \quad \quad \text{getPsgCom } xs) \wedge \\
& \quad (\forall xs v_{68} v_{136} v_{135}. \\
& \quad \quad \text{getPsgCom (v}_{135} \text{ meet } v_{136} \text{ says prop } v_{68}::xs) =
\end{aligned}$$

```

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}) ::} \\
 & \quad \quad \quad 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]

```

 $\vdash (\text{getRecon} \ [] = [\text{NONE}]) \wedge$ 
 $(\forall xs.$ 
 $\quad \text{getRecon}$ 
 $\quad (\text{Name PlatoonLeader says prop (SOME (SLc (PL recon)))} ::$ 
 $\quad \quad xs) =$ 
 $\quad [\text{SOME (SLc (PL recon))}] \wedge$ 
 $\quad (\forall xs. \ \text{getRecon (TT::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs. \ \text{getRecon (FF::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_2. \ \text{getRecon (prop } v_2 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_3. \ \text{getRecon (notf } v_3 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_5 \ v_4. \ \text{getRecon (v}_4 \ \text{andf } v_5 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_7 \ v_6. \ \text{getRecon (v}_6 \ \text{orf } v_7 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_9 \ v_8. \ \text{getRecon (v}_8 \ \text{impf } v_9 :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_{11} \ v_{10}. \ \text{getRecon (v}_{10} \ \text{eqf } v_{11} :: \ xs) = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_{12}. \ \text{getRecon (v}_{12} \ \text{says TT::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v_{12}. \ \text{getRecon (v}_{12} \ \text{says FF::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v134.$ 
 $\quad \quad \text{getRecon (Name } v134 \ \text{says prop NONE::xs)} = \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs \ v146.$ 
 $\quad \quad \text{getRecon}$ 
 $\quad \quad (\text{Name PlatoonLeader says prop (SOME (ESCc } v146))::xs) =$ 
 $\quad \quad \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs.$ 
 $\quad \quad \text{getRecon}$ 
 $\quad \quad (\text{Name PlatoonLeader says}$ 
 $\quad \quad \quad \text{prop (SOME (SLc (PL receiveMission)))::xs)} =$ 
 $\quad \quad \quad \text{getRecon xs}) \wedge$ 
 $\quad (\forall xs.$ 

```

```

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 ⇒

```

```


$$\begin{aligned}
& P \\
& \quad (\text{Name PlatoonLeader says prop (SOME (ESCc v146))} :: \\
& \quad \quad xs) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL receiveMission)))} :: xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL warno)))} :: xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL recon)))} :: xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL report1)))} :: xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL completePlan)))} :: xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL opoid)))} :: xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL supervise)))} :: xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL report2)))} :: xs)) \wedge \\
(\forall xs. & \quad P \; xs \Rightarrow \\
& \quad P \\
& \quad (\text{Name PlatoonLeader says} \\
& \quad \quad \text{prop (SOME (SLc (PL complete)))} :: xs)) \wedge \\
(\forall xs. &
\end{aligned}$$


```

```

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 D

Parametrizable Secure State Machine Theories: HOL Script Files

D.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 *)

```

D.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 E

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

E.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

```

E.2 escapeLevel

E.3 TopLevel

E.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

```

E.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",
          "PBIntegratedDefTheory", "PBTypeIntegratedTheory"];

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

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

val _ = new_theory "PBIntegratedDef";

(* ****
(* 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

```

E.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 ==>
(*   PlatoLeader controls crossLD.
(* In the ssmPlanPB ssm, the last state is COMPLETE. This is reached when the *)
(* the appropriate authority says complete and the transition is made.
(* Note that following the ACL, if P says x and P controls x, then x.
(* Therefore, it is not necessary for anyone to say x at the topLevel, because *)
(* it is already proved at the lower level.
(* However, indicating that at the topLevel remains something to workout.
(* ****)
```

```
structure ssmPBIntegratedScript = struct

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

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

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

```
val _ = new_theory "ssmPBIntegrated";
```

```
(* ****
(* Define next-state and next-output functions
(* ****)
val PBNS_def =
Define `
```

```
(PBNS PLAN_PB (exec 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 (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: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

```

E.4 Horizontal Slice

E.4.1 ssmPlanPB

E.4.1.1 PlanPBType Theory: Type Definitions

```
(* ****)
(* PlanPBType contains definitions for datatypes that are used in *)
(* the PlanPB state machine. *)
(* Contributors: *)
(*   Lori Pickering (HOL implementation), *)
(*   Jesse Hall (content expert), *)
(*   Prof. Shiu-Kai Chin (Principal Investigator). *)
(* Date created: 4 March 2018 *)
(* ****)

structure PlanPBTypeScript = struct

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

open HolKernel Parse boolLib bossLib;
open TypeBase

val _ = new_theory "PlanPBType";

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

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

val _ = Datatype `psgCommand = initiateMovement
| psgIncomplete
| invalidPsgCommand` 

val psgCommand_distinct_clauses = distinct_of ``:psgCommand``
val _ = save_thm("psgCommand_distinct_clauses", psgCommand_distinct_clauses)

val _ = Datatype `slCommand = PL plCommand
| PSG psgCommand` 

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 `slState = PLAN_PB
| RECEIVE_MISSION
| WARNO
| TENTATIVE_PLAN
```

```

| INITIATE_MOVEMENT
| RECON
| REPORT1
| COMPLETE_PLAN
| OPOID
| SUPERVISE
| REPORT2
| COMPLETE

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

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

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

val _ = Datatype `stateRole = PlatoonLeader
| PlatoonSergeant` 

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

val _ = export_theory();

end

