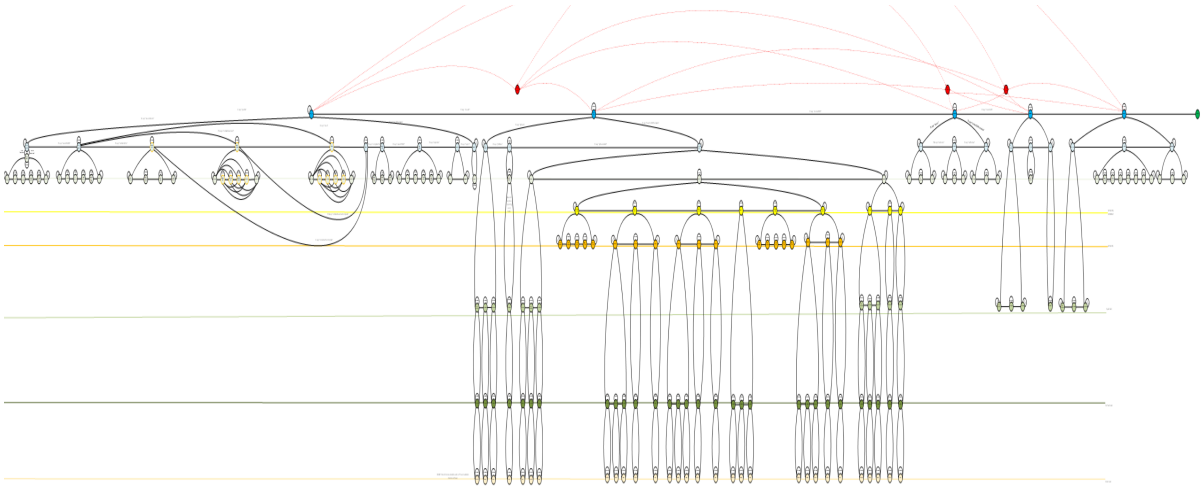

Figure 1: This is a caption for this figure.

Abstract

Copyright

Disclaimer

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

Acknowledgements

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

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

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

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

Table Of Contents

List of Figures

List of Tables

List of Acronyms

ACL access-control logic.

CSBD Certified Security by Design.

WFF well-formed formula.

Chapter 1

Introduction

1.1 Motivation

1.1.1 Systems Are Everywhere

1.2 This Master Thesis

This master thesis describes a method for designing secure systems. The method is called Certified Security by Design (CSBD). CSBD has been successfully demonstrated on automated systems such as ... and .... But, until this research, it has not been demonstrated on non-automated, human-centered systems.

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

- The first question addressed in this master thesis is whether CSBD could be successfully applied to non-automated, human-centered systems. This is the

primary objective. An example of a non-automated, human-centered system is the patrol base operations defined in the United States Army Ranger Handbook[1]. Patrol base operations exemplify a non-automated, human-centered system wherein security is critical to mission success. In this master thesis, the results of applying CSBD to patrol base operations is discussed.

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

- These thesis describes a hierarchy of secure state machines (SSMs) used to model the patrol base operations. This approach demonstrates that formal methods can be applied to large scale and complicated systems. The hierarchy manages patrol base operations by successful levels of decreasing abstraction. Each level in the hierarchy consists of one or more SSMs. Each SSM is modularized and models one aspect of the patrol base operations at one level of abstraction. The levels and modules are connected together by an OMNI level, all-seeing, principal. Each module only needs to be aware of this OMNI level principal. They do not need to be aware of the details of any other module. With this divide-and-conquer approach, CSBD can be readily applied to large and complicated systems.

- The successful application of CSBD to patrol base operations also suggests its use in combining automation with human-centered systems. The approach employed by this master thesis involves describing the patrol base operations as a hierarchy of secure state machines. This hierarchy has the property that it is easy to demonstrate security properties of the system, which is the goal of CSBD. But, it also has the property that it describes the patrol base operations as a system that is amiable to automation. Such automations of pre-defined non-automated,

human-centered systems could include, for example, accountability systems for tracking supplies and personal.

In the not-so-distant future, the military, in particular, will most likely seek tracking and accountability systems for pre-existing, non-automated military operations. These systems, like all security-sensitive military systems, should be designed according to NIST 800-160 standards. These standards require the formal verification and documentation provided by CSBD and demonstrated in this thesis.

Chapter 2

Background

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

2.1 Complete Mediation

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

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

> *proposals to gain performance by remembering the result of an authority check be examined skeptically. If a change in authority occurs, such remembered results must be systematically updated.* [4]

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

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

2.2 Confidentiality, Integrity, and Availability

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

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

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

Availability refers to accessibility. Wikipedia describes availability as a measure of operability or degree to which the system is mission capable [5].

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

2.3 Formal Methods

(Primary source for this section is [6])

Formal methods are aimed at improving the reliability and correctness of systems[7]. 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 process[2].

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 [6]. Specification consists of describing the system in a mathematical-based modeling language [6]. Verification entails proving properties of the system, typically using either model checking or theorem proving techniques. Implementation depends on the type of system (i.e., software, hardware, human-centered system, etc.).

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

[2]For example, there is notable progress being made in formally verifying c code. See for example [8] and [9]

the complexity of the system [10].

Theorem proving, on the other hand, employs a formal logic to prove properties of the system. Formal proofs remain true for any test case [6]. 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 [11]. It often requires specialized knowledge and a sufficient degree of competency in mathematics[3]. However, formal theorem proving techniques have been successfully taught to undergraduates[4].

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 [12].

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[6].

Specification has its benefits, not only as a precursor to verification, but also as a tool for understanding. In cases wherein properties of the system are not proved, the act of formally specifying the system adds a degree of rigor to the process. This rigor often brings new insights about the system because it causes the engineer to think more systematically about the system's design. In addition, the conversion of a field's jargon into a precise specification language also aids in reproducibility and communicability[5]

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

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

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

[6].

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

2.4 Functional Programming

(Primary source for this section is [13])

Functional programing is a style of programming that uses functions to define program behavior. Functional programing is inherently different than procedural or object-oriented programming. These styles of program use procedures or objects and classes to define program behavior. c and Pascal are examples of procedural programming languages. c++ and Java are examples of object-oriented programming languages. Haskell and ML[6] (meta language) are examples of functional programming languages.

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

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

[6]ML is not considered a purely functional.

2.5 Algebraic Data Types in ML

The primary source for this section is *Introduction to Programming Languages/Algebraic Data Types* [14]. This section is helpful in understanding section 4.3. The reader may skip this section and return to it before reading that section.

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

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

A note on type constructors: in ML[7] a type constructors should be thought of as a function that takes a type as its parameter and returns a type. In the declaration `Node of 'a tree * 'a * 'a tree`. `Node` behaves like a function which returns something of type `'a tree` [?].

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

Figure 2.1: An implementation of the boolean datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [14]

.

The boolean datatype can be thought of as the union of the 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

[7]Standard ML according to the author [?].

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

Figure 2.2: An implementation of a tree datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [14]

.

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

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

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

Figure 2.3: An implementation of the weekday datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [14]

.

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

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

Figure 2.4: An implementation of the function is_weekend in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [14]

.

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

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

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

Figure 2.5: A parameterized implementation of a tree datatype in ML. Image from *Introduction to Programming Languages/Algebraic Data Types* [14]

.

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

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

2.6 Higher Order Logic (HOL) Interactive Theorem Prover

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

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

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

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

The version of HOL used for this master thesis is HOL4. HOL4 is free software that is BSD licensed [15]. Download instructions can be found at the HOL website hol-theorem-prover.og [16].According to Wikipedia, HOL4 is descended from HOL88. The HOL88 project is Mike Gordon's effort.. HOL4 is implemented in polyML which is an implementation of Standard ML [17].

2.7 Other Interactive Theorem Provers

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

For example, Isabelle/HOL[8] is another popular interactive theorem prover. The access-control logic (ACL)has been partially implemented in Isabelle/HOL by Scott Constable, a PhD student in the College of Engineering and Computer Science and Syracuse University.

2.8 How to Compile The Included Files

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

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

Chapter 3

Systems Security Engineering & Patrol Base Operations

3.1 The Systems Perspective

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

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

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

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

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

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

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

3.2 Systems Engineering

The recognized authoritative standard on systems engineering is ISO/IEC/IEEE 15288 [18]. This document is titled "Systems And Software Engineering–System Life Cycle Processing." A precursor to this standard is ISO/IEC TR 24748 1 [19] titled "Systems And Software Engineering–Life Cycle Management–Part 1: Guide for Life Cycle Management." These are the primary sources for this section.

