The Common Criteria, Formal Methods and ACL2

Raymond Richards, David Greve, Matthew Wilding
Rockwell Collins Advanced Technology Center
Cedar Rapids, Iowa 52498 USA
{rjricha1,dagreve,mmwildin}@rockwellcollins.com

and

W. Mark Vanfleet
U.S. Department of Defense

Abstract

The "Common Criteria" is an international standard for evaluating secure computer systems. The Common Criteria defines seven distinct Evaluation Assurance Levels (EALs). The three highest EALs, the so-called high-assurance levels, require some degree of formalism in development. This paper discusses requirements for formalism in the high-assurance levels and our initial experiences in satisfying these requirements using ACL2.

1. Introduction

The *Common Criteria for Information Technology Security Evaluation* [1], often referred to as simply the Common Criteria, is an international standard for the evaluation of security related computing technology. The intention is to provide a *lingua franca* for consumers, developers and evaluators of such technology, which allows a standard way of comparing evaluations. The Common Criteria defines seven Evaluation Assurance Levels (EALs). The EALs range from EAL 1 (functionally tested) to EAL 7 (formally verified design and tested). To allow flexibility, the Common Criteria can be tailored for a class of applications (e.g. Real-Time Operating Systems) with a *Protection Profile* document and/or for a single application (e.g. Windows NT) with a *Security Target* document.

The Common Criteria defines requirements for certification that are categorized into classifications. Some examples of classifications are Configuration Management,

Assurance Maintenance, and Development. The classifications are identified by a three letter identifier such as ADV for Development. Each requirement may have multiple definitions, generally increasing in stringency. The requirements are also identified by a three letter identifier and appended to its classification with an underscore. (e.g. ADV_HLD for the High Level Design requirement in the Development classification.) The level of each requirement is identified with a numeral, starting at 1 and increasing with the stringency. (e.g. ADV_HLD.2.) An EAL definition specifies the set of requirements and which definition is used for each requirement.

Requirement Name	Identifier
Functional Specification	ADV_FSP
High-Level Design	ADV_HLD
Implementation	ADV_IMP
Internals	ADV_INT
Low-Level Design	ADV_LLD
Representation Correspondence	ADV_RCR
Security Policy Modeling	ADV_SPM

Table 1: Development Classification Requirements

The EALs are generally used as guidelines. Any system that is to be certified must conform to either a *Protection Profile* or a *Security Target.* These documents define the set of requirements and level of definition for each requirement. These documents are not required to follow the Common Criteria specification for an EAL level, often they call out stricter definitions for one or more requirements.

The higher EAL levels (5, 6 and 7) are sometimes referred to as the *high-assurance levels*. These levels require some application of formal methods to demonstrate that the appropriate level of assurance has been met. The requirements of interest are in the *Development* classification and allow the formalism to range from informal to semi-formal to formal, depending on the EAL level. We discuss in this paper these formalism requirements and how those requirements have been met using ACL2 in Rockwell Collins projects.

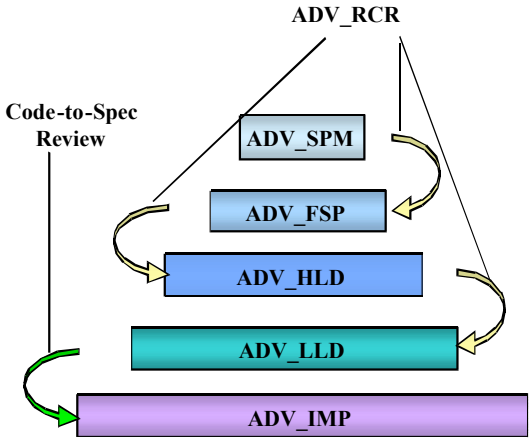

Figure 1: Design Assurance Architecture

The Development classification of requirements contains seven requirements, which are shown in Table 1. Of these seven requirements, two have no direct formalism requirement. “Internals” is concerned with design modularity and reduction of complexity and “Implementation” is concerned with the implementation artifacts (source code).