```

E.4.1.2 PlanPBDef Theory: Authentication & Authorization Definitions

```

structure PlanPBDefScript = struct

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

open TypeBase listTheory optionTheory
aclemanticsTheory aclfoundationTheory
uavUtilities
OMNITypeTheory PlanPBTTypeTheory
===== end Interactive Mode ===== *)

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

val _ = new_theory "PlanPBDef";
(* ****
(* Stuff
(* ****
(* _____
(* state Interpretation function
(* _____
(* This function doesn't do anything but is necessary to specialize other
(* theorems.
(* _____
val secContextNull_def = Define `
```

```

secContextNull (x:((slCommand command)option , stateRole , 'd,'e)Form list) =
  [(TT:((slCommand command)option , stateRole , 'd,'e)Form)]`  

(* _____ *)  

(* Security context *)  

(* _____ *)  

(* mimick Prof. Chin's C2_L1R1_RB_Auth_def *)  

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`` )`  

`:`((slCommand command)option , stateRole , 'd,'e)Form` )`  

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))`  

`:  

(* Make a function to check for elements in the list *)  

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)`  

val getTentativePlan_def = Define `  

  (getTentativePlan ([]:((slCommand command)option , stateRole , 'd,'e)Form list) =  

   [NONE]) /\  

  (getTentativePlan ((Name PlatoonLeader) says (prop (SOME (SLc (PL tentativePlan))))  

                     :((slCommand command)option , stateRole , 'd,'e)Form::xs)  

                     = [SOME (SLc (PL tentativePlan)):(slCommand command)option]) /\  

  (getTentativePlan (_::xs) = getTentativePlan xs)`  

val getReport_def = Define `  

  (getReport ([]:((slCommand command)option , stateRole , 'd,'e)Form list) =  

   [NONE]) /\  

  (getReport (((Name PlatoonLeader) says (prop (SOME (SLc (PL report1))))  

              :((slCommand command)option , stateRole , 'd,'e)Form::xs)  

              = [SOME (SLc (PL report1)):(slCommand command)option]) /\  

  (getReport (_::xs) = getReport xs)`  

val getInitMove_def = Define `  

  (getInitMove ([]:((slCommand command)option , stateRole , 'd,'e)Form list) =  

   [NONE]) /\  

  (getInitMove (((Name PlatoonSergeant) says (prop (SOME (SLc (PSG initiateMovement))))  

                 :((slCommand command)option , stateRole , 'd,'e)Form::xs)  

                 = [SOME (SLc (PSG initiateMovement)):(slCommand command)option]) /\  

  (getInitMove (_::xs) = getInitMove xs)`  

val getPlCom_def = Define `  

  (getPlCom ([]:((slCommand command)option , stateRole , 'd,'e)Form list) =  

   invalidPlCommand)  

  /\  

  (getPlCom (((Name PlatoonLeader) says (prop (SOME (SLc (PL cmd))))  

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

              cmd))`  


```

```

/\ 
(getPlCom (_::xs) = getPlCom xs)` 

val getPsgCom_def = Define ` 
  (getPsgCom ([]:((slCommand command)option , stateRole , 'd,'e)Form list) = 
   invalidPsgCommand)
/\ 
(getPsgCom (((Name PlatoonSergeant) says (prop (SOME (SLc (PSG cmd))))) 
             :((slCommand command)option , stateRole , 'd,'e)Form::xs) = 
             cmd)
/\ 
(getPsgCom (_::xs) = getPsgCom xs)` 

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)]` 

(* ===== Back-up copy =====
(* Test theorem for EmitTeX printing. *) 

val testTheorem = ASSUME`T=T` 
val _ = save_thm("testTheorem", testTheorem)

===== End back-up copy ===== *)
val _ = export_theory();
end

```

E.4.1.3 ssmPlanPB Theory: Theorems

```

(* ****
(* ssmPlanPB describes the secure state machine derived from the PLAN_PB *)
(* state at the top-level. This is the first secure state machine to use the *)
(* new ssm parameterizable secure state machine. *)
(* Contributors: Lori Pickering (HOL Implementation) *)
(*                 Jesse Nathaniel Hall (subject matter) *)
(* ****)

structure ssmPlanPBScript = struct

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

open TypeBase listTheory optionTheory listSyntax
      acl_infRules aclDrulesTheory aclrulesTheory
      aclsemanticsTheory aclfoundationTheory

```

```

satListTheory ssmTheory ssminfRules uavUtilities
OMNITypeTheory PlanPBTTypeTheory PlanPBDefTheory
ssmPlanPBTheory
===== end Interactive Mode ===== *)

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open acl_infRules aclDrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfRules uavUtilities
open OMNITypeTheory PlanPBTTypeTheory PlanPBDefTheory

val _ = new_theory "ssmPlanPB";

(* -----
(* Define the next-state & next-output functions for the state machine. *)
(* ----- *)
*)

val planPBNS_def = Define `

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

val planPBOut_def = Define `

(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:slState) (trap _) = unAuthorized) /\
(planPBOut (s:slState) (discard _) = unAuthenticated) `

(* -----
(* inputOK: authentication test function *)
(* ----- *)
*)

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)`

(* Any unauthorized command is rejected *)
val inputOK_cmd_reject_lemma =
TAC PROOF(
[], !cmd. ~ (inputOK
((prop (SOME cmd)):((slCommand command) option , stateRole , 'd,'e)Form)) ``),
PROVE_TAC[inputOK_def]`)

(* _____ *)
(* Theorem: PlatoonLeader is authorized on any plCommand *)
(* iff not in WARNO state and *)
(* the plCommand is not report1. *)
(* _____ *)

(* Helper functions *)
val th1 =
ISPECI
[ ``inputOK:((slCommand command) option , stateRole , 'd,'e)Form -> bool``,
``secContextNull :((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 PlatoonLeader) says (prop (SOME (SLc (PL plCommand)))):
((slCommand command) option , stateRole , 'd,'e)Form)] ``,
``ins:((slCommand command) option , stateRole , 'd,'e)Form list list ``,
``(s:s1State)``,
``outs:s1Output output list trType list ``] TR_exec_cmd_rule

val temp = fst(dest_imp(concl th1))

(* lemma *)
val PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma =
TAC PROOF(
[], Term `(~((s:s1State) = WARNO)) ==>
(~((plCommand:plCommand) = invalidPlCommand)) ==>
(~((plCommand:plCommand) = report1)) ==> ^temp`),

DISCH_TAC THEN
DISCH_TAC THEN
DISCH_TAC THEN
ASM_REWRITE_TAC
[CFGInterpret_def, secContext_def, secContextNull_def,
getRecon_def, getTenativePlan_def, getReport_def, getInitMove_def,
getPlCom_def, getPsgCom_def, PL_notWARNO_Auth_def,
inputList_def, extractInput_def, MAP,
propCommandList_def, extractPropCommand_def, satList_CONS,
satList_nil, GSYM satList_conj]
THEN
PROVE_TAC[Controls, Modus_Ponens])

val _ = save_thm ("PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma",
PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma)

(* helper functions *)
val temp2 = snd(dest_imp(concl th1))

(* lemma *)
val PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma =
TAC PROOF(
[], Term `(~((s:s1State) = WARNO)) ==>
(~((plCommand:plCommand) = invalidPlCommand)) ==>
(~((plCommand:plCommand) = report1)) ==> ^temp2`),

DISCH_TAC THEN
DISCH_TAC THEN

