A Tool Bus for Anytime Verification*

Natarajan Shankar[1]

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
shankar@csl.sri.com
URL: http://www.csl.sri.com/~shankar/

Abstract. The science and technology of verification has advanced rapidly in recent years and it is now being used in substantive industrial projects. However, these opportunistic applications do not yet reflect a radical transformation of practice. Such a transformation can only come about through the flexible and pervasive use of formalization and formal analysis. Anytime Verification (AV) is a way of exploiting verification technology to the extent that it is fruitful while deriving the maximal benefit from it. The basic idea in AV is to systematically deepen the link between a design artifact and its putative properties to the level of rigor that is feasible. In AV, tools are used to the extent that they are robustly effective in supporting modeling, discharging proof obligations, optimizing code, and in generating counterexamples and test cases. The infrastructure for AV is provided by the Evidential Tool Bus (ETB) which supports interoperability between multiple inference tools in aid of specific verification tasks.

In order for something to be usable, it must first be useful. Verification technology is getting there. We can now use it to support the development of software through modeling, test generation, type checking, static analysis, abstraction, optimization, and certification. But these advances have not yet reached the programming masses. This chasm between the possibilities raised by formal technologies and currently entrenched practice is not easily bridged. Modern verification techniques offer a lot of entry points for the prospective user. Some users might focus on the modeling and specification capabilities, while others focus on finding bugs and security vulnerabilities in their software. How can we package the useful verification technology so that it is usable across a range of applications? We advocate an approach based on anytime verification (AV) where users can expoit the technology to the extent that it delivers value. We outline a framework for AV based on the SRI suite of verification tools (PVS, SAL/HybridSAL, Yices, PCE, and Simcheck) integrated using the prototype Evidential Tool Bus (ETB).

1 The Ecology of Verification Tools

Verification is used to prove or refute the property under test (PUT) of a design under test (DUT). The DUT may itself be an executable program fragment, and there

* This research was supported by NSF Grants CSR-EHCS(CPS)-0834810 and CNS-0917375, and by NASA Cooperative Agreement NNA10DE73C.

are several techniques that can be used for this purpose. For example, both the PUT and DUT are expressed as formulas in a logic and we attempt to prove the implication $\mathrm{DUT} \Rightarrow \mathrm{PUT}$, or provide a counterexample refuting it. The DUT can be expressed in a modeling or programming language from which proof obligations corresponding to the PUT are generated. Techniques such as binary decision diagrams, propositional satisfiability solvers (SAT), solvers for satisfiability modulo theories (SMT), and interactive theorem proving can be used to perform these proofs with varying degrees of automation. Static analysis (including typechecking) can be employed to refute or derive properties of the DUT. Alternately, the PUT can be expressed as a monitor and analyzed in composition with the DUT as is the case in model checking or in dynamic analysis. Designs and properties can be abstracted to simplify the task of finding proofs or refutations. Verification techniques can also be used to formalize PUTs from informal requirements, or to synthesize DUTs from the PUTs. We now have a wide range of modeling techniques including finite-state, discrete, continuous, andor hybrid models, many different ways of representing control, data, and communication, and a variety of notations for representing properties covering type correctness, assertional correctness, safety, liveness, and information flow.

In sum, verification has many uses, and there are many different ways of doing verification. The purest form of verification is when it is employed to carry out total correctness by establishing that an executable design, implemented in hardware, software, or a combination, satisfies its specification. However, this kind of verification is still partial. The specification might not cover all of the desirable properties, and both the design and the specification are subject to evolution. For this reason, verification should be regarded as part of the design process rather than the final word on the correctness of a design.

2 Anytime Verification

If we view verification as an integral part of the design process, then it is not enough to simply incorporate it into design tools. Verification must also support the evolutionary nature of design where new requirements are added, old ones are revised, and designs themselves are improved, modified, and adapted. This evolution is often triggered by issues that are uncovered during verification. We therefore need approaches to verification that are robust with respect to change. Complete soup-to-nuts verification is a admirable goal, but not always a worthwhile pursuit. Many projects can benefit from verification even when the extent of verification is partial. In such situations, it is more important to focus scarce verification resources on those facets that deliver the biggest payoff.