Figure 1 shows the Common Criteria design assurance architecture. At the top level, the Security Policy defines the characteristics of the system that must be demonstrated. The functional specification is a high-level description of the user-visible interface to the system. The high-level design decomposes the system into modules, or subsystems, which provide the functionality described in the functional specification. The low-level design provides a specification of the internal workings of each module. The representation correspondence demonstrates the correlation between each adjacent level in the architecture, with the exception of the correspondence between the low-level design and the implementation. This correspondence is provided by an evaluation activity known as a '*code-to-spec review*'. A code-to-spec review is an intensive walk-through in the presence of certification authority evaluators, comparing the low-level design model to the implementation so as to demonstrate their correspondence.

The Common Criteria requirements that dictate the use of formality are shown in Table 2 for the high-assurance EALs. It should be noted that the representation correspondence requirement between a pair of representations only holds if both layers are of at least that level of formalism, otherwise the rigor of correspondence needs to be only the lesser level of the two layers. For example, in EAL7, the correspondence between the high-level design and the low-level design need only be semiformal, since the low-level design is only represented semiformally.

Requirement	**EAL5**	**EAL6**	**EAL7**
ADV_SPM	Formal TOE Security Policy Model	Formal TOE Security Policy Model	Formal TOE Security Policy Model
ADV_FSP	Semiformal Functional Specification	Semiformal Functional Specification	Formal Functional Specification
ADV_HLD	Semiformal High-Level Design	Semiformal High-Level Explanation	Formal High-Level Design
ADV_LLD	Descriptive Low-Level Design	Semiformal Low-Level Design	Semiformal Low-Level Design
ADV_RCR	Semiformal Correspondence Demonstration	Semiformal Correspondence Demonstration	Formal Correspondence Demonstration

Table 2: High assurance requirements

The Common Criteria specifies three levels of rigor in the Development class of requirements. These three levels are informal, semiformal and formal. Informal representations may be represented in the prose of a natural language. Formal representations must be represented in a notation based upon well-established mathematical concepts. The formal level also requires that there is evidence that it is impossible to derive contradictions and all rules supporting the notation need to be defined or referenced.

The semiformal level of rigor requires representation in a 'restricted syntax' language. This can include natural language, with restrictions placed on sentence structure and keywords. This can also include graphical languages. The notion of a semiformal representation can take a great number of forms. However, since the final goal is certification of the system, it is important that a semiformal representation be rigorous enough to satisfy the certification authority evaluators. Also, consideration must be given to how correspondence between semiformal representations will be achieved, and if there will be any future need to fully formalize that representation.

2. Meeting the Requirements

This section will focus on how requirements with semiformal or formal levels of rigor are satisfied in Rockwell Collins projects; we will not discuss the informal level of rigor.

The layered design assurance architecture presented in the previous section provides increasing levels of abstraction as you move upward in the layers. ACL2 has been demonstrated to be a tool adept at layering specifications with increasing levels of abstraction, as shown in the CLI short stack [2]. In the CLI short stack, a series of machines are defined that describe hardware implementation and specification, compiler/assembler implementation and specification, and application implementation and specification. Each layer is proved to implement its specification, assuming the correctness of the layers beneath it. ACL2 is well suited to address the formal and semiformal requirements of the Common Criteria by stacking layers with increasing abstraction, much as was described for the short stack. Later in the document we describe how the Common Criteria requirements form a layered proof architecture, the ability to prove each layer implements its specification is part and parcel of the certification effort. Moreover, ACL2 facilitates the composition of larger systems from certified components by allowing the properties of one component to be used as assumptions of a higher-level component. This ability facilitates the so-called "MILS Architecture" being developed by a consortium of companies. The layers of this architecture (such as an RTOS, middleware and applications) can be developed and certified independently, relying on the soundness of all layers below. For this composition, the formal security policy of one layer becomes an assumption made by all layers above it. Just as the implementations "stack", so too do the correctness statements and proofs.