```

```

DISCH_TAC THEN
PROVE_TAC
  [PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma,
   TR_exec_cmd_rule])

val _ = save_thm ("PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma",
                  PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma)

(* Main theorem *)
val PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm =
REWRITE_RULE
[propCommandList_def, inputList_def, extractPropCommand_def,
 extractInput_def, MAP] PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma

val _ = save_thm("PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm",
                 PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm)

(* -----
(* PlatoonLeader is authorized on any report1 if this is the WARNO state and *)
(* PlatoonLeader says recon /| *)
(* PlatoonLeader says tentativePlan /| *)
(* PlatoonSergeant says initiateMovement /| *)
(* PlatoonLeader says report1 *)
(* ----- *)
val th1w =
ISPECL
[``inputOK:((slCommand command)option , stateRole , 'd,'e)Form -> bool```,
 ``secContextNull :((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 PlatoonLeader) says (prop (SOME (SLc (PL recon))))]
   :((slCommand command)option , stateRole , 'd,'e)Form;
  (Name PlatoonLeader) says (prop (SOME (SLc (PL tentativePlan))))]
   :((slCommand command)option , stateRole , 'd,'e)Form;
  (Name PlatoonSergeant) says (prop (SOME (SLc (PSG initiateMovement))))]
   :((slCommand command)option , stateRole , 'd,'e)Form;
  (Name PlatoonLeader) says (prop (SOME (SLc (PL report1))))]
   :((slCommand command)option , stateRole , 'd,'e)Form] ```,
 ``ins :((slCommand command)option , stateRole , 'd,'e)Form list list ```,
 ``(WARNO)``,
 ``outs:slOutput output list trType list ``] TR_exec_cmd_rule

(* lemma *)
val PlatoonLeader_WARNO_exec_report1_lemma =
TAC_PROOF(
  ([] , fst(dest_imp(concl th1w))),
ASM_REWRITE_TAC
[CFGInterpret_def, secContextNull_def, secContext_def,
 propCommandList_def, MAP, extractPropCommand_def ,
 satList_CONS, satList_nil, GSYM satList_conj,
 getRecon_def, getTenativePlan_def, getReport_def, getInitMove_def,
 getPICom_def, getPsgCom_def, PL_WARNO_Auth_def]
THEN
PROVE_TAC[Controls , Modus_Ponens])

val _ = save_thm ("PlatoonLeader_WARNO_exec_report1_lemma",
                  PlatoonLeader_WARNO_exec_report1_lemma)

(* lemma *)
val PlatoonLeader_WARNO_exec_report1_justified_lemma =
TAC_PROOF(
  ([] , snd(dest_imp(concl th1w))),
PROVE_TAC
[PlatoonLeader_WARNO_exec_report1_lemma,
 TR_exec_cmd_rule])

val _ = save_thm ("PlatoonLeader_WARNO_exec_report1_justified_lemma",
                  PlatoonLeader_WARNO_exec_report1_justified_lemma)

```

```

val th23 = REWRITE_RULE
[PlatoonLeader_WARNO_exec_report1_lemma,
 PlatoonLeader_WARNO_exec_report1_justified_lemma] th1w

(* Main theorem *)

val PlatoonLeader_WARNO_exec_report1_justified_thm =
REWRITE_RULE
[propCommandList_def, inputList_def, extractPropCommand_def,
 extractInput_def, MAP]
PlatoonLeader_WARNO_exec_report1_justified_lemma

val _ = save_thm("PlatoonLeader_WARNO_exec_report1_justified_thm",
                  PlatoonLeader_WARNO_exec_report1_justified_lemma)

(* _____ *)
(* Theorem: PlatoonLeader is not discarded any psgCommand. *)
(* Note that this is just meant to demonstrate the authenticated inputs are *)
(* not discarded. Instead, they should be trapped. This is because of how *)
(* how the inputOK (authentication) was defined. Note, this proof would also *)
(* be valid for PlatoonLeader on any plCommand. It is not necessary to prove *)
(* this. *)
(* _____ *)
(* Should we put this GENL in the TR_discard_cmd_rule? *)
val th1d =
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 th2d =
ISPECL
[``inputOK:((slCommand command)option, stateRole, 'd,'e)Form -> bool``,
 ``secContextNull :((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 PlatoonLeader) says (prop (SOME (SLc (PSG psgCommand))))``,
   :((slCommand command)option, stateRole, 'd,'e)Form]``,
 ``ins:((slCommand command)option, stateRole, 'd,'e)Form list list``,
 ``(s:slState)``,
 ``outs:slOutput output list trType list``] th1d

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_psgCommand_notDiscard_thm = REWRITE RULE
[inputList_def, extractInput_def, authenticationTest_def, MAP,
 inputOK_def, FOLDRL, th3d, th3d2] th2d

val _ = save_thm("PlatoonLeader_psgCommand_notDiscard_thm",
                  PlatoonLeader_psgCommand_notDiscard_thm)

(* _____ *)
(* Theorem: PlatoonLeader is trapped on any psgCommand. *)
(* _____ *)
val th1t =
ISPECL

```

```

``inputOK:((slCommand command)option , stateRole , 'd,'e)Form -> bool`` ,
``secContextNull :((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 PlatoonLeader) says (prop (SOME (SLc (PSG psgCommand))))]
:((slCommand command)option , stateRole , 'd,'e)Form] `` ,
``ins:((slCommand command)option , stateRole , 'd,'e)Form list list `` ,
``(s:slState)``,
``outs:slOutput output list trType list `` ]
TR_trap_cmd_rule

val PlatoonLeader_trap_psgCommand_lemma =
TAC_PROOF(
[],  

fst(dest_imp(concl th1t))),  

ASM_REWRITE_TAC  

[CFGInterpret_def, inputOK_def, secContext_def, secContextNull_def]  

THEN  

ASM_REWRITE_TAC  

[getRecon_def, getTenativePlan_def, getReport_def, getInitMove_def,  

getPICom_def, satList_CONS, satList_nil, GSYM satList_conj]  

THEN  

ASM_REWRITE_TAC  

[NOT_NONE_SOME_NOT_NONE_SOME_11, list_11,  

slCommand_one_one, slCommand_distinct_clauses,  

plCommand_distinct_clauses, psgCommand_distinct_clauses,  

GSYM slCommand_distinct_clauses,  

GSYM plCommand_distinct_clauses,  

GSYM psgCommand_distinct_clauses]  

THEN  

PROVE_TAC[satList_CONS, satList_nil])

val _ = save_thm("PlatoonLeader_trap_psgCommand_lemma",
PlatoonLeader_trap_psgCommand_lemma)

val PlatoonLeader_trap_psgCommand_justified_lemma =
TAC_PROOF(
[],  

snd(dest_imp(concl th1t))),  

PROVE_TAC  

[PlatoonLeader_trap_psgCommand_lemma, TR_trap_cmd_rule])

val _ = save_thm("PlatoonLeader_trap_psgCommand_justified_lemma",
PlatoonLeader_trap_psgCommand_justified_lemma)

val PlatoonLeader_trap_psgCommand_justified_thm =
REWRITE_RULE
[propCommandList_def, inputList_def, extractPropCommand_def,
extractInput_def, MAP]
PlatoonLeader_trap_psgCommand_justified_lemma

val _ = save_thm("PlatoonLeader_trap_psgCommand_justified_lemma",
PlatoonLeader_trap_psgCommand_justified_lemma)

(* _____ *)
(* Theorem: PlatoonSergeant is trapped on any plCommand. *)
(* _____ *)
(* _____ *)
val th1tt =
ISPECL
[``inputOK:((slCommand command)option , stateRole , 'd,'e)Form -> bool`` ,
``secContextNull :((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 PlatoonSergeant) says (prop (SOME (SLc (PL plCommand))))]
:((slCommand command)option , stateRole , 'd,'e)Form] `` ,
``ins:((slCommand command)option , stateRole , 'd,'e)Form list list `` ,
``(s:slState)``,
``outs:slOutput output list trType list `` ]

