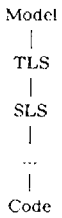
Towards a Definition of "Beyond A1" Verification

Ben Di Vito

TRW Defense Systems Group
One Space Park
Redondo Beach, CA 90278

In the Trusted Computer System Evaluation Criteria of the DoD Computer Security Center, the "A1" level of assurance is meant to capture the current state-of-the-art in formal verification technology. This corresponds to what is known as "design verification." The more rigorous "code verification" is considered by much of the Verification Community as beyond the state-of-the-art and hence a "beyond A1" assurance technique. We believe, however, that such characterizations are unnecessarily rigid and do not reflect the true state-of-the-art. Therefore, we propose a middle-ground approach for defining a useful "beyond A1" verification concept that can be realized with existing technology.

As we all know, the verification paradigm that gave rise to the terms "design verification" and "code verification" is based on a hierarchy of abstract state machine specifications, as exemplified by HDM and FDM. Above the Top Level Specification (TLS) is placed a security model (or other kind of application model), which expresses the desired system properties. The implementation or coding step is reached after the lowest level specification is elaborated.



Proofs of correspondence between adjacent levels are sufficient to guarantee that the code satisfies the requirements of the model.

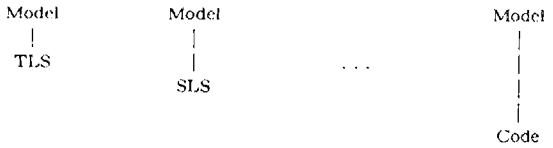
In current practice, only design verification (proof of TLS with respect to model) is carried out with mathematical rigor. Code verification (proof of code with respect to TLS) is considered beyond the state of the art for any nontrivial system. Indeed, for code of any appreciable size, it is clear that verification of code against a specification of full functionality is a formidable task.

Nevertheless, there are useful alternatives to verifying code against a full functionality specification. One alternative is to prove directly that the code satisfies the properties of the model. This can be done using conventional Floyd-Hoare techniques, as exemplified by Gypsy, AFFIRM and SPV. In fact, most applications of Gypsy have followed this approach.

The advantage of proving code directly to the model is low cost. Security properties embodied in a model are typically very weak properties compared to the total system functionality. Thus, it is far easier to prove the code with respect to a model than to a TLS. Moreover, we believe this observation accounts for the difference of opinion among members of the Verification Community regarding the feasibility of code verification. Generally speaking, those who hold that code verification is beyond the state-of-the-art subscribe to the prove-code-against-the-TLS school, while those who hold that code verification is feasible subscribe to the prove-code-against-the-model school. In reality, there is no difference of opinion: both sides are correct relative to their respective definitions of the problem.

Of course, the disadvantage of the direct approach is that the functional properties do not get verified. It will be necessary to establish them via testing and other conventional techniques. We feel this tradeoff is quite justifiable. It is better to prove rigorously today what we are capable of proving rather than settling for less satisfying proofs and waiting for technological advancements to come along and save the day. While we wait for this improved technology to emerge we will be missing many opportunities to apply current code verification technology to the production of trusted or reliable software.

To this end, we propose a scheme for "beyond A1" verification that preserves the notions of design and code verification. In its general form, successive refinements of specifications are still supported with the proviso that at each step a proof is performed back to the model rather than to the layer directly above.



In the end, the proof of code to model is the one that really counts, with the others merely being used as stepping stones to guide the design to a final implementation. This final proof is not as strong as one involving a proof of code to TLS, but it can be accomplished with the current level of technology whereas the other cannot. It offers us the possibility of constructing trusted systems whose implementations are verified to satisfy their security properties.

Using this alternate approach, the advantages of design verification are completely retained. In particular, errors in preliminary design stages can be detected and corrected before they are propagated to detailed design and implementation stages. The act of formalizing specifications, which is often touted to impart a better understanding of the problem domain to designers, can be performed very early with all its attendant benefits. Furthermore, formal specifications can be used to drive the test data generation process, with automation becoming available as the technology matures. If the formal specifications can be made executable as well, then the specifications themselves can be tested and incorporated into some kind of rapid prototyping facility.