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

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

Concept The concept phase describes stakeholders and the Concept of Operations (ConOps). Stakeholders are those who have a stake in the system. ConOps are defined in a variety of ways. Wikipedia defines ConOps as "...a document describing the characteristics of a proposed system from the viewpoint of an individual who will use that system [20]. U.S. Air Force Policy Directive 10-28 defines ConOps as "a high level concept whose purpose is to describe operational mission areas, enablers and effects necessary to achieve desired results/outcomes" [21]. ConOps are context specific.

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

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

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

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

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

Production Production is self-explanatory, however not so obvious with a system such as the patrol base operations. There are no products save for the trained soldier. In this aspect, the production phase would entail educating the soldiers. Neither this master thesis nor CSBD cover[1] education of soldiers. (Of course, one could imagine a lecture on the importance of authentication and authorization as a critical component of any soldier's primary training.)

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

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

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

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

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

3.3 NIST Special Publication 800-160

One work in the field of Systems Security Engineering is such a critical component to the field of SSE that it warrants pointing out. The National Institute of Standards and Technology (NIST) defines its mission as such: "To promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve our quality of life." [22]. As their name suggests, NIST develops standards for various industries of relevance to the nation.

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

3.4 Systems Security Engineering

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

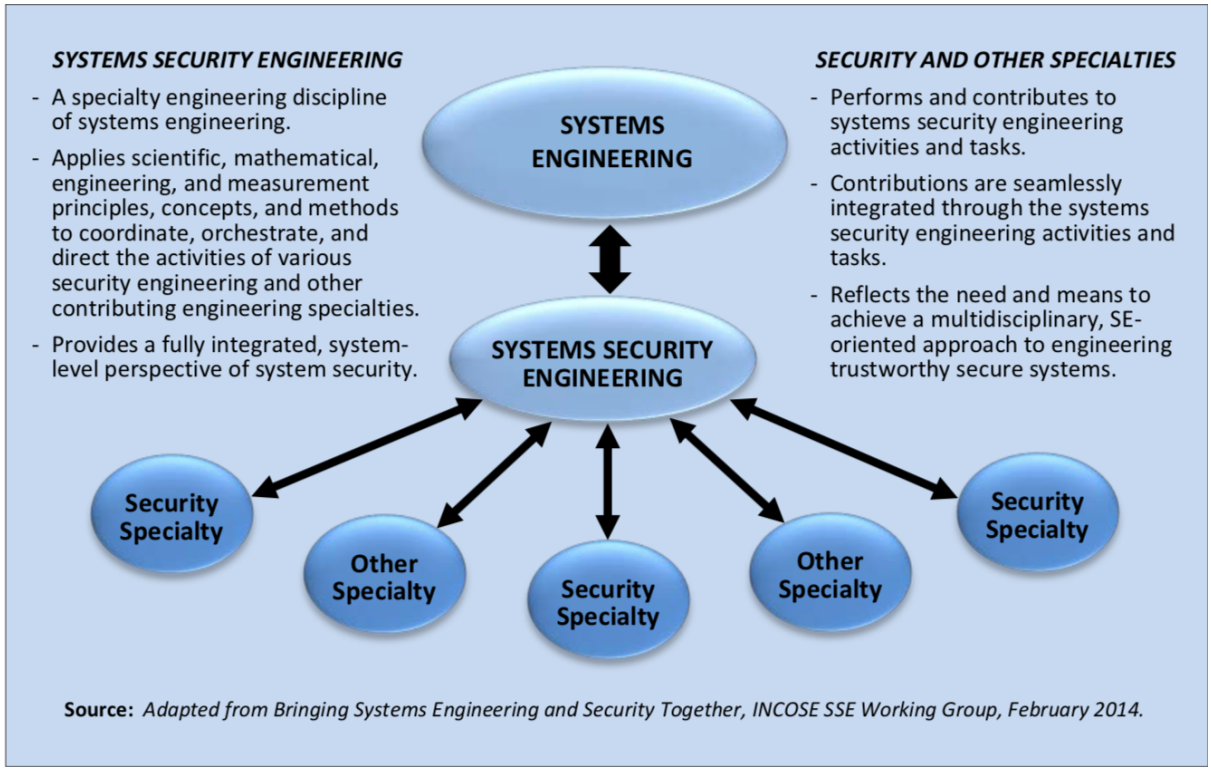

Figure 3.1: Systems security engineering in relation to systems engineering. (Image from NIST Special Publication 800-160: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.)

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

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

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

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

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

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

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

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

3.5 Systems Security Engineering Framework

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

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

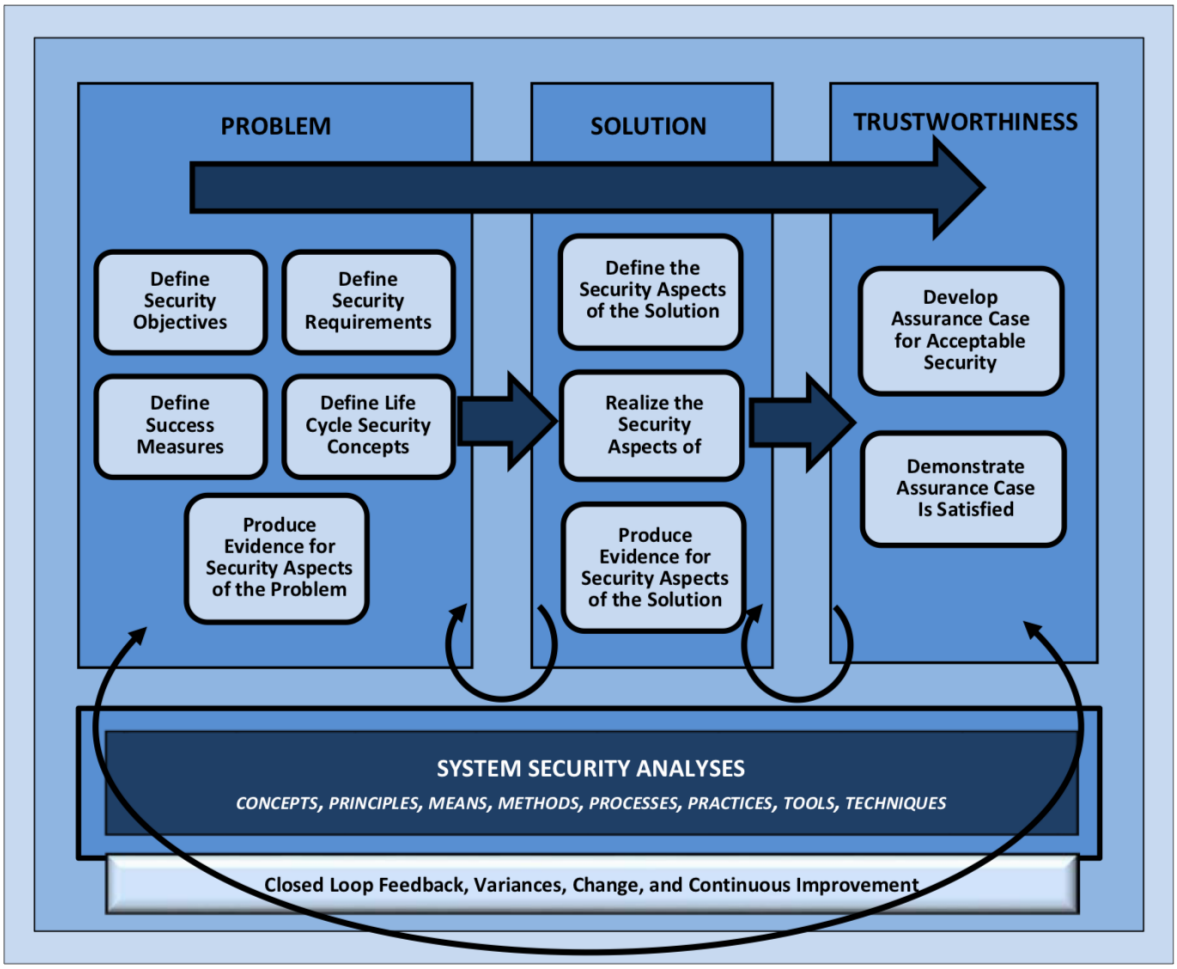

Figure 3.2: Systems security engineering Framework. (Image from NIST Special Publication 800-160: Systems Security Engineering Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems.)

3.5.1 Problem

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

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

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

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

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

3.5.2 Solution

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

SSMs are readily defined with complete mediation embedded into their design. SSMs are discussed in detail in Chapter 6[2].

The objects to be accessed varies at each level of abstraction. At the top and most abstract level, the objects are the transitions. That is, control over the transitions between phases (or states) of the patrol base operations are to be protected. Objects are defined similarly for each module and at each level for the entire hierarchy of the model. These are deducible from the descriptions of each module (or SSM) in their appropriate section (see section 5.5.2 to start). For the sake of brevity, this section only

[2]In fact, a parametrizable model of an SSM[3] was already designed and implemented in HOL prior to the start of this project. The constraints of this project suggested Improvements[4] to this parametrizable SSMMuch to the chagrin of the author, a significant portion of the SSMs had to be redone with the new parametrizable SSM.

discusses the top level to illustrate the key concepts.