```

```

``outs:slOutput output list trType list``
TR_trap_cmd_rule

val PlatoonSergeant_trap_plCommand_lemma =
TAC_PROOF(
  ([] ,
   fst(dest_imp(concl th1tt))) ,
ASM_REWRITE_TAC
[CFGInterpret_def, inputOK_def, secContext_def, secContextNull_def]
THEN
ASM_REWRITE_TAC
[getRecon_def, getTenativePlan_def, getReport_def, getInitMove_def,
 getPlCom_def, satList_CONS, satList_nil, GSYM satList_conj]
THEN
ASM_REWRITE_TAC
[NOT_NONE_SOME_NOT_SOME, SOME_11, list_11,
 slCommand_one_one, slCommand_distinct_clauses,
 plCommand_distinct_clauses, psgCommand_distinct_clauses,
 GSYM slCommand_distinct_clauses,
 GSYM plCommand_distinct_clauses,
 GSYM psgCommand_distinct_clauses]
THEN
PROVE_TAC[satList_CONS, satList_nil])

val _ = save_thm("PlatoonSergeant_trap_plCommand_lemma",
                 PlatoonSergeant_trap_plCommand_lemma)

val PlatoonSergeant_trap_plCommand_justified_lemma =
TAC_PROOF(
  ([] ,
   snd(dest_imp(concl th1tt))) ,
PROVE_TAC
[PlatoonSergeant_trap_plCommand_lemma, TR_trap_cmd_rule])

val _ = save_thm("PlatoonSergeant_trap_plCommand_justified_lemma",
                 PlatoonSergeant_trap_plCommand_justified_lemma)

val PlatoonSergeant_trap_plCommand_justified_thm =
REWRITE_RULE
[propCommandList_def, inputList_def, extractPropCommand_def,
 extractInput_def, MAP]
PlatoonSergeant_trap_plCommand_justified_lemma

val _ = save_thm("PlatoonSergeant_trap_plCommand_justified_thm",
                 PlatoonSergeant_trap_plCommand_justified_lemma)

(* === Start testing here ===

==== End Testing Here === *)
val _ = export_theory();

end

```

E.4.2 ssmMoveToORP

E.4.2.1 MoveToORPType Theory: Type Definitions

E.4.2.2 MoveToORPDef Theory: Authentication & Authorization Definitions

E.4.2.3 ssmMoveToORP Theory: Theorems

E.4.3 ssmConductORP

E.4.3.1 ConductORPType Theory: Type Definitions

```
(* ****)
(* ConductORPType contains definitions for datatypes that are used in      *)
(* the conductORP state machine ssmConductORP.                                *)
(* Contributors:                                                               *)
(*   Lori Pickering (HOL implementation),                                         *)
(*   Jesse Hall (content expert),                                              *)
(*   Prof. Shiu-Kai Chin (Principal Investigator).                            *)
(* Date created: 16 June 2017                                                 *)
(* ****)

structure ConductORPTypeScript = struct

  (* ____ Interactive Mode ____*
  app load ["TypeBase"];
  open TypeBase
  ____ end Interactive Mode ____ *)

  open HolKernel Parse boolLib bossLib;
  open TypeBase

  val _ = new_theory "ConductORPType";
  (* _____
  (* slcommand, slState, slOutput, and stateRole
  (* _____
  (* *****commands ***** *)
  val _ =
    Datatype `plCommand = secure
      | withdraw
      | complete
      | plIncomplete `

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

  val _ =
    Datatype `psgCommand = actionsIn
      | psgIncomplete `

  val psgCommand_distinct_clauses = distinct_of ``:psgCommand``
  val _ = save_thm("psgCommand_distinct_clauses", psgCommand_distinct_clauses)

  val _ =
    Datatype `omniCommand = ssmSecureComplete
      | ssmActionsIncomplete
      | ssmWithdrawComplete
      | invalidOmniCommand `

  val omniCommand_distinct_clauses = distinct_of ``:omniCommand``
```

```

val _ = save_thm("omniCommand_distinct_clauses", omniCommand_distinct_clauses)

val _=
Datatype `slCommand = PL plCommand
| PSG psgCommand
| 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)

(****** states *****)
val _=
Datatype `slState = CONDUCT_ORP
| SECURE
| ACTIONS_IN
| WITHDRAW
| COMPLETE

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

(****** output *****)
val _=
Datatype `slOutput = ConductORP
| Secure
| ActionsIn
| Withdraw
| Complete
| unAuthenticated
| unAuthorized` 

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

(****** principals *****)
val _=
Datatype `stateRole = PlatoonLeader
| PlatoonSergeant
| Omni` 

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

val _ = export_theory();

end

```

E.4.3.2 ConductORPDef Theory: Authentication & Authorization Definitions

```

(******)
(* ConductORPDef *)
(* Author: Lori Pickering *)
(* Date: 11 August 2018 *)
(* Definitions for ssmConductORPDef *)
(******)
structure ConductORPDefScript = struct

(* === Interactive Mode ===
app load ["TypeBase", "listTheory", "optionTheory",
          "acl_infrules", "aclDrulesTheory", "aclrulesTheory",
          "satListTheory", "ssmTheory", "ssminfRules",
          "OMNITypeTheory", "ConductORPTypeTheory"];

"ssmConductORPTheory"];

```

```

open TypeBase listTheory optionTheory
acl_infrules aclDrulesTheory aclrulesTheory
satListTheory ssmTheory ssminfrules
OMNITypeTheory ConductORPTypeTheory
ssmConductORPTheory
===== end Interactive Mode ===== *)