Either a *Protection Profile* or a *Security Target* document describes the security function of a system. Deriving a formal security policy specification for the security function described by these documents is a challenging task. A description of such a security policy (known as the GWV security policy) is given in [4]. What makes this problem challenging is that it is difficult to know if the formal specification adequately captures the intent of the security function. To provide assurance that the security policy specification is appropriate, it can be helpful to use this specification as a property in a larger system. Proving the correct operation of such a higher-level system increases confidence that the formal security policy has been correctly captured.

The Common Criteria is explicit in stating that each representation does not need to be a separate document. One ACL2 model may satisfy more than one of the required representations, which has the benefit of making correspondence between these layers trivial. We present two representation architectures, one from a past project and one that is proposed for future projects. Both of these representation architectures have a single representation that satisfies more then one requirement.

The 2-model representation architecture shown in Figure 2 has been used in the Rockwell Collins AAMP7 verification project [3]. The security policy is modeled by a theorem that states the separation property that must hold for this system. The ADV_FSP and ADV_HLD requirements are satisfied by a single abstract formal ACL2 model. The ADV_LLD requirement is satisfied by a detailed formal ACL2 model. In this project all models and correspondences are represented formally, which meets EAL 7 requirements plus satisfies ADV_LLD.3.

The representation shown in Figure 3 has been proposed for future Rockwell Collins projects. The Security Policy is modeled by a theorem that states the important separation property to be proved on this system. The requirements of ADV_FSP, ADV_HLD and ADV_LLD are all satisfied by a single ACL2 model. The rigor of this model would depend on the certification requirements to be satisfied.

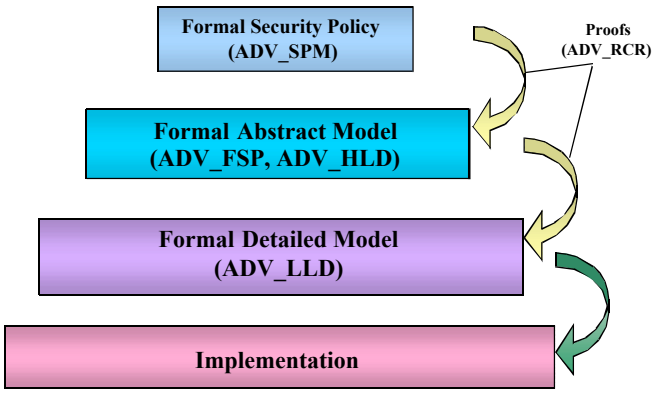

Figure 2: Representation Architecture 1

We proved that the AAMP7's formal security policy held over its low-level model. We claim its proof over the high-level design does not add any certainty to the validation. We will argue that we could have used this 1-model representation architecture for the AAMP7 program. Figure 4 contrasts the two theorems needed to be proved for the 2-model representation architecture with the one theorem to be proved in the 1-model representation architecture.