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

In the real patrol base operations, soldiers recognize each other by face. Thus, it is sufficient to just authenticate the platoon leader at each level without requiring any passwords or other means of authentication[5].

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

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

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

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

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

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

if OMNI-Level says the previous phase is complete[6]*, and*
Platoon Leader says move to the next phase
then Platoon Leader has the authority to move to the next phase

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

if Platoon Leader's identity is authenticated, and
Platoon Leader says move to the next phase, and
Platoon Leader has the authority to move to the next phase
then move to the next phase.

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

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

3.5.3 Trustworthiness

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

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

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

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

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

assurance cases,

3.6 Verification & Documentation

3.6.1 Verification

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

3.6.2 Documentation

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

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

3.6.2.1 Accountability

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

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

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

3.6.2.2 Reproducibility

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

Chapter 4

Certified Security by Design (CSBD) & Access-Control Logic (ACL)

4.1 Certified Security by Design (CSBD)

In 1970 The Rand Corporation published a report[23] for the Office of The Director of Defense Research And Engineering. This report titled, Security Controls For Computer Systems, noted that "Providing satisfactory security controls in a computer system is in itself a system design problem." NIST 800-160 also highlights the importance of incorporating security into the design phase of the system engineering process. CSBD focuses on the design phase of systems engineering, applying formal methods to verify that a system satisfies the principle of complete mediation.

More specifically, CSBD is a method for formally verifying and documenting the security properties of a systems. It focuses on designing systems that satisfy the principle of complete mediation. It uses an access-control logic (ACL) to reason about

access to security sensitive objects of a system. It uses computer-aided reasoning such as the Higher Order Logic (HOL) Interactive theorem prover to formally verify and document these security properties. The outcomes of CSBD applied to a system conform to the guidelines set fourth in NIST 800-160 [verify and discuss this.]

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

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

4.2 Access-Control Logic (ACL)

4.2.1 ACL: A Command and Control (C2) Calculus

ACL is a logic for reasoning about access to object. In the jargon of the day it is a command and control (C2) calculus[1]. All the axioms and definitions described above are implemented in the ACL. The actual code is included with the files for this thesis. A pretty-printed print-out of the ACL is included in appendix A.

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

4.2.2 Principals

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

Princ ::= PName / Princ & Princ / Princ |Princ

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

4.2.3 Propositional Variables, Requests, Authority, and Jurisdiction

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

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

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

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

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

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

4.2.4 Well-formed Formulas

Well-formed formulas (WFFs) define the syntax (or grammatical structure) of the propositional logic. They are valid statements in the ACL. All statements must be a WFF. The text book defines the set of WFFs using the identifier **Form**:

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

(**Form** $\wedge$ **Form**) / (**Form** $\supset$ **Form**) / (**Form** $\equiv$ **Form**) /

(**Princ** $\Rightarrow$ **Princ**) / (**Princ** says **Form**) / (**Princ** controls **Form**) /

(**Princ** reps **Princ** on **Form**[2])

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 [24].

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

4.2.5.1 Kripke Structures

Before defining the Kripke semantics, the concept of a Kripke structure is necessary. A Kripke structure primarily deals with three things: worlds, propositions, and principals. The worlds can be thought of as possible states or configurations of some system. Propositions are just statements that are either true or false. And, principals are just actors. A Kripke structure $\mathcal{M} = \langle W, I, J \rangle$ is defined as a three-tuple consisting of the following: a set of worlds W; a function I called the assignment function that maps propositions to worlds, and ; a function J that maps principals to relations on worlds,

[2]The last line is from [3].

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

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

One way to think of Kripke structures is as a logic on multiple worlds or possible states. For example, consider two planets Earth and Tatooine[3]. Let φ be the proposition "this is planet Tatooine" and θ be the proposition "this is a planet." Then $I(\varphi) = \{Tatooine\}$ and $I(\theta) = \{Earth, Tatooine\} = W$, where W is the set of all worlds ({Earth, Tatooine}). Furthermore, let "Luke" and "Lea" and "Uncle Owen" be principals. Now consider the following: Luke can get from Earth to Earth and from Tatooine to Earth (and vis-a-vis). He can also get from Tatooine to Tatooine. Lead can get from Earth to Earth and from Tatooine from Earth. But, she can not get from Tatooine to Tatooine. Uncle Owen can only get from Tatooine to Tatooine. He can not get to Earth. Then, $J(Luke) =$ $\{(Earth, Earth), (Earth, Tatooine), (Tatooine, Earth), (Tatooine, Tatooine)\}$. $J(Lea) = \{(Earth, Earth), (Earth, Tatooine), (Tatooine, Earth)\}$. And, $J(UncleOwen) = \{(Tatooine, Tatooine)\}$. With W, J, and I defined, this forms a Kripke structure.

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

[3]Tatooine is the fictional home planet of Luke Skywalker. Tatooine has two suns. [25]

4.2.5.2 Kripke Semantics

The Kripke semantics define the meanings of 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*[2]

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

4.2.5.3 Satisfies

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

[4]Think about it.

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

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

4.2.5.4 Soundness

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

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

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

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

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

4.2.6 Inference Rules

The inference rules for the access-control logic (ACL) are shown in figure 4.2. All the inference rules are sound. Details of proofs of soundness can be found in *Access Control, Security, and Trust: A Logical Approach*[2].

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

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

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

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

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

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

Figure 4.2: The ACL inference rules. Image taken from *Access Control, Security, and Trust: A Logical Approach*[2]

4.2.7 Complete mediation

Fundamental to this work is the concept of complete mediation (discussed in section **??**). In the ACL, this means that each principal must be authenticated and authorized on each request. ACL does this primarily by the *Controls* inference rule in figure 4.2 and shown again here in figure 4.3.

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

Figure 4.3: The *Controls* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[2]

ACL refers to the left statement as an authorization[5]. The principal P controls (is authorized on) some action φ. ACL refers to the right statement in this inference rule as a request [6]. The principal P requests some action φ. The conjunction of the authorization and the request of P on φ results in the action φ. That is, if *P controls* φ and *P says* φ then φ is true.

[5] or a *control* in the C2 calculus
[6] or a *command* in the C2 calculus

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

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

Figure 4.4: The *Reps* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[2]

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

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

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

Figure 4.5: The *Derived Speaks For* inference rule. Image taken from *Access Control, Security, and Trust: A Logical Approach*[2]

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

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

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

4.3 ACL in HOL

The material in this section is adapted from *Access Control, Security, and Trust: A Logical Approach*[2] and *Certified Security by Design Using Higher Order Logic*[3]. In this section, the former is referred to as "the text" and the later is referred to as "the manual."

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

4.3.1 Principals

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

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

Figure 4.6: The HOL implementation of principle (***Princ***). Image from *Certified Security by Design Using Higher Order Logic*[3]

.

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

The first thing to notice about ***Princ*** is that there are three definitions just as there are in the definition of ***Princ*** in section 4.2.2. The first line within the definition above corresponds to ***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.

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

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

$$NamePlatoonLeader, or$$

$$(NamePlatoonLeader)\text{`}meet\text{`}(NamePlatoonSergeant), or$$

$$(NamePlatoonLeader)\text{`}quoting\text{`}(NamePlatoonSergeant).$$

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

4.3.2 Well-Formed Formulas

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

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

Figure 4.7: The definition for **Form** in HOL. *Certified Security by Design Using Higher Order Logic*[3]

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

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

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

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

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

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

4.3.3 Kripke structures

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

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

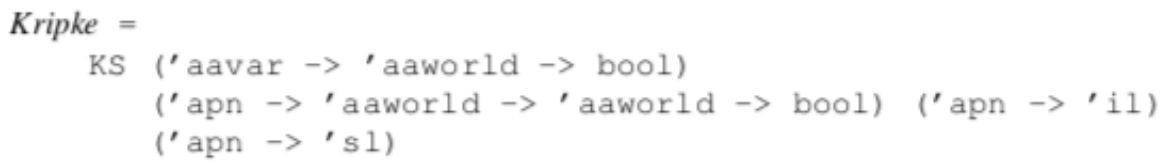

Figure 4.8: The HOL implementation of a Kripke structure (***Princ***). Image from *Certified Security by Design Using Higher Order Logic*[3]

.

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

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

$$(M, Oi, Os)$$

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

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

4.3.4 ACL Formulas

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

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

Figure 4.9: The ACL formulas in HOL. Image taken from *Access Control, Security, and Trust: A Logical Approach*[2]

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

```
prop command
```

A request in HOL is coded as:

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

Parentheses are used when necessary[7]. 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.