```

```

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open acl_infrules aclDrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfrules
open OMNITypeTheory ConductORPTypeTheory

val _ = new_theory "ConductORPDef";

(* ****
(* Helper functions for extracting commands *)
(* ****)
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))`
```

```

(* ****
(* security context, non state-dependent. *)
(* ****)
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))`
```

```

(* ****
(* security context, state-dependent. *)
(* ****)
val secContext_def =
Define `

  (secContext (CONDUCT_ORP) (xs:((slCommand command)option ,stateRole ,'d,'e)Form list) =
    [(Name PlatoonLeader) controls prop (SOME (SLc (PL secure)))  

     :((slCommand command)option , stateRole , 'd,'e)Form]) /\
  (secContext (SECURE) (xs:((slCommand command)option ,stateRole ,'d,'e)Form list) =
    if ((getOmniCommand xs) = ssmSecureComplete:omniCommand)  

    then  

      [(prop (SOME (SLc (OMNI ssmSecureComplete))))  

       :((slCommand command)option , stateRole , 'd,'e)Form] impf
```

```

        (Name PlatoonSergeant) controls prop (SOME (SLc (PSG actionsIn )))
        :((slCommand command)option , stateRole , 'd , 'e)Form]
      else [prop NONE] /\

(secContext (ACTIONS_IN) (xs:(( slCommand command)option ,stateRole , 'd , 'e)Form list ) =
  if ((getOmniCommand xs) = ssmActionsIncomplete)
  then
    [(prop (SOME (SLc (OMNI ssmActionsIncomplete))))
     :((slCommand command)option , stateRole , 'd , 'e)Form) impf
     (Name PlatoonLeader) controls prop (SOME (SLc (PL withdraw)))
     :((slCommand command)option , stateRole , 'd , 'e)Form]
  else [prop NONE] /\

(secContext (WITHDRAW) (xs:(( slCommand command)option ,stateRole , 'd , 'e)Form list ) =
  if ((getOmniCommand xs) = ssmWithdrawComplete)
  then
    [(prop (SOME (SLc (OMNI ssmWithdrawComplete))))
     :((slCommand command)option , stateRole , 'd , 'e)Form) impf
     (Name PlatoonLeader) controls prop (SOME (SLc (PL complete)))
     :((slCommand command)option , stateRole , 'd , 'e)Form]
  else [prop NONE])`

val _ = export_theory();
end

```

E.4.3.3 ssmConductORP Theory: Theorems

```

(* ****
(* ssmConductORP defines the ConductORP sub-level state machine for the      *)
(* patrol base.                                                               *) 
(* Each state , save for the end states , has a sub-level state machine, and   *)
(* some have sub-sub-level state machines. These are implemented in separate   *)
(* theories.                                                                *) 
(* Author: Lori Pickering in collaboration with Jesse Nathaniel Hall          *)
(* Date: 16 July 2017                                                       *) 
(* ****)

structure ssmConductORPScript = struct

(* === Interactive Mode ===
app load ["TypeBase", "listTheory", "optionTheory",
          "acl_infRules", "aclDrulesTheory", "aclrulesTheory",
          "satListTheory", "ssmTheory", "ssminfRules",
          "OMNITypeTheory",
          "ConductORPTypeTheory", "ConductORPDefTheory",
          "ssmConductORPTheory"];
*)

open TypeBase listTheory optionTheory
acl_infRules aclDrulesTheory aclrulesTheory
satListTheory ssmTheory ssminfRules
OMNITypeTheory
ConductORPTypeTheory ConductORPDefTheory
ssmConductORPTheory
== end Interactive Mode == *)

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open acl_infRules aclDrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfRules
open OMNITypeTheory ConductORPTypeTheory ConductORPDefTheory

val _ = new_theory "ssmConductORP";

(* _____
(* Define the next state function for the state machine.                    *)
(* _____
(* _____
val conductORPNS_def =
Define
` 

  (conductORPNS CONDUCT_ORP (exec x) =
   if (getPlCom x) = secure then SECURE else CONDUCT_ORP) /\

  (conductORPNS SECURE (exec x) =

```

```

    if (getPsgCom x) = actionsIn then ACTIONS_IN else SECURE) /\ 
(conductORPNS ACTIONS_IN (exec x) = 
    if (getPlCom x) = withdraw then WITHDRAW else ACTIONS_IN) /\ 
(conductORPNS WITHDRAW (exec x) = 
    if (getPlCom x) = complete then COMPLETE else WITHDRAW) /\ 
(* trapping *) 
    (conductORPNS (s:slState) (trap x) = s) /\ 
(* discarding *) 
    (conductORPNS (s:slState) (discard x) = s)` 

(* _____ *) 
(* Define next-output function for the state machine *) 
(* _____ *)

```

```

val conductORPOut_def = 
Define 
` 

    (conductORPOut CONDUCT_ORP (exec x) = 
        if (getPlCom x) = secure then Secure else ConductORP) /\ 
    (conductORPOut SECURE (exec x) = 
        if (getPsgCom x) = actionsIn then ActionsIn else Secure) /\ 
    (conductORPOut ACTIONS_IN (exec x) = 
        if (getPlCom x) = withdraw then Withdraw else ActionsIn) /\ 
    (conductORPOut WITHDRAW (exec x) = 
        if (getPlCom x) = complete then Complete else Withdraw) /\ 
(* trapping *) 
    (conductORPOut (s:slState) (trap x) = unAuthorized) /\ 
(* discarding *) 
    (conductORPOut (s:slState) (discard x) = unAuthenticated)` 

(* *****) 
(* Input Authentication *) 
(* *****) 

```

```

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)` 

(* *****) 
(* "A theorem showing commands without a principal are rejected."--Prof Chin, SM0Script.sml *) 
(* *****) 

```

```

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

val _ = save_thm("inputOK_cmd_reject_lemma", 
    inputOK_cmd_reject_lemma) 

(* *****) 
(* PlatoonLeader is justified on secure. *) 
(* *****) 

```

```

val th1 = 
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 PlatoonLeader) says (prop (SOME (SLc (PL secure)))))) 

