

The Verified Software Roadmap: Interoperable Tools

Yves Bertot, John Harrison, Warren Hunt, Rupak
Majumdar, John Matthews, N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

Overview

Verification tools can be classified as

1. **Libraries:** Term representations, BDDs, arithmetic, automata
2. **Procedures:** SAT solvers, SMT solvers, typecheckers, rewriters, model checkers, static analyzers, code generators, invariant generators, termination checkers, optimization software, computer algebra
3. **Specification/verification systems, logical frameworks:** Combine above procedures to verify models or programs, or to generate and refine code from high-level descriptions.
4. **Proof libraries:** Definitions and theorems verified using theorem provers.

Goals for the Roadmap

- To outline a program of research leading to an open, powerful, integrated suite of tools that interoperate through a common semantic framework (a semantic toolbus).

Goals

To build a *robust suite of interoperable tools* that

1. Build on the best available theoretical (algorithmic and semantic) knowledge
2. Have been benchmarked extensively for performance and reliability
3. Offer clear, comprehensive, embeddable interfaces
4. Communicate through semantic interchange
5. Deliver evidence in the form of proofs, counterexamples, witnesses, and explanations
6. Interoperate through a tool bus architecture for embedded use

Libraries

In the short-term, more of the basic verification functionality has to be developed within libraries. These libraries must be

1. Versatile enough to handle a wide range of application scenarios (online, backtracking, resetting, explaining, propagating, persistent) without compromising on efficiency
2. Standardized and well-documented interface for interchangeability and interoperability, and standardized input formats
3. Embeddable in higher-level implementation languages
4. Benchmarked on an extensive and representative set of problems.

Libraries: The Long Term

Over the longer term, libraries in addition to providing versatility and speed, must be

1. Evidence producing so that results can be used to construct an assurance case.
2. Verified so that we've at least taken our own medicine.
3. Deeply interoperable so that each library can invoke functions from other libraries.

Tools: The Short Term

1. Moving more functionality into libraries
2. Decomposing system capabilities into individual tools.
3. Improvements in efficiency and accuracy through specialization, better use of semantic information, incrementality, modularity, abstraction, and approximation.
4. Exploit synergy through tool interoperation, proof/library reuse, problem decomposition.
5. Benchmarking on large codes and formulas (subject only to machine limitations).

Tools: A Semantic Tool Bus

The tool bus is an open architecture for tying together tools by exchanging semantic information.

The semantic information is represented in the form of judgments (well-formedness, typing, translation, validity, satisfaction).

Tools can be invoked implicitly or explicitly within the tool bus.

The tool bus itself can be embedded within various scripting frameworks and employed within integrated design environments.

Systems: The Short Term

The main short-term challenges are

1. Identifying useful and uniform intermediate languages.
2. Mapping well-defined subsets of real-world languages and models to the intermediate languages.
3. Developing verification methodologies that combine the capabilities of various tools (CEGAR).
4. Achieving push-button automation on most routine verification tasks.
5. Delivering useful diagnostic and assurance information.

Proof Library

- Topic-specific libraries/books
- Reasonably standardized across different systems.
- Translating proof libraries across different systems.
- Efficient Search/Browsing tools
- Maintenance and Stability

Summary

The challenge for the verification tools are

1. Bridging the semantic gap between logic and programming (*intermediate formats*)
2. Bridging the semantic gap between different tools and logics (*APIs and interchange formats*)
3. Developing modular tools (*architecture, interoperability*)
4. Providing tool support for verification-in-the-large (*abstraction, decomposition, slicing*)
5. Performing fast syntactic and semantic analyses on large programs and formulas (*tools and benchmarks*)
6. Delivering interaction and assurance in a form fit for human consumption (*evidence, interaction*)