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

4.3.5 Kripke Semantics: The Evaluation Function

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

```
[Efn_def]
⊢ (∀Oi Os M. Efn Oi Os M TT = 𝒰(:'v)) ∧
  (∀Oi Os M. Efn Oi Os M FF = {}) ∧
  (∀Oi Os M p. Efn Oi Os M (prop p) = intpKS M p) ∧
  (∀Oi Os M f.
     Efn Oi Os M (notf f) = 𝒰(:'v) DIFF Efn Oi Os M f) ∧
  (∀Oi Os M f1 f2.
     Efn Oi Os M (f1 andf f2) =
     Efn Oi Os M f1 ∩ Efn Oi Os M f2) ∧
  (∀Oi Os M f1 f2.
     Efn Oi Os M (f1 orf f2) =
     Efn Oi Os M f1 ∪ Efn Oi Os M f2) ∧
  (∀Oi Os M f1 f2.
     Efn Oi Os M (f1 impf f2) =
     𝒰(:'v) DIFF Efn Oi Os M f1 ∪ Efn Oi Os M f2) ∧
  (∀Oi Os M f1 f2.
     Efn Oi Os M (f1 eqf f2) =
     (𝒰(:'v) DIFF Efn Oi Os M f1 ∪ Efn Oi Os M f2) ∩
     (𝒰(:'v) DIFF Efn Oi Os M f2 ∪ Efn Oi Os M f1)) ∧
```

Figure 4.10: The HOL implementation of the Kripke semantics (evaluation function). Image from *Certified Security by Design Using Higher Order Logic*[3]

4.3.6 Satisfies

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

The first part of the definition is `∀ M Oi Os f.`. This is the universal quantifier applied to all Kripke structure, intgegrity levels, security levels, and functions. The next

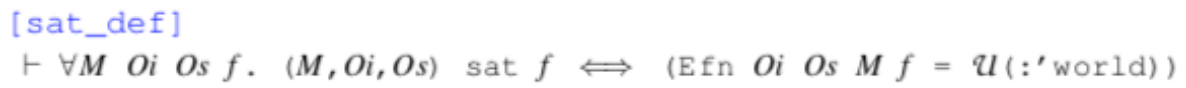

Figure 4.11: The HOL implementation of the "satisfies" property. Image from *Certified Security by Design Using Higher Order Logic*[3]

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

4.3.7 Soundness

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

$$\vdash \forall M\ O_i\ O_s.(M,O_i,O_s)\ sat\ H_1 \Rightarrow \cdots \Rightarrow (M,O_i,O_s)\ sat\ H_k \Rightarrow (M,O_i,O_s)\ sat\ C,$$

Figure 4.12: The HOL representation of the “soundness" property. Image from *Certified Security by Design Using Higher Order Logic*[3]

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

Chapter 5

Patrol Base Operations

5.1 Motivation

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

5.2 Patrol Base Operations

Patrol base operations are described in the United States Army Ranger Handbook.

> *A patrol base is a position established when a patrol halts for an extended period of time, by means of securing and occupying an appropriate location in order to (1) avoid detection, (2) hide, (3) maintain weapons and equipment, (4) eat and rest, (5) plan and issue orders, and (6) conduct operations* [Ranger Manual, 5-30].

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

5.3 Modeling the Patrol Base Operations from the Ranger Handbook

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

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

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

5.4 Overview of The Hierarchy of Secure State Machines

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

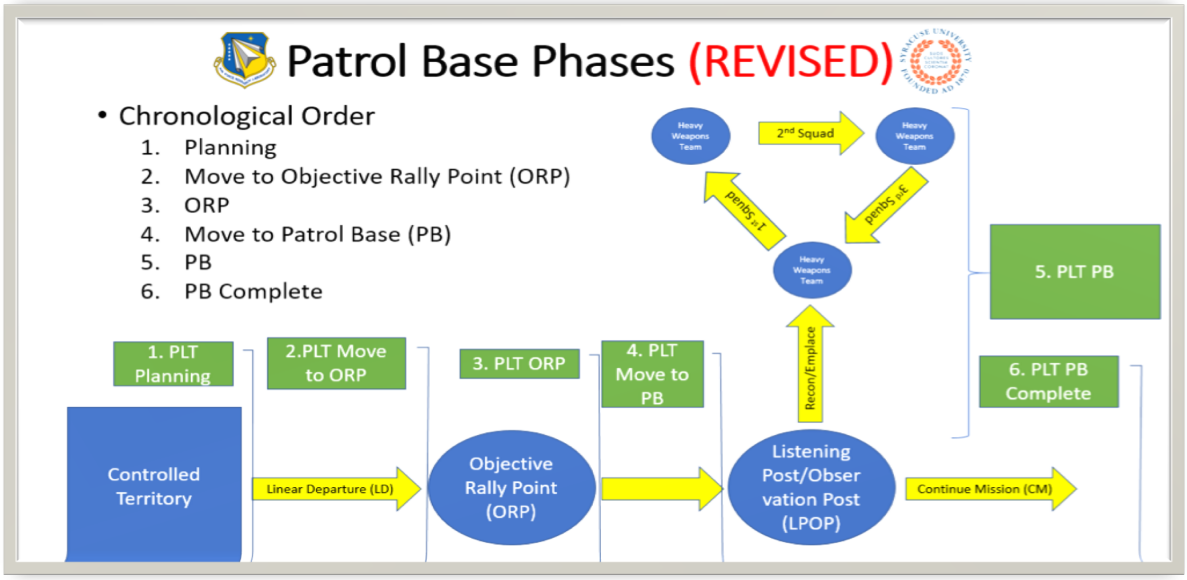

Figure 5.1: A diagram of the most abstract level in the hierarchy of secure state machines. Image generated by Jesse Nathaniel as part of the research involved in this master thesis. Shall I get his permission?

The diagram describes a chronological order of abstract phases (modeled as states) of the patrol base operations. The operations begin with the planning phase (1). Next, they move to the objective rally point (ORP) (2). At the ORP, operations commence (3). When these are complete, the patrol base operations move to the actual patrol base (4). At the patrol base, operations proceed (5). Finally, the patrol base operations are complete (6). These are the six states in the top level SSM.

The next level of abstraction in the hierarchy of SSMs represents a horizontal slice through the patrol base operations. This is the second level of the hierarchical description of the patrol base operations. It is referred to as the sub level. In this

documentation, SSMs at this level are referred to as the sub-level, sublevel, or subLevel SSMs. This slice describes the patrol base operations at a lower level of abstraction. It expands each of the states in the top level (except for the last state PB Complete). For example, the planning phase (1) in figure 5.1 is expanded into an SSM of its own. This is called ssmPlanPB. It consists of several states (see section 5.5.5.1) which detail activities conducted during the planning phase of the patrol base operations. Each state in the top level (except for PB Complete) has it's own SSM (see the next section).

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

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

The vertical slice begins at the top level SSM. Next, it expands upon one state at this level, the *move to ORP* state (2). This results in a sub level SSM named ssmMoveToORP. The vertical slice progresses in this manner, by expanding one state at each level into a new SSM. From ssmMoveToORP, the state *secure halt* is expanded to ssmSecureHalt. From within this SSM, the state *ORP Recon* is expanded into ssmORPRecon. From within this SSM, the state *Move to ORP 4L* (fourth level move to ORP state) is expanded into ssmMoveToORP4L. Finally, from within this SSM, the state *Form RT* is expanded into ssmFormRT.

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

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

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

Note that, the purpose of this master thesis is to demonstrate that the properties of complete mediation could be applied and verified on a non-automated, human-centered systems. Thus, it is sufficient to demonstrate this on a horizontal and vertical slice of the patrol base operations.

5.5 Hierarchy of Secure State Machines

5.5.1 Diagrammatic Description in Visio

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

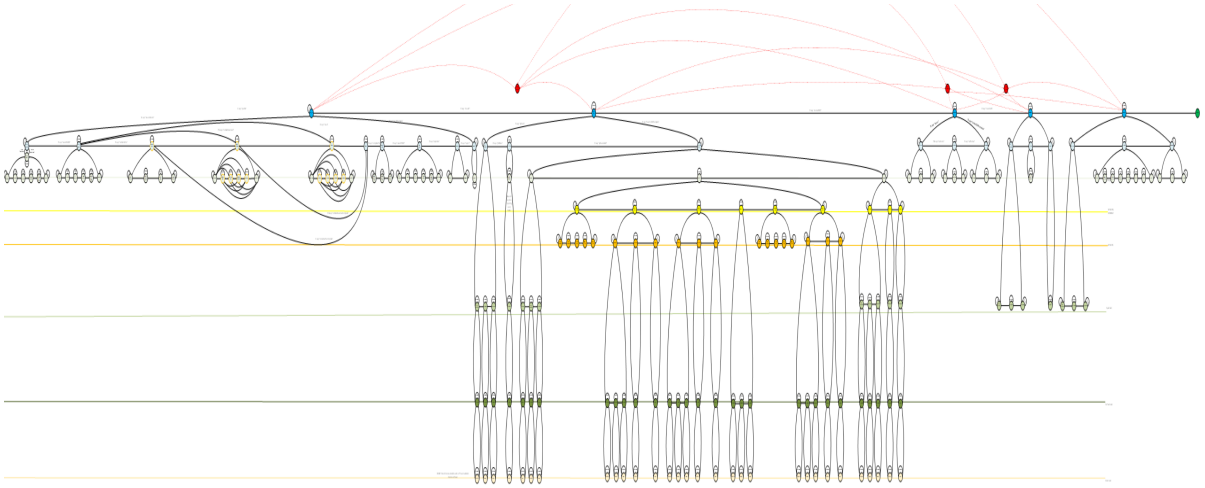

Figure 5.2: Diagrammatic description of patrol base operations as a hierarchy of secure state machines. (Generated by Jesse Nathaniel Hall.)

