Appears in *Applied Formal Methods—FM-Trends 98*, Dieter Hutter, Werner Stephan, Paolo Traverso, and Markus Ullman Eds., Springer Verlag Lecture Notes in Computer Science Vol. 1641, pp. 338–345, Boppard, Germany, October 1998.

PVS: An Experience Report*

S. Owre, J. M. Rushby, N. Shankar, and D. W. J. Stringer-Calvert

Computer Science Laboratory, SRI International, Menlo Park CA 94025 USA
{owre, rushby, shankar, dave_sc}@csl.sri.com
URL: http://pvs.csl.sri.com

Abstract. PVS is a comprehensive interactive tool for specification and verification combining an expressive specification language with an integrated suite of tools for theorem proving and model checking. PVS has many academic and industrial users and has been applied to a wide range of verification tasks. In this note, we summarize some of its applications.

1 Introduction to PVS

PVS (Prototype Verification System) is an environment for constructing clear and precise specifications and for efficient mechanized verification. It is designed to exploit the synergies between language and deduction, automation and interaction, and theorem proving and model checking. The PVS specification language is a typed higher-order logic with a richly expressive type system with predicate subtypes and dependent types. Typechecking in this language requires the services of a theorem prover to discharge proof obligations corresponding to subtyping constraints.

The development of PVS began in 1990, and it was first made publicly available in 1993. Subsequent releases have increased its robustness and speed, and added a host of new capabilities. The essential features of PVS have already been described in prior publications [30, 32, 40], and comprehensive details can be found in the system manuals that are available from the PVS web site at `http://pvs.csl.sri.com`. In this note, we indicate the capabilities of the system through a survey of some of the applications for which it has been used. Due to space constraints, this is only a small sampling of the applications that have been performed using PVS, and even those that are mentioned are often given without full citations (we generally cite only the most accessible and the most recent works). We apologize to all PVS users whose work is omitted or mentioned without citation, and refer all readers to the online PVS Bibliography for a comprehensive list of citations to work concerning PVS [38].

We divide PVS activities and applications into a few broad subject areas: library development, requirements analysis, hardware verification, fault-tolerant algorithms, distributed algorithms, semantic embeddings/backend support, real-time and hybrid systems, security and safety, and compiler correctness.

* The development of PVS was funded by SRI International through Internal R&D funds. Various applications and customizations have been funded by NSF Grants CCR-930044 and CCR-9509931, and by contracts F49620-95-C0044 from AFOSR, NAS1-20334 from NASA, and N00015-92-C-2177 from NRL.

2 PVS Library Development

A major cost in undertaking formal specification and verification is that of developing formalizations for all the "background knowledge" that is required. PVS libraries help reduce this cost by providing formalizations for many common mathematical domains. Good libraries are challenging to develop: not only must they provide foundational definitions and axiomatizations that are correct, together with a body of derived constructions and lemmata that are rich enough to support development of clean, succinct, and readable specifications, but they must express these in a way that allows the PVS theorem prover to make effective use of them.

The "prelude" library built in to PVS provides many useful definitions and theorems covering basic mathematical concepts such as sets, bags, functions, relations, and orderings, together with properties of real and integer arithmetic outside the domain of the PVS decision procedures (principally those involving nonlinear arithmetic).

External PVS libraries provide finite sets, floor and div/mod, bitvectors, coalgebras, real analysis, graphs, quaternions, μ-calculus, and linear and branching time temporal logics. Development of libraries is very much a community effort in which sharing, modification, and extension has allowed the PVS libraries to grow into effective, robust and reusable assets. For example, the library for undirected graphs was developed by NASA Langley to support a proof of Menger's theorem [7]. This was extended to directed graphs by the University of Utah to support analysis of PCI bus transactions [28], and subsequently re-adopted and generalized by NASA.

3 Requirements

There is extensive evidence that requirements capture is the most error-prone stage in the software engineering lifecycle, and that detection and removal of those errors at later stages is very costly. Requirements provide a fruitful application area for formal methods because relatively "lightweight" techniques have proved effective in detecting numerous and serious errors. PVS supports these activities by providing direct support for consistency and completeness checking of tabular specifications [31], and through the process of "formal challenges" [39] where expected properties are stated of a specification and examined by theorem proving or model checking.

PVS has been used by multiple NASA centers to analyze requirements for the Cassini Spacecraft [13] and for the Space Shuttle [9], and by the SafeFM project (University of London) in the analysis of requirements for avionics control systems [12].

4 Hardware Verification

Applications of PVS to hardware verification fall into two broad classes. One class is concerned with verification of the complete microarchitecture against the instruction set architecture seen by machine code programmers. While the presence of pipelining and other optimizations introduces complexities, the basic approach to this class of verifications depends on efficient symbolic simulation and equality reasoning, which in PVS are achieved by its tight integration of cooperating decision procedures with rewriting, combined with BDD-based simplification. PVS has been used for the full or partial verification of microcoded avionics and Java processors developed by Rockwell Collins [18], as well as for a number of smaller DLX-like processors with complex pipelines.

The other class of hardware applications concerns the complex circuits, algorithms, and protocols that are the building blocks of modern processors; these applications are sufficiently difficult that success depends on finding an effective methodology. Examples include verification of SRT dividers and other arithmetic circuits at NASA [27] and SRI, out-of-order execution at the University of Utah and SRI [23] and the Weizmann Institute [36], and cache-coherence at Stanford University [33]. Some applications are best handled using a combination of tools; PVS was used in this way by Fujitsu for the validation of the high-level design of an ATM switch [37].

5 Fault-Tolerant Algorithms

Mechanisms for fault tolerance are a significant component of many safety-critical systems: they can account for half the software in a typical flight-control system, and are sufficiently complicated that they can become its primary source of failure! Verifications of practical fault-tolerant designs are quite difficult and are often achieved incrementally, as more real-world complexities are layered on to a basic algorithm. The parameterized theories and strict dependency checking of PVS help in these incremental constructions.

For example, formal analysis of Byzantine fault tolerant clock synchronization has been elaborated over nearly a decade, with contributions from SRI and NASA Langley (using a predecessor to PVS) and the University of Ulm, culminating in verification of the algorithm used in a commercial system for safety-critical automobile control [35]. Comparable developments at SRI, NASA, Allied Signal, and Ulm have verified practical algorithms for consensus, diagnosis, and group membership, together with overall architectures for state machine replication and time-triggered execution of synchronous algorithms.

6 Distributed Algorithms