From a practical standpoint, employing the proposed verification scheme in its full generality probably would not be cost effective. It should suffice instead to construct proofs only for the TLS and code; verifying the intermediate levels of specification (should they be written) is not likely to pay off for systems of moderate size. Thus, we are left with a nominal two-proof scheme:

Design verification and code verification under our proposal are defined in terms of these proofs against a common application model, although additional interpretation of the model may be necessary for code proofs. In some cases, special tools may be needed to conduct code verification, such as information flow analyzers for specific programming languages. Existing techniques appear to be adequate for accomplishing a substantial amount of code proof; it is primarily tools for specific languages that are lacking.

In essence, our proposal is a plea for flexibility in the way we apply current verification technology. Prevailing notions of design verification and code verification, while obviously valid, should not be prescribed as constituting the only permissible methodology. Until such time as someone produces a uniqueness proof for the validity of a given verification methodology, we should remain open to different approaches for applying our admittedly limited capabilities. The problems we face are difficult enough. We should not add to them by acquiescing to false intractabilities that result from our insistence on a particular approach. With a little bit of imagination, our existing knowledge and tools can be stretched to achieve far greater results than are currently perceived possible.

Position Paper: Verkshop III

Beyond Functional Behavior: Combining Methods to Specify Different Classes of Properties of Large Systems

Jeannette M. Wing[*]
Computer Science Department
University of Southern California
Los Angeles, California 90089-0782

1. Introduction to a Problem

Most of the past specification and verification technology has concentrated on the functional behavior of a piece of software. Once one begins to explore the possibility of the practical use of such technology for larger and larger pieces of software, one finds that showing the correctness of only the functional behavior of a system is not usually sufficient to satisfy the customer. Verifying other properties, such as reliability, security, performance, and real-time behavior, is in as much of the customer's interest as verifying functional behavior.

Work done on specifying and verifying one of these properties is typically done at best with the assumption that the other properties are satisfied or at worst in complete disregard of them. Little work has addressed the language and model issues of integrating the specification and verification of all these properties. A few exceptions include work done on fault-tolerance [Wensley 78] and more recently, on concurrency (e.g., [Lamport 83]), reliability [Cristian 83], and performance [Zave 84]. Properties such as response time and space efficiency, which both are of critical importance when analyzing the behavior of a large program, have thus far been ignored by most in the specification and verification communities.

We need to study the unaddressed task of combining the specifications of these different classes of properties. Such a combination should consequently yield a specification that more completely describes the behavior of a large program than a traditional specification that describes only the program's input-output behavior. Furthermore, we need to reassess the verification process in light of the proofs of these other properties. Performing the proofs themselves raises some challenging problems. Moreover, these proofs will not necessarily be independent of one another, and thus, the formal meaning of their combination, raises other theoretical and practical problems for verification technology.

In what follows I will dwell on only the specification end of the problem. In the third section, I will make some final remarks and briefly mention some of the corresponding problems at the verification end.

2. Towards a Solution: Combining Methods

Combining methods, languages, and models of specifications is a reasonable approach toward handling the problem of specifying properties of different classes. For example, for concurrency, a definitional method of specifying concurrent properties, e.g., via temporal logic, should blend in well with a definitional method of specifying (sequential) functional behavior, e.g., via algebraic specifications. In the same spirit, CCS and Meta-IV of the Vienna Definition Method would also be a good blend (see [Folkjar and Bjorner 80] for combining CSP and Meta-IV) of operational methods. With some work, it similarly should be possible to combine methods, languages, or models of specifications of reliability, performance, etc., with those of the input-output behavior of programs.

[*] This work was supported in part by the National Science Foundation under Grant No. ECS-8403905.