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

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

In figure 5.2, the red dots at the top of the diagram represent the escape level SSM. But, they do not represent states in the SSM. This is because the escape level SSM is accessible by all phases of the patrol base operations. This means that if can not be ascribed to any one level of the hierarchy.

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

The red dots representing the escape level SSM are best thought of as multiple copies of

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

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

Details of each level follow in the next section.

5.5.2 OMNI-Level

5.5.3 Escape

5.5.4 Top Level

5.5.5 Horizontal Slice

5.5.5.1 ssmPlanPB

5.5.5.2 ssmMoveToORP

5.5.5.3 ssmConductORP

5.5.5.4 ssmMoveToPB

5.5.5.5 ssmConductPB

5.5.6 Vertical Slice

5.5.6.1 ssmSecureHalt

5.5.6.2 ssmORPRecon

5.5.6.3 ssmMoveToORP4L

5.5.6.4 ssmFormRT

Chapter 6

Secure State Machine Model

6.1 State Machines

6.1.1 Transition Commands

6.1.2 Next-state Function

6.1.3 Next-output Function

6.1.4 Configuration

6.2 Secure State Machines

6.2.1 State Machine Versus Secure State Machine

6.2.2 Monitors

6.2.3 Transition Types

Chapter 7

Patrol Base Operations as Secure State Machines

7.1 ssmPB: A Typical Example from the Hierarchy

7.1.1 Principals

7.1.2 States

7.1.3 Commands

7.1.4 Next-State Function

7.1.5 Next-Output Function

7.1.6 Authentication

7.1.7 Authorization

Chapter 8

Other Theories

This section discusses other theories that are not verified in HOL. These theories highlight the potential for a greater variety of representations of the patrol base operations.

8.1 Authentication & Roles

This master thesis discussed authentication as a visual confirmation among the soldiers. For example, most often, most soldiers in a platoon know who their platoon leader is and do not require a password. But, this is not always the case. One example where it may not be readily obvious is when the platoon leader has been incapacitated and can no longer function in that role. The military has a clearly defined succession hierarchy and soldier are (or should be) aware of this. But, computers aren't as smart as soldiers...unless they are designed to be so. In this case, two concepts come to mind, that of more stringent authentication and of soldiers acting in roles is useful.

More stringent authentication means using means using things such as identification

cards, password, or any variety of authentication techniques. These are all readily modeled in the ACL. A secure state machines that accounts for more stringent authentication is already implemented in HOL.

The idea behind defining the platoon leader as the authorized principal in, for example, the top level SSM allows for the system to work regardless of who the platoon leader is. Nevertheless, it is the soldier acting in the role of the platoon leader who is actually authorized. This can also be represented in the ACL and SSM model. A parametrizable SSM is also already implemented in HOL to do this.

8.2 Soldier, Squad, and Platoon Theories

Chapter 9

Discussion

9.1 Summary

9.2 Lessons Learned

This master thesis demonstrates that CSBD is effective for verifying complete mediation on non-automated, human-centered systems. The CSBD approach is effective, but it is not unique. Therefore, the implication of this work is that complete mediation can be verified on non-automated, human-centered systems.

9.3 Difficulties And Limitations of The Approach

Chapter 10

Future Work

10.1 Applicability

Emergency Management systems require human communications. As with most systems of communication, the communicators need to work with the confidence that they are receiving instructions from the right source with the right authority. A remodeling of the system with complete mediation in mind. From the work in this thesis, the system could simply be modeled in a way that highlights communication avenues and applies CSBD to verify complete mediation.

On perhaps a smaller scale, but no less significant, are non-automated, human-centered school drills such as fire drills and active shooter drills. Clearly, a systems engineering approach could be used to model these drills. The systems security engineering aspect would cover things such as verifying that safety zones are assigned and reasonably accessible, "what to do if" scenarios problems are discussed and solved, and who to trust under what circumstances are solved. Complete mediation would be intertwined within this system design. For example, when should students leave their classrooms after lock-down? Who says so? How do you know if that person is trust worthy? If its

the teacher or a police officer, then authentication is typically a matter of visual recognition as it is with authentication in the patrol base operations. Regardless of how these are decided, these things should be decided during the design phase of the drill and the concepts of complete mediation should be built-in to the design.

10.1.1 Accountability Systems

Appendices

Appendix A

Access Control Logic Theories: Pretty-Printed Theories

Contents

1 aclfoundation Theory

Built: 25 February 2018
Parent Theories: indexedLists, patternMatches

1.1 Datatypes

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

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

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

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

SecLevel = `sLab 'sl | sl 'apn`

1.2 Definitions

[imapKS_def]

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

[intpKS_def]

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

[jKS_def]

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

[O1_def]

$\vdash$ O1 = PO one_weakorder

[one_weakorder_def]

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

[po_TY_DEF]

$\vdash \exists rep.$ TYPE_DEFINITION WeakOrder rep

[po_tybij]

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

[prod_PO_def]

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

[smapKS_def]

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

[Subset_PO_def]

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

1.3 Theorems

[abs_po11]

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

[absPO_fn_onto]

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

[antisym_prod_antisym]

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

[EQ_WeakOrder]

$\vdash$ WeakOrder (=)

[KS_bij]

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

[one_weakorder_WO]

$\vdash$ WeakOrder one_weakorder

[onto_po]

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

[po_bij]

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

[PO_repPO]

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

[refl_prod_refl]

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

[repPO_iPO_partial_order]

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

[repPO_O1]

$\vdash$ repPO O1 = one_weakorder

[repPO_prod_PO]

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

[repPO_Subset_PO]

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

[RPROD_THM]

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

[SUBSET_WO]

$\vdash$ WeakOrder ($\subseteq$)

[trans_prod_trans]

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

[WeakOrder_Exists]

$\vdash \exists R.$ WeakOrder R

[WO_prod_WO]

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

[WO_repPO]

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

2 aclsemantics Theory

Built: 25 February 2018
Parent Theories: aclfoundation

2.1 Definitions

[Efn_def]

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

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

[Jext_def]

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

[Lifn_def]

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

[Lsfn_def]

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

2.2 Theorems

[andf_def]

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

[controls_def]

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

[controls_says]

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

[domi_def]

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

[doms_def]

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

[eqf_def]

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

[eqf_impf]

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

[eqi_def]

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

[eqi_domi]

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

[eqn_def]

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

[eqs_def]

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

[eqs_doms]

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

[FF_def]

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

[impf_def]

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

[lt_def]

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

[lte_def]

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

[meet_def]

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

[name_def]

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

[notf_def]

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

[orf_def]

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

[prop_def]

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

[quoting_def]

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

[reps_def]

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

[says_def]

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

[speaks_for_def]

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

[TT_def]

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

3 aclrules Theory

Built: 25 February 2018
Parent Theories: aclsemantics

3.1 Definitions

[sat_def]

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

3.2 Theorems

[And_Says]

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

[And_Says_Eq]

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

[and_says_lemma]

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

[Controls_Eq]

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

[DIFF_UNIV_SUBSET]

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

[domi_antisymmetric]

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

[domi_reflexive]

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

[domi_transitive]

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

[doms_antisymmetric]

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

[doms_reflexive]

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

[doms_transitive]

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

[eqf_and_impf]

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

[eqf_andf1]

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

[eqf_andf2]

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

[eqf_controls]

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

[eqf_eq]

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

[eqf_eqf1]

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

[eqf_eqf2]

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

[eqf_impf1]

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

[eqf_impf2]

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

[eqf_notf]

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

[eqf_orf1]

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

[eqf_orf2]

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

[eqf_reps]

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

[eqf_sat]

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

[eqf_says]

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

[eqi_Eq]

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

[eqs_Eq]

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

[Idemp_Speaks_For]

