PVS Wish List[1]

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

Aug 7, 2009

[1] This research was supported NSF Grants CNS-0823086 and CNS-0644783.

Overview

- Prototype Verification System (PVS) is an interactive theorem prover that combines automation and interaction.
- It also explores the synergy between language and inference (*the computational Sapir-Whorf hypothesis*) through features such as predicate subtypes, dependent types, parametric theories.
- PVS integrates a variety of external tools: MONA, Yices, BDDs, mu-calculus model checking.
- PVS is also a back-end tool for many other systems (InVesT, LOOP, ESC/Java2, Why, TAME, Rockwell-Collins, etc.).
- PVS is available in source code and is actively maintained.
- We list some ongoing and planned improvements.

Looking Back

- PVS has been available since 1993 and used in several projects.
- External users have contributed libraries and useful tools (PVSio, batchmode).
- We've taken the "prototype" aspect of this seriously to experiment with and evolve the system based on user feedback.
- Users have largely found the system quite easy to learn: typically few weeks.
- The language is quite complex: types, recursive data types, predicate and structural subtypes, recursive and co-recursive datatypes, parametric theories, and theory interpretations.
- Most of the features are extremely popular, but a simpler kernel language might be helpful.

Ground Evaluation

- The type system ensures that well-typed programs can only crash by exceeding resource bounds. Actually, it does a lot more than that.
- The ground evaluator does an update analysis to perform safe destructive updates.
- But many difficult features are not handled by the code generator, e.g., theory interpretations, possibly executable functions.
- New prototype ground evaluator does type-reified evaluation to instrument the code generation to be type-sensitive.
- We and others are also targetting other languages (e.g., Why, C).

Other Improvements

- Theory parameters are too heavy-handed for simple definitions, e.g., `map`, that can be made polymorphic at the declaration level.
- The model checker used in PVS is very old and needs to be upgraded to use the CUDD library.
- Nonlinear arithmetic: Grant Passmore has built the RAHD extension to PVS, but this needs a lot more work.
- Yices 1 is used as an end-game prover, but Yices 2 can be used to as an online decision procedure with multiple contexts.
- Better quantifer instantiation (next slide).
- Faster rewriting.

SMT-Backed Declarative Proof

- With declarative proofs, you want to build proofs from primitive demonstrations of the form $\Gamma \vdash A$.
- Each such step could be checked by an SMT solver using unsatisfiable cores to ensure minimality.
- This can be used to produce robust and readable proofs.