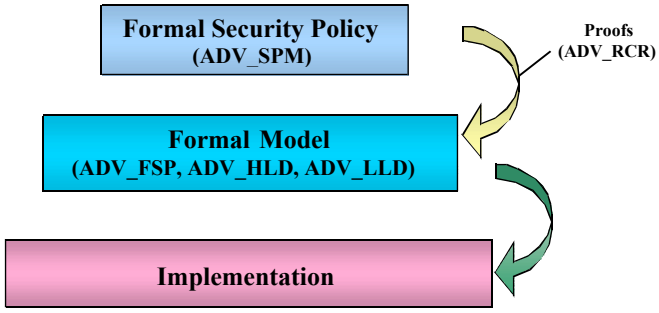

Figure 3: Representation Architecture 2

```
(defthm lift-next-commute
  (implies
    (and
      (hld::secure-configuration spex)
      (spex-hyp :any :trusted :raw spex  lld::st))
   (equal (lift-raw spex   (lld::next spex  lld::st))
          (hld::next spex (lift-raw spex  lld::st)))))


(defthm hld-separation
  (implies
    (let ((dia (raw-dia-from-abstract seg st1)))
      (and
        (equal (raw-selectlist dia st1)
               (raw-selectlist dia st2))
        (equal (raw-select '(:external) st1) (raw  -select '(:external)  st2))))
   (iff (equal (raw-select seg (next spex st1))
               (raw-select seg (next spex st2))) t)))


(defthm lld-separation
  (implies
    (and
      (secure-configuration spex)
      (spex-hyp :any :trusted :raw spex  lld::st1)
      (spex-hyp :any :trusted :raw spex lld ::st2))

    (implies
      (let ((hld::st1 (lift-raw spex lld::st1))
            (hld::st2 (lift-raw spex lld::st2)))
        (and
          (let ((segs (intersection-equal
                        (dia-fs seg hld::st1)
                         (segs-fs (current hld::st1) hld ::st1))))
            (equal (raw-selectlist segs hld ::st1)
                   (raw-selectlist segs hld ::st2)))
           (equal (current hld ::st1)
                  (current hld ::st2))
           (equal (raw-select seg hld ::st1)
                  (raw-select seg hld ::st2))))
      (equal
        (raw-select seg (lift-raw spex (lld::next spex lld ::st1)))
    (raw-select seg (lift-raw spex (lld::next spex lld ::st2)))))))
```

Figure 4: The two theorems *lift-next-commute* and *hld-separation* are needed to satisfy the Common Criteria requirements when using a 2-model Representation Architecture. The *lift-next-commute* theorem demonstrates that the two models are equivalent. The *hld-separation* theorem shows that the GWV separation policy holds for the high-level model. In contrast, if the 1-model representation Architecture is used then only the *lld-separtion* theorem needs to be proved.

Levels of Formality

The AAMP7 verification used fully formal representation of all layers of the assurance architecture. Most verifications will require only semiformal representations of various levels of the assurance architecture. Therefore, validating this system with a single model requires a model with a level of formalism equal to the highest in any of the

requirements. The added formalism may seem like unnecessary work, but this is more than compensated by the fact that the correspondence between the high and low-level models becomes trivial, since they are one and the same.

ACL2 is a good choice for semiformal representations. The strategic use of axioms can be viewed as semiformalism, under the condition that compelling informal/semiformal rationales on the validity of the axioms is provided. This has the advantage of providing a pathway to fully formal representations by later providing proofs of the axioms. The resulting representations still maintain the stackability property, allowing the development of assurance architectures and component composition. It is unclear how other restricted syntax languages would support these notions. Furthermore, using ACL2 for semiformal representations provides justification that the representations are internally consistent, which is required for Common Criteria semiformal representations.

3. Conclusion

The Common Criteria does not prescribe the use of any particular tool or notation to satisfy either its formal or semiformal representation requirements. However, ACL2 is well suited to provide either formal or semiformal representations in a Common Criteria certification context. ACL2 is adept at stacking models with increasing levels of abstraction. Not only does this directly support the Common Criteria design assurance architecture, it supports the use of properties of a certified system as assumptions in a higher level system. Furthermore, using ACL2 with a reasonable set of axioms for a semiformal presentation provides the ability to make the presentation fully formal by providing formal proofs of the axioms.

References

[1] *Common Criteria for Information Technology Security Evaluation,* Version 2.1, August 1999, CCIMB-99-031, http://csrc.nist.gov/cc/CC-v2.1.html.

[2] Special Issue on System Verification, with W.R. Bevier, W.A. Hunt, and W.D. Young. *Journal of Automated Reasoning*, Kluwer Academic Publishers, **5**(4), 1989, pp. 461-492.

[3] David Greve, Raymond Richards and Matthew Wilding, *A Summary of Intrinsic Partitioning Verification*, Fifth International Workshop on the ACL2 Prover and Its Applications (ACL2-2004), Austin, TX.

[4] David Greve, Matthew Wilding and W. Mark Vanfleet, "A Separation Kernel Formal Security Policy", Fourth International Workshop on the ACL2 Prover and Its Applications (ACL2-2003), Boulder, CO, July 2003.