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

[Image_cmp]

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

[Image_SUBSET]

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

[Image_UNION]

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

[INTER_EQ_UNIV]

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

[Modus_Ponens]

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

[Mono_speaks_for]

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

[MP_Says]

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

[Quoting]

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

[Quoting_Eq]

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

[reps_def_lemma]

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

[Reps_Eq]

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

[sat_allworld]

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

[sat_andf_eq_and_sat]

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

[sat_TT]

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

[Says]

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

[says_and_lemma]

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

[Speaks_For]

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

[speaks_for_SUBSET]

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

[SUBSET_Image_SUBSET]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4 aclDrules Theory

Built: 25 February 2018
Parent Theories: aclrules

4.1 Theorems

[Conjunction]

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

[Controls]

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

[Derived_Controls]

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

[Derived_Speaks_For]

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

[Disjunction1]

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

[Disjunction2]

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

[Disjunctive_Syllogism]

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

[Double_Negation]

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

[eqn_eqn]

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

[eqn_lt]

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

[eqn_lte]

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

[Hypothetical_Syllogism]

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

[il_domi]

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

[INTER_EQ_UNIV]

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

[Modus_Tollens]

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

[Rep_Controls_Eq]

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

[Rep_Says]

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

[Reps]

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

[Says_Simplification1]

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

[Says_Simplification2]

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

[Simplification1]

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

[Simplification2]

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

[sl_doms]

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

Index

Appendix B

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

Contents

1 OMNIType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

1.1 Datatypes

command = ESCc escCommand | SLc 'slCommand

escCommand = returnToBase | changeMission | resupply
| reactToContact

escOutput = ReturnToBase | ChangeMission | Resupply
| ReactToContact

escState = RTB | CM | RESUPPLY | RTC

output = ESCo escOutput | SLo 'slOutput

principal = SR 'stateRole

state = ESCs escState | SLs 'slState

1.2 Theorems

[command_distinct_clauses]

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

[command_one_one]

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

[escCommand_distinct_clauses]

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

[escOutput_distinct_clauses]

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

[escState_distinct_clauses]

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

[output_distinct_clauses]

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

[output_one_one]

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

[principal_one_one]

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

[state_distinct_clauses]

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

[state_one_one]

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

2 ssm11 Theory

Built: 13 May 2018
Parent Theories: satList

2.1 Datatypes

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

order = SOME 'command | NONE

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

2.2 Definitions

[TR_def]

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

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

2.3 Theorems

[CFGInterpret_def]

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

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

[CFGInterpret_ind]

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

[configuration_one_one]

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

[order_distinct_clauses]

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

[order_one_one]

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

[TR_cases]

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

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

[TR_discard_cmd_rule]

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

[TR_EQ_rules_thm]

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

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

[TR_exec_cmd_rule]

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

[TR_ind]

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

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

[TR_rules]

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

[TR_strongind]

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

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

[TR_trap_cmd_rule]

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

[TRrule0]

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

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

[TRrule1]

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

[trType_distinct_clauses]

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

[trType_one_one]

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

3 ssm Theory

Built: 13 May 2018
Parent Theories: satList

3.1 Datatypes

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

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

3.2 Definitions

[authenticationTest_def]

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

[commandList_def]

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

[inputList_def]

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

[propCommandList_def]

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

[TR_def]

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

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

3.3 Theorems

[CFGInterpret_def]

$$
\begin{array}{l}
\vdash \texttt{CFGInterpret}\ (M,Oi,Os) \\
\quad (\texttt{CFG}\ elementTest\ stateInterp\ context\ (x::ins)\ state \\
\quad\quad outStream) \iff \\
\quad (M,Oi,Os)\ \texttt{satList}\ context\ x \land (M,Oi,Os)\ \texttt{satList}\ x \land \\
\quad (M,Oi,Os)\ \texttt{satList}\ stateInterp\ state\ x
\end{array}
$$

[CFGInterpret_ind]

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

[configuration_one_one]

$$
\begin{array}{l}
\vdash \forall a_0\ a_1\ a_2\ a_3\ a_4\ a_5\ a_0'\ a_1'\ a_2'\ a_3'\ a_4'\ a_5'. \\
\quad (\texttt{CFG}\ a_0\ a_1\ a_2\ a_3\ a_4\ a_5 = \texttt{CFG}\ a_0'\ a_1'\ a_2'\ a_3'\ a_4'\ a_5') \iff \\
\quad (a_0 = a_0') \land (a_1 = a_1') \land (a_2 = a_2') \land (a_3 = a_3') \land \\
\quad (a_4 = a_4') \land (a_5 = a_5')
\end{array}
$$

[extractCommand_def]

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

[extractCommand_ind]

$$
\begin{array}{l}
\vdash \forall P'. \\
\quad (\forall P\ cmd.\ P'\ (P\ \texttt{says}\ \texttt{prop}\ (\texttt{SOME}\ cmd))) \land P'\ \texttt{TT} \land P'\ \texttt{FF} \land \\
\quad (\forall v_1.\ P'\ (\texttt{prop}\ v_1)) \land (\forall v_3.\ P'\ (\texttt{notf}\ v_3)) \land
\end{array}
$$

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

[extractInput_def]

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

[extractInput_ind]

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

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

[extractPropCommand_def]

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

[extractPropCommand_ind]

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

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

[TR_cases]

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

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

[TR_discard_cmd_rule]

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

[TR_EQ_rules_thm]

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

[TR_exec_cmd_rule]

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

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

[TR_ind]

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

[TR_rules]

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

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

[TR_strongind]

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

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

[TR_trap_cmd_rule]

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

[TRrule0]

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

[TRrule1]

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

[trType_distinct_clauses]

⊢ ($\forall a'\ a$. discard $a \neq$ trap a') ∧ ($\forall a'\ a$. discard $a \neq$ exec a') ∧
$\forall a'\ a$. trap $a \neq$ exec a'

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

4 satList Theory

Built: 13 May 2018
Parent Theories: aclDrules

4.1 Definitions

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

4.2 Theorems

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

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

[satList_nil]
⊢ (M,Oi,Os) satList []

5 ssmPB Theory

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

5.1 Definitions

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

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

5.2 Theorems

[authenticationTest_cmd_reject_lemma]

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

[authenticationTest_def]

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

[authenticationTest_ind]

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

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

[PBNS_def]

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

[PBNS_ind]

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

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

[PBOut_def]

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

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

[PBOut_ind]

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

[PlatoonLeader_exec_slCommand_justified_thm]

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

[PlatoonLeader_slCommand_lemma]

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

6 PBTypeIntegrated Theory

Built: 13 May 2018
Parent Theories: OMNIType

6.1 Datatypes

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

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

slCommand = PL PBTypeIntegrated$plCommand | OMNI omniCommand

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

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

stateRole = PlatoonLeader | Omni

6.2 Theorems

[omniCommand_distinct_clauses]

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

[plCommand_distinct_clauses]

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

[slCommand_distinct_clauses]

⊢ ∀ a' a. PL a ≠ OMNI a'

[slCommand_one_one]

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

[slOutput_distinct_clauses]

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

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

[slState_distinct_clauses]

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

[stateRole_distinct_clauses]

```
⊢ PlatoonLeader ≠ Omni
```

7 PBIntegratedDef Theory

Built: 13 May 2018

Parent Theories: PBTypeIntegrated, aclfoundation

7.1 Definitions

[secAuthorization_def]

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

[secHelper_def]

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

7.2 Theorems

[getOmniCommand_def]

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

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

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

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

[getOmniCommand_ind]

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

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

[secContext_def]

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

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

[secContext_ind]

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

8 ssmConductORP Theory

Built: 13 May 2018

Parent Theories: ConductORPType, ssm11, OMNIType

8.1 Definitions

[secContextConductORP_def]

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

[ssmConductORPStateInterp_def]

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

8.2 Theorems

[authTestConductORP_cmd_reject_lemma]

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

[authTestConductORP_def]

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

[authTestConductORP_ind]

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

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

[conductORPNS_def]

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

[conductORPNS_ind]

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

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

[conductORPOut_def]

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

[conductORPOut_ind]

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

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

[PlatoonLeader_exec_plCommand_justified_thm]

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

[PlatoonLeader_plCommand_lemma]