```

```

    :((slCommand command)option , stateRole , 'd,'e)Form`` ,
``ins:((slCommand command)option , stateRole , 'd,'e)Form list list`` ,
``(CONDUCT_ORP)``,
``outs:slOutput output list trType list``] TR_exec_cmd_rule

val temp = fst(dest_imp(concl th1))

val PlatoonLeader_CONDUCT_ORP_exec_secure_lemma =
TAC_PROOF(
[],  

  fst(dest_imp(concl th1))),  

REWRITE_TAC  

[CFGInterpret_def , secContext_def , secAuthorization_def ,  

 getOmniCommand_def ,  

 inputList_def , extractInput_def , MAP ,  

 propCommandList_def , extractPropCommand_def , satList_CONS ,  

 satList_nil , GSYM satList_conj] THEN  

PROVE_TAC[Controls])

val _ = save_thm("PlatoonLeader_CONDUCT_ORP_exec_secure_lemma" ,  

  PlatoonLeader_CONDUCT_ORP_exec_secure_lemma)

val PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm =
TAC_PROOF(  

[],  

  snd(dest_imp(concl th1))),  

PROVE_TAC[PlatoonLeader_CONDUCT_ORP_exec_secure_lemma , TR_exec_cmd_rule])

val _ = save_thm("PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm" ,  

  PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm)

val PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm =
REWRITE_RULE[inputList_def , extractInput_def , MAP , propCommandList_def ,  

 extractPropCommand_def , PlatoonLeader_CONDUCT_ORP_exec_secure_lemma]  

PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm

val _ = save_thm("PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm" ,  

  PlatoonLeader_CONDUCT_ORP_exec_secure_justified_thm)

(* ****)
(* PlatoonSergeant is justified on actionsIn if *)
(* if Omni says ssmSecureComplete. *)
(* ****)
val th1 =
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 ssmSecureComplete))))  

  :((slCommand command)option , stateRole , 'd,'e)Form;  

  (Name PlatoonSergeant) says (prop (SOME (SLc (PSG actionsIn))))  

  :((slCommand command)option , stateRole , 'd,'e)Form]`` ,
` `ins:((slCommand command)option , stateRole , 'd,'e)Form list list`` ,
` `(SECURE)` ,
` `outs:slOutput output list trType list``] TR_exec_cmd_rule

val PlatoonSergeant_SECURE_exec_lemma =
TAC_PROOF(
[],  

  fst(dest_imp(concl th1))),  

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("PlatoonSergeant_SECURE_exec_lemma" ,  

  PlatoonSergeant_SECURE_exec_lemma)

val PlatoonSergeant_SECURE_exec_justified_lemma =
TAC_PROOF(

```

```

([], snd(dest_imp(concl_th1))),
PROVE_TAC[PlatoonSergeant_SECURE_exec_lemma, TR_exec_cmd_rule])

val _ = save_thm("PlatoonSergeant_SECURE_exec_justified_lemma",
                 PlatoonSergeant_SECURE_exec_justified_lemma)

val PlatoonSergeant_SECURE_exec_justified_thm =
REWRITE_RULE[inputList_def, extractInput_def, MAP, propCommandList_def,
            extractPropCommand_def, PlatoonSergeant_SECURE_exec_lemma]
PlatoonSergeant_SECURE_exec_justified_lemma

val _ = save_thm("PlatoonSergeant_SECURE_exec_justified_thm",
                 PlatoonSergeant_SECURE_exec_justified_thm)

(* ****
(* PlatoonLeader is justified on withdraw if
*) if Omni says ssmActionsIncomplete.
(* ****)
val th1 =
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 ssmActionsIncomplete)))) :
((slCommand command)option , stateRole , 'd,'e)Form;
 (Name PlatoonLeader) says (prop (SOME (SLc (PL withdraw)))) :
((slCommand command)option , stateRole , 'd,'e)Form] ```,
 ``ins:((slCommand command)option , stateRole , 'd,'e)Form list list ```,
 ``(ACTIONS_IN)``,
 ``outs:slOutput output list trType list ```] TR_exec_cmd_rule

val PlatoonLeader_ACTIONS_IN_exec_lemma =
TAC_PROOF(
[], fst(dest_imp(concl_th1))),
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_ACTIONS_IN_exec_lemma",
                 PlatoonLeader_ACTIONS_IN_exec_lemma)

val PlatoonLeader_ACTIONS_IN_exec_justified_lemma =
TAC_PROOF(
[], snd(dest_imp(concl_th1))),
PROVE_TAC[PlatoonLeader_ACTIONS_IN_exec_lemma, TR_exec_cmd_rule])

val _ = save_thm("PlatoonLeader_ACTIONS_IN_exec_justified_lemma",
                 PlatoonLeader_ACTIONS_IN_exec_justified_lemma)

val PlatoonLeader_ACTIONS_IN_exec_justified_thm =
REWRITE_RULE[inputList_def, extractInput_def, MAP, propCommandList_def,
            extractPropCommand_def, PlatoonLeader_ACTIONS_IN_exec_lemma]
PlatoonLeader_ACTIONS_IN_exec_justified_lemma

val _ = save_thm("PlatoonLeader_ACTIONS_IN_exec_justified_thm",
                 PlatoonLeader_ACTIONS_IN_exec_justified_thm)

(* ****
(* PlatoonLeader is trapped on withdraw if
*) if not Omni says ssmActionsIncomplete.
(* ****)
val thTrap =
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 withdraw))))
  :((slCommand command)option , stateRole , 'd,'e)Form] ``,
``ins:((slCommand command)option , stateRole , 'd,'e)Form list list ``,
``(ACTIONS_IN)``,
``outs:slOutput output list trType list ``] TR_trap_cmd_rule

val temp2 = fst(dest_imp(concl thTrap))

val PlatoonLeader_ACTIONS_IN_trap_lemma =
TAC PROOF(
[], 
  Term`(~((omniCommand:omniCommand) = ssmActionsIncomplete)) ==>
    ((s:s1State) = ACTIONS_IN) ==>
    ^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_ACTIONS_IN_trap_lemma",
  PlatoonLeader_ACTIONS_IN_trap_lemma)

val temp3 = snd(dest_imp(concl thTrap))

val PlatoonLeader_ACTIONS_IN_trap_justified_lemma =
TAC PROOF(
[], 
  Term`(~((omniCommand:omniCommand) = ssmActionsIncomplete)) ==>
    ((s:s1State) = ACTIONS_IN) ==>
    ^temp3`),
DISCH_TAC THEN
DISCH_TAC THEN
PROVE_TAC[PlatoonLeader_ACTIONS_IN_trap_lemma, TR_trap_cmd_rule])

val _ = save_thm("PlatoonLeader_ACTIONS_IN_trap_justified_lemma",
  PlatoonLeader_ACTIONS_IN_trap_justified_lemma)

val PlatoonLeader_ACTIONS_IN_trap_justified_thm =
REWRITE_RULE[inputList_def, extractInput_def, MAP, propCommandList_def,
  extractPropCommand_def,
  PlatoonLeader_ACTIONS_IN_trap_lemma]
  PlatoonLeader_ACTIONS_IN_trap_justified_lemma

val _ = save_thm("PlatoonLeader_ACTIONS_IN_trap_justified_thm",
  PlatoonLeader_ACTIONS_IN_trap_justified_thm)

(* ===== Interactive Mode =====
===== Interactive Mode ===== *)
val _ = export_theory();
end