Anytime verification (AV) takes the approach that verification can enhance the reliability and efficiency of a design process. The technology forces the designers and developers to document the properties of their design, to consider examples and counterexamples for these properties, and to ensure that these properties are preserved at all phases of the design. AV allows verification to be used in a flexible way through modeling languages, type systems, assertion languages, composition frameworks, and run-time monitors. It provides a range of verification techniques from lightweight meth-

ods for typechecking and static and dynamic analysis, to the more heavy-weight ones such as interactive theorem proving. To exploit the *anytime* nature of AV, the design is structured to generate proof obligations. These proof obligations can either be refuted, discharged, or monitored. The use of assertions (or dependent typing) in specification and programming languages is a paradigmatic instance of AV.

3 A Tool Bus for Anytime Verification

For anytime verification to work, we need to go from an ecology of verification tools to an eco-system that can bridge the gap between the design frameworks and verification tools. We also need greater synergy between the verification tools themselves so that they can be composed efficiently to craft methods that are effective for specific classes of problems. Example methods include the hybrid techniques for test generation, counterexample-guided abstraction refinement (CEGAR), concolic testing, and the Yogi method for combining test generation and verification. Our methods should also be flexible with respect to incorporating legacy implementations and interfaces within a freshly designed system.

The basic infrastructure for AV comes from the Evidential Tool Bus (ETB) currently being developed at SRI in collaboration with John Rushby, Sam Owre, and Bruno Dutertre. In ETB, tools are registered with their invocation API. A verification *goal* is processed using a *script* that employs the registered tools. When a tool or a script succeeds, it generates a *judgement*, for example, that a given file contains a first-order logic formula that is valid. These judgements are composed into a *proof structure* for a verification claim that can be directly checked. The claim is composed of a conclusion and premises. These claims can be examined in several alternative ways to develop confidence, and can also be rechecked by highly-assured tools that have, for exampled, themselves been verified.

The current ETB prototype is being implemented in XSB Prolog. Tools can be invoked through scripts on specific goals. Each tool generates a claim supported by an argument in response to the goal. For example, the goal might require showing that a property is invariant over a transition system. The generated claim might display a counterexample demonstrating that the property is not inductive. This counterexample can be used to iteratively strengthen the invariant, until an inductive invariant implying the original property has been derived.

The ETB will be populated with modeling tools, test generators, static analyzers, satisfiability solvers, model checkers, and interactive provers. These tools can be used to generate counterexamples or proofs for verification claims. Scripts can be written using Prolog. Tools can integrated as shell commands, sub-processes, or foreign function calls. The ETB is semantically neutral: it manipulates tools, files, and judgements, but is not sensitive to their semantic content. For example, if a proof obligation generator takes an annotated program and generates proof obligations, then the ETB constructs the claim that the program is correct relative to its annotations by attempting to discharge all of the proof obligations. Any unproved assertions can be transformed into run-time checks. The eco-system of verification tools and intermediate languages makes it possible to quickly prototype verification methods that can be applied to real

designs. For example, we plan to use the tool bus to support our SimCheck tool for handling proof obligations from MATLAB Simulink designs. The ETB serves to bridge the gap between applications and tools by allowing tools to be combined in flexible ways while carefully constructing checkable proofs of reproducible claims.

4 Conclusions

It is always a challenge to make any advanced technology usable by a wider audience. While powerful verification tools are now freely available, only a narrow segment of these tools are used even by experts. The use of these tools is also restricted to a narrow range of applications. To increase the audience for verification technology, we have to ensure that the tools can be easily integrated and adapted to specific patterns of usage in modeling, test generation, run-time monitoring, type-checking extended with assertions, and program verification. Many of these applications might involve partial or light-weight approaches to verification that focus on bug-finding or optimization. The heavy-weight methods might be used only on the critical components. We propose the Evidential Tool Bus as a framework for rapidly building and embedding verification tools into design frameworks. This allows verification tools to be used in a wide range of applications ranging from test generation to the verification of total correctness properties.