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

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

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

9 ConductORPType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

9.1 Datatypes

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

psgCommand = actionsIn | psgIncomplete

slCommand =
    PL ConductORPType$plCommand
  | PSG ConductORPType$psgCommand

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

slState = CONDUCT_ORP | SECURE | ACTIONS_IN | WITHDRAW
        | COMPLETE

stateRole = PlatoonLeader | PlatoonSergeant
```

9.2 Theorems

[plCommand_distinct_clauses]

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

[psgCommand_distinct_clauses]

$\vdash$ actionsIn $\neq$ psgIncomplete

[slCommand_distinct_clauses]

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

[slCommand_one_one]

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

[slOutput_distinct_clauses]

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

[slRole_distinct_clauses]

$\vdash$ PlatoonLeader $\neq$ PlatoonSergeant

[slState_distinct_clauses]

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

10 ssmConductPB Theory

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

10.1 Definitions

[secContextConductPB_def]

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

[ssmConductPBStateInterp_def]

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

10.2 Theorems

[authTestConductPB_cmd_reject_lemma]

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

[authTestConductPB_def]

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

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

[authTestConductPB_ind]

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

$\forall v.\ P\ v$

[conductPBNS_def]

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

[conductPBNS_ind]

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

[conductPBOut_def]

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

[conductPBOut_ind]

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

[PlatoonLeader_exec_plCommandPB_justified_thm]

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

[PlatoonLeader_plCommandPB_lemma]

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

[PlatoonSergeant_exec_psgCommandPB_justified_thm]

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

[PlatoonSergeant_psgCommandPB_lemma]

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

11 ConductPBType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

11.1 Datatypes

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

psgCommandPB = actionsInPB | psgIncompletePB

slCommand = PL plCommandPB | PSG psgCommandPB

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

slState = CONDUCT_PB | SECURE_PB | ACTIONS_IN_PB | WITHDRAW_PB
        | COMPLETE_PB

stateRole = PlatoonLeader | PlatoonSergeant
```

11.2 Theorems

[plCommandPB_distinct_clauses]

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

[psgCommandPB_distinct_clauses]

```
⊢ actionsInPB ≠ psgIncompletePB
```

[slCommand_distinct_clauses]

```
⊢ ∀a' a. PL a ≠ PSG a'
```

[slCommand_one_one]

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

[slOutput_distinct_clauses]

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

[slRole_distinct_clauses]

⊢ PlatoonLeader ≠ PlatoonSergeant

[slState_distinct_clauses]

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

12 ssmMoveToORP Theory

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

12.1 Definitions

[secContextMoveToORP_def]

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

[ssmMoveToORPStateInterp_def]

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

12.2 Theorems

[authTestMoveToORP_cmd_reject_lemma]

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

[authTestMoveToORP_def]

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

[authTestMoveToORP_ind]

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

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

[moveToORPNS_def]

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

[moveToORPNS_ind]

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

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

[moveToORPOut_def]

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

[moveToORPOut_ind]

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

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

[PlatoonLeader_exec_slCommand_justified_thm]

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

[PlatoonLeader_slCommand_lemma]

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

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

13 MoveToORPType Theory

Built: 13 May 2018

Parent Theories: indexedLists, patternMatches

13.1 Datatypes

slCommand = pltForm | pltMove | pltSecureHalt | complete
| incomplete

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

slState = MOVE_TO_ORP | PLT_FORM | PLT_MOVE | PLT_SECURE_HALT
| COMPLETE

stateRole = PlatoonLeader

13.2 Theorems

[slCommand_distinct_clauses]

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

[slOutput_distinct_clauses]

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

[slState_distinct_clauses]

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

14 ssmMoveToPB Theory

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

14.1 Definitions

[secContextMoveToPB_def]

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

[ssmMoveToPBStateInterp_def]

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

14.2 Theorems

[authTestMoveToPB_cmd_reject_lemma]

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

[authTestMoveToPB_def]

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

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

[authTestMoveToPB_ind]

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

[moveToPBNS_def]

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

[moveToPBNS_ind]

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

[moveToPBOut_def]

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

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

[moveToPBOut_ind]

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

[PlatoonLeader_exec_slCommand_justified_thm]

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

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

[PlatoonLeader_slCommand_lemma]

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

15 MoveToPBType Theory

Built: 13 May 2018
Parent Theories: indexedLists, patternMatches

15.1 Datatypes

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

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

slState = MOVE_TO_PB | PLT_FORM | PLT_MOVE | PLT_HALT | COMPLETE

stateRole = PlatoonLeader
```

15.2 Theorems

[slCommand_distinct_clauses]

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

[slOutput_distinct_clauses]

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

[slState_distinct_clauses]

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

16 ssmPlanPB Theory

Built: 13 May 2018
Parent Theories: PlanPBDef, ssm

16.1 Theorems

[inputOK_def]

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

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

[inputOK_ind]

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

[planPBNS_def]

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

[planPBNS_ind]

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

[planPBOut_def]

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

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

[planPBOut_ind]

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

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_lemma]

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

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

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_justified_thm]

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

[PlatoonLeader_notWARNO_notreport1_exec_plCommand_lemma]

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

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

[PlatoonLeader_psgCommand_notDiscard_thm]

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

[PlatoonLeader_trap_psgCommand_justified_lemma]

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

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

[PlatoonLeader_trap_psgCommand_lemma]

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

[PlatoonLeader_WARNO_exec_report1_justified_lemma]

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

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

[PlatoonLeader_WARNO_exec_report1_justified_thm]

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

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

[PlatoonLeader_WARNO_exec_report1_lemma]

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

[PlatoonSergeant_trap_plCommand_justified_lemma]

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

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

[PlatoonSergeant_trap_plCommand_justified_thm]

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

[PlatoonSergeant_trap_plCommand_lemma]

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

17 PlanPBType Theory

Built: 13 May 2018

Parent Theories: indexedLists, patternMatches

17.1 Datatypes

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

psgCommand = initiateMovement | psgIncomplete
           | invalidPsgCommand

slCommand = PL plCommand | PSG psgCommand

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

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

stateRole = PlatoonLeader | PlatoonSergeant

17.2 Theorems

[plCommand_distinct_clauses]

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

[psgCommand_distinct_clauses]

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

[slCommand_distinct_clauses]

⊢ ∀ a' a. PL a ≠ PSG a'

[slCommand_one_one]

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

[slOutput_distinct_clauses]

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

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

[slRole_distinct_clauses]

⊢ PlatoonLeader ≠ PlatoonSergeant

[slState_distinct_clauses]

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

Index

Appendix C

Secure State Machine Theories: HOL Script Files

C.1 ssm

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


structure ssmScript = struct

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

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

open HolKernel boolLib Parse bossLib
open TypeBase listTheory optionTheory ssminfRules acl_infRules satListTheory
(************************)
(* create a new theory *)
(************************)
val _ = new_theory "ssm";

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

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

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

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

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

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

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

val _ = save_thm("configuration_one_one",configuration_one_one)

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

(*****************************************************************************)
(* In the following definitions of authenticationTest, extractCommand, and   *)
(* commandList, we implicitly assume that the only authenticated inputs are  *)
(* of the form P says phi, i.e., we know who is making statement phi.        *)
(*****************************************************************************)

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

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

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

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

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

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

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

(* ------------------------------------------------------------------------- *)
(* Define transition relation among configurations. This definition is       *)
(* parameterized in terms of next-state transition function and output       *)
(* function.                                                                 *)
(* ------------------------------------------------------------------------- *)
val (TR_rules, TR_ind, TR_cases) =
Hol_reln
`(!(elementTest:('command option,'principal,'d,'e)Form -> bool)
    (NS: 'state -> ('command option list) trType -> 'state) M Oi Os Out (s:'state)
    (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
    (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
     ('command option,'principal,'d,'e)Form list)
    (x:('command option,'principal,'d,'e)Form list)
    (ins:('command option,'principal,'d,'e)Form list list)
    (outs:'output list).
 (authenticationTest elementTest x) /\
 (CFGInterpret (M,Oi,Os)
  (CFG elementTest stateInterp context (x::ins) s outs)) ==>
 (TR
  ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
  (exec (inputList x))
  (CFG elementTest stateInterp context (x::ins) s outs)
  (CFG elementTest stateInterp context ins
       (NS s (exec (inputList x)))
       ((Out s (exec (inputList x)))::outs)))) /\
 (!(elementTest:('command option,'principal,'d,'e)Form -> bool)
    (NS: 'state -> ('command option list) trType -> 'state) M Oi Os Out (s:'state)
    (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
    (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
     ('command option,'principal,'d,'e)Form list)
    (x:('command option,'principal,'d,'e)Form list)
    (ins:('command option,'principal,'d,'e)Form list list)
    (outs:'output list).
 (authenticationTest elementTest x)  /\
 (CFGInterpret (M,Oi,Os)
  (CFG elementTest stateInterp context (x::ins) s outs)) ==>
 (TR
  ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
  (trap (inputList x))
 (CFG elementTest stateInterp context (x::ins) s outs)
 (CFG elementTest stateInterp context ins
```

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


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

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


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

val TR_lemma2 =
TAC_PROOF(([],flip_TR_rules rule2),
DISCH_TAC THEN
IMP_RES_TAC TR_cases THEN
PAT_ASSUM
 ``discard (inputList x)= y``
 (fn th => ASSUME_TAC(REWRITE_RULE[trType_one_one,trType_distinct_clauses]th)) THEN
PROVE_TAC[configuration_one_one,list_11,trType_distinct_clauses])