```


E.4.4 ssmMoveToPB

E.4.4.1 MoveToPBType Theory: Type Definitions

E.4.4.2 MoveToPBDef Theory: Authentication & Authorization Definitions

E.4.4.3 ssmMoveToPB Theory: Theorems

E.4.5 ssmConductPB

E.4.5.1 ConductPBType Theory: Type Definitions

E.4.5.2 ConductPBDef Theory: Authentication & Authorization Definitions

E.4.5.3 ssmConductPB Theory: Theorems

E.5 Vertical Slice

E.5.1 ssmSecureHalt

E.5.1.1 SecureHaltType Theory: Type Definitions

E.5.1.2 SecureHaltDef Theory: Authentication & Authorization Definitions

E.5.1.3 ssmSecureHalt Theory: Theorems

E.5.2 ssmORPRecon

E.5.2.1 ORPReconType Theory: Type Definitions

E.5.2.2 ORPReconDef Theory: Authentication & Authorization Definitions

E.5.2.3 ssmORPRecon Theory: Theorems

375

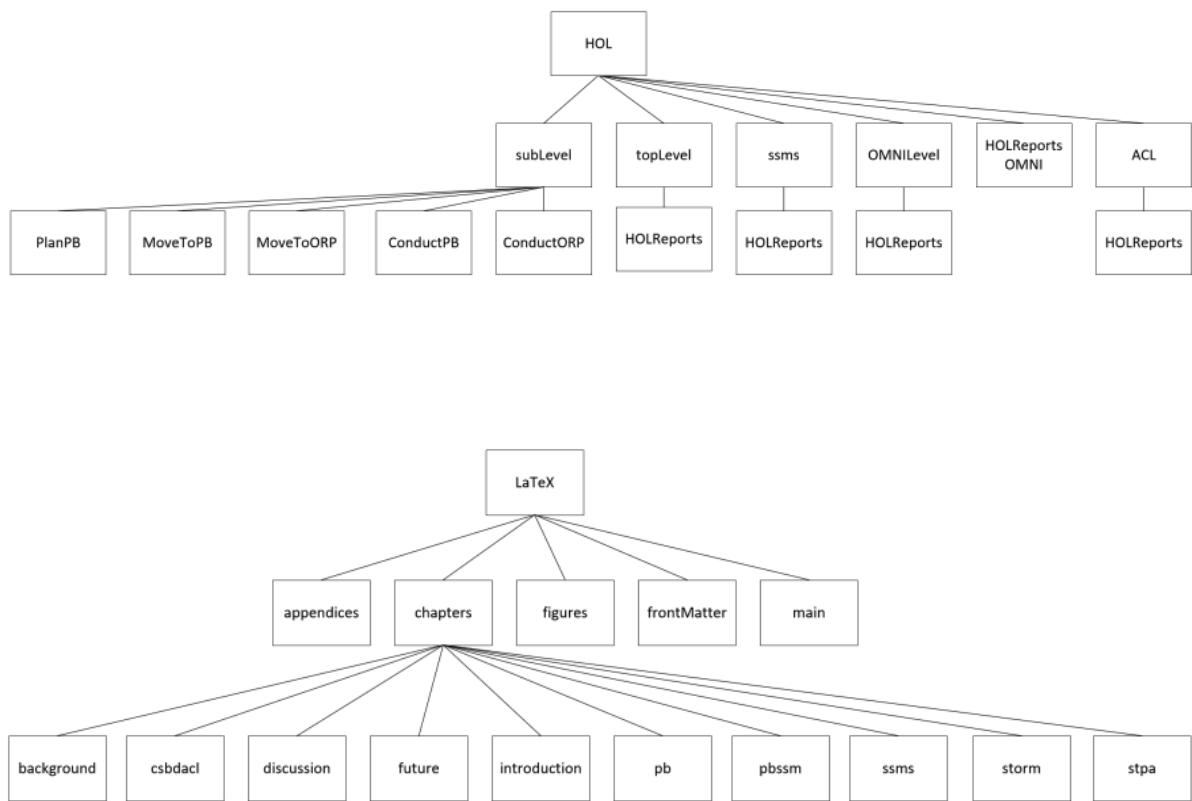
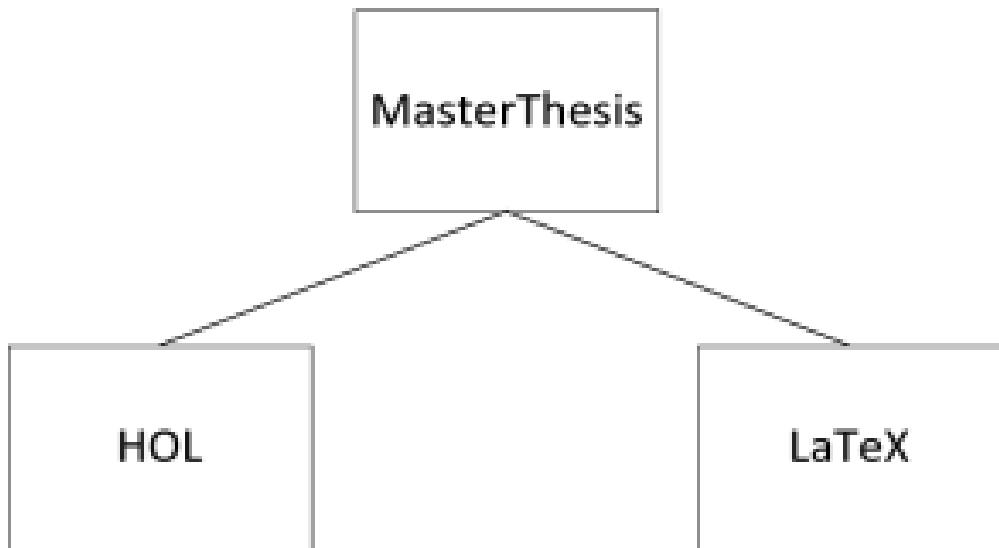
E.5.3 ssmMoveToORP4L

Appendix F

Map of The File Folder Structure

Compile the files by opening a terminal, navigating to the MasterThesis folder, and typing `make` and then enter.

A map of the folder structure for this project is shown below.