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

val TR_EQ_rules_thm = TR_EQ_rules TR_rules TR_rules_converse

val _ = save_thm("TR_EQ_rules_thm",TR_EQ_rules_thm)

val [TRrule0,TRrule1,TR_discard_cmd_rule] = CONJUNCTS TR_EQ_rules_thm

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

(* ------------------------------------------------------------------------- *)
(* If (CFGInterpret                                                          *)
```

```
(*      (M,Oi,Os)                                                          *)
(*      (CFG elementTest stateInterpret certList                           *)
(*           ((P says (prop (CMD cmd)))::ins) s outs) ==>                  *)
(*    ((M,Oi,Os) sat (prop (CMD cmd))))                                   *)
(* is a valid inference rule, then executing cmd the exec(CMD cmd) transition *)
(* occurs if and only if prop (CMD cmd), elementTest, and                 *)
(* CFGInterpret (M,Oi,Os)                                                 *)
(*  (CFG elementTest stateInterpret certList (P says prop (CMD cmd)::ins) s outs)  *)
(* are true.                                                              *)
(* ---------------------------------------------------------------------- *)
val TR_exec_cmd_rule =
TAC_PROOF(([],
``!elementTest context stateInterp (x:('command option,'principal,'d,'e)Form list)
   ins s outs.
 (!M Oi Os.
 (CFGInterpret
  ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po), (Os :'e po))
  (CFG elementTest
       (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
        ('command option,'principal,'d,'e)Form list) context
       (x::ins)
       (s:'state) (outs:'output list))) ==>
  (M,Oi,Os) satList (propCommandList (x:('command option, 'principal, 'd, 'e)Form list))) ==>
(!NS Out M Oi Os.
 TR
  ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po),
   (Os :'e po)) (exec (inputList x))
  (CFG (elementTest :('command option, 'principal, 'd, 'e) Form -> bool)
           (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
            ('command option,'principal,'d,'e)Form list)
           (context :('command option, 'principal, 'd, 'e) Form list ->
            ('command option, 'principal, 'd, 'e) Form list)
           (x::ins)
           (s :'state) (outs :'output list))
   (CFG elementTest stateInterp context ins
           ((NS :'state -> 'command option list trType -> 'state) s (exec (inputList x)))
           (Out s (exec (inputList x))::outs)) <=>
  (authenticationTest elementTest x) /\
  (CFGInterpret (M,Oi,Os)
   (CFG elementTest stateInterp context (x::ins) s outs)) /\
   (M,Oi,Os) satList (propCommandList x))``),
REWRITE_TAC[TRrule0] THEN
REPEAT STRIP_TAC THEN
EQ_TAC THEN
REPEAT STRIP_TAC THEN
PROVE_TAC[])

val _ = save_thm("TR_exec_cmd_rule",TR_exec_cmd_rule)

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

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

val _ = save_thm("TR_trap_cmd_rule",TR_trap_cmd_rule)

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

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

end (* structure *)
```

C.2 satList

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

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

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

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

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

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

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

val _ = save_thm("satList_nil",satList_nil)

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

val _ = save_thm("satList_conj",satList_conj)

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

val _ = save_thm("satList_CONS",satList_CONS)

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

end (* structure *)
```

Appendix D

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

D.1 OMNILevel

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

structure OMNIScript = struct

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

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

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

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

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

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

val _=
Datatype `slCommand = OMNI omniCommand`


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

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



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

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



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

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

val _= export_theory();
end
```

D.2 escapeLevel

D.3 TopLevel

D.3.1 PBTypeIntegrated Theory: Type Definitions

```
(*************************************************************************)
(* PBTypeIntegrated                                                      *)
(* Author: Lori Pickering                                                *)
(* Date 12 May 2018                                                      *)
```

```
(* This theory contains the type definitions for ssmPBIntegrated              *)
(*****************************************************************************)
structure PBTypeIntegratedScript = struct


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


open HolKernel Parse boolLib bossLib;
open TypeBase OMNITypeTheory

val _= new_theory "PBTypeIntegrated";

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

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

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

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


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


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


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



val _=
Datatype `stateRole = PlatoonLeader | Omni`

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


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

```
                    | COMPLETE_PB`

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

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

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


val _= export_theory();
end
```

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

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

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

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

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


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

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

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

val _= export_theory();
end
```

D.3.3 ssmPlanPBIntegrated Theory: Theorems

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


structure ssmPBIntegratedScript = struct

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

open TypeBase listTheory optionTheory listSyntax
```

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

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


val _ = new_theory "ssmPBIntegrated";


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

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

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


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

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



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

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

val _ = export_theory();

end
```

D.4 Horizontal Slice

D.4.1 ssmPlanPB

D.4.1.1 PlanPBType Theory: Type Definitions

D.4.1.2 PlanPBDef Theory: Authentication & Authorization Definitions

D.4.1.3 ssmPlanPB Theory: Theorems

D.4.2 ssmMoveToORP

D.4.2.1 MoveToORPType Theory: Type Definitions

D.4.2.2 MoveToORPDef Theory: Authentication & Authorization Definitions

D.4.2.3 ssmMoveToORP Theory: Theorems

D.4.3 ssmConductORP

D.4.3.1 ConductORPType Theory: Type Definitions

D.4.3.2 ConductORPDef Theory: Authentication & Authorization Definitions

D.4.3.3 ssmConductORP Theory: Theorems

D.4.4 ssmMoveToPB

D.4.4.1 MoveToPBType Theory: Type Definitions

D.4.4.2 MoveToPBDef Theory: Authentication & Authorization Definitions

D.4.4.3 ssmMoveToPB Theory: Theorems

D.4.5 ssmConductPB

Appendix E

Map of The File Folder Structure

References

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

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

[3] Shiu-Kai Chin and Susan Older. *Certified Security by Design Using Higher Order Logic*, volume Version 1.5. Department of Electrical Engineering and Computer Science Syracuse University, Syracuse University, Syracuse, New York 13244, November 2017.

[4] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems, April 1975.

[5] Availability, January 2018.

[6] Michael Collins. Formal methods: 18-849b dependable embedded systems. Technical report, Carnegie Mellon University, Spring 1998.

[7] Edmund M. Clarke and et al Jeannette M. Wing. Formal methods: State of the art and future directions, December 1996.

[8] A. Prasad Sistla, V.N.Venkatakrishnan, Michelle Zhou, and Hilary Branske. Cmv: Automatic verification of complete mediation for java virtual machines. In *ASIACCS '08 Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 100–111, March 2008.

[9] TOKUOKA Hiroki, MIYAZAKI Yoshiaki, and HASHIMOTO Yuusuke. C-language verifiction tool using formal methods "varvel".

[10] ZHU Xin-feng, WANG Jian-dong, Li Bin, ZHU Jun-wu, and Wu Jun. Methods to tackle state explosion problem in model checking. In *2009 Third International Symposium on Intelligent Information Technology Application*, number ISBN: New-2005_POD_978-0-7695-3859-4, pages 329–331. IEEE, December 2009.

[11] Formal methods. *Wikipedia*, February 2018.

[12] David Turner and Yannick Welsch. Reliable by design: Applying formal methods to destributed systems.

[13] Jagatheesan Kunasaikaran, Azlan Iqbal, Jalan Dua, and Chan Sow Lin. A brief overview of functional programming languages, 2016.

[14] *Introduction to Programming Languages/Algebraic Data Types*. used under the Creative Commons Attribution-ShareAlike 3.0 License, August 2017.

[15] Bsd licenses, May 2018.

[16] Hol interactive theorem prover.

[17] Hol (proof assistant), April 2016.

[18] ISO/IEC/IEEE 15288 international standard - systems and software engineering – system life cycle processes. Technical Report ISO/IEC/IEEE 15288:2015(E), International Organization for Standardization (ISO) & International Electrotechnical Commission (IEC) & Institute of Electrical and Electronics Engineers (IEEE), 3 Park Avenue, New York, NY 10016-5997, USA, May 2015.

[19] ISO. Systems and software engineering–life cycle management–part 1: guide for life cycle management. Technical Report ISO/IEC TR 24748-1:2010(E), International Organization for Standardization (ISO), September 2010.

[20] Concepts of operations, May 2018.

[21] Lt Gen Charles F. Wald or Lt Col Sanders. Air force policy directive 10-28: Conops development. Operations 10-28, Secrtary of The Air Force, January 2002.

[22] NIST. Nist mission, vision, core competencies, and core values.

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

[24] Saul kripke, May 2018.

[25] Charles Q. Choi. Alien planets with extra suns can have strange orbits. *Space.com*.

[26] Algebraic data types, May 2018.