References

- [1] R. Ross, M. McEvilley, and J. C. Oren, “Systems security engineering considerations for a multidisciplinary approach in the engineering of trustworthy secure systems,” National Institute of Standards and Technology (NIST), 100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930, Special Publication 800-160, May 2016. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/sp/800-160/archive/2016-05-04/documents/sp800_160_second-draft.pdf
- [2] W. Y. Jr and R. Porada. (2017, March) System-theoretic process analysis for security (stpa-sec): System-theoretic process analysis for security (stpa-sec): Cyber security and stpa. 2017 STAMP Conference, Boston, MA, March 27, 2017. [Online]. Available: http://psas.scripts.mit.edu/home/wp-content/uploads/2017/04/STAMP_2017_STPA_SEC_TUTORIAL_as-presented.pdf
- [3] N. G. Leveson and J. P. Thomas, *STPA Handbook*, March 2018.
- [4] N. G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2012.
- [5] *Systems thinking for safety and security*. New Orleans: Proceedings of the 2013 Annual Computer Security Applications Conference (ACSAC 2013), December 2013.
- [6] S.-K. Chin and S. B. Older, *Access Control, Security, and Trust: A Logical Approach*, ser. Chapman & Hall: CRC Cryptography and Network Security Series. Chapman and Hall/CRC, July 2010. [Online]. Available: <https://www.crcpress.com/Access-Control-Security-and-Trust-A-Logical-Approach/Chin-Older/p/book/9781584888628>
- [7] S.-K. Chin and S. Older, *Certified Security by Design Using Higher Order Logic*. Syracuse University, Syracuse, New York 13244: Department of Electrical Engineering and Computer Science Syracuse University, November 2017, vol. Version 1.5.
- [8] *Introduction to Programming Languages/Algebraic Data Types*. used under the Creative Commons Attribution-ShareAlike 3.0 License, August 2017. [Online]. Available: https://en.wikibooks.org/wiki/Introduction_to_Programming_Languages/Algebraic_Data_Types

- [9] *Ranger handbook*, United States Army Ranger School, ATTN: ATSH-RB, 10850 Schneider Rd, Bldg 5024, Ft Benning, GA 31905, April 2017.
- [10] D. P. Berg and W. M. M. Carnahan, “The kingdom,” Motion Picture Movie, September 2007.
- [11] “ISO/IEC/IEEE 24748-5 international standard - systems and software engineering—life cycle management—part 5: Software development planning,” 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, Tech. Rep. ISO/IEC/IEEE 24748-5:2017(E), May 2017.
- [12] “ISO/IEC/IEEE 15288 international standard - systems and software engineering – system life cycle processes,” 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, Tech. Rep. ISO/IEC/IEEE 15288:2015(E), May 2015. [Online]. Available: <https://www.ieee.org>
- [13] R. Abrams. (2017, May) Target to pay \$18.5 million to 47 states in security breach settlement. [Online]. Available: <https://www.nytimes.com/2017/05/23/business/target-security-breach-settlement.html>
- [14] J. S. protocols Glenn Benson, S.-K. Chin, S. Croston, K. Jayaraman, and S. Older, “Banking on interoperability: Secure, interoperable credential management,” *Computer Networks*, vol. 67, pp. 235–251, 2014.
- [15] P. Stukus, “Systems-theoretic accident model and processes (stamp) applied to a u.s. coast guard buoy tender integrated control system,” 2017.
- [16] *Simultaneous Analysis of Safety and Security of a Critical System*, September 2017. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2017/09/simultaneous-analysis-of-safety-and-security-of-a-critical-system.html
- [17] Wikipedia. (2018, January) Availability. [Online]. Available: <https://en.wikipedia.org/wiki/Availability>
- [18] J. H. Saltzer and M. D. Schroeder. (1975, April) The protection of information in computer systems. [Online]. Available: <http://www.cs.virginia.edu/~evans/cs551/saltzer/>
- [19] U. Army. U.s. army organization: Mission. [Online]. Available: <https://www.army.mil/info/organization/>
- [20] T. R. Corporation, “Security controls for computer systems: Report of defence science board task force on computer security,” Office of the Director of Defence Research And Engineering, Washington D.C. 20301, Tech. Rep., February 1970.
- [21] (2018, May) Saul kripke. [Online]. Available: https://en.wikipedia.org/wiki/Saul_Kripke

- [22] M. Collins, "Formal methods: 18-849b dependable embedded systems," Carnegie Mellon University, Tech. Rep., Spring 1998. [Online]. Available: https://users.ece.cmu.edu/~koopman/des_s99/formal_methods/
- [23] E. M. Clarke and e. a. Jeannette M. Wing. (1996, December) Formal methods: State of the art and future directions. Pittsburgh, PA. [Online]. Available: <http://homepage.cs.uiowa.edu/~tinelli/classes/181/Fall15/Papers/Clar96.pdf>
- [24] A. P. Sistla, V.N.Venkatakrishnan, M. Zhou, and H. 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*, March 2008, pp. 100–111.
- [25] T. Hiroki, M. Yoshiaki, and H. Yuusuke, "C-language verifiction tool using formal methods "varvel"." [Online]. Available: <https://pdfs.semanticscholar.org/c9af/cbdf6f37ffce7d13405358f136641cde1f87.pdf>
- [26] Z. Xin-feng, W. Jian-dong, L. Bin, Z. Jun-wu, and W. Jun, "Methods to tackle state explosion problem in model checking," in *2009 Third International Symposium on Intelligent Information Technology Application*, no. ISBN: New-2005 _ POD _ 978-0-7695-3859-4. IEEE, December 2009, pp. 329–331.
- [27] "Formal methods," *Wikipedia*, February 2018. [Online]. Available: https://en.wikipedia.org/wiki/Formal_methods
- [28] D. Turner and Y. Welsch. Reliable by design: Applying formal methods to distributed systems. [Online]. Available: <https://www.elastic.co/elasticon/conf/2018/sf/reliable-by-design-applying-formal-methods-to-distributed-systems>
- [29] J. Kunasaikaran, A. Iqbal, J. Dua, and C. S. Lin. (2016) A brief overview of functional programming languages. [Online]. Available: https://www.researchgate.net/publication/311493362_A_Brief_Overview_of_Functional_Programming_Languages
- [30] M. Fluet. (2015, January) Typeconstructor. [Online]. Available: <http://mlton.org/TypeConstructor>
- [31] Wikipedia. (2018, May) Bsd licenses. [Online]. Available: [https://en.wikipedia.org/wiki/BSD_licenses#4-clause_license_\(original_"BSD_License"\)](https://en.wikipedia.org/wiki/BSD_licenses#4-clause_license_(original_)
- [32] Hol interactive theorem prover. [Online]. Available: <https://hol-theorem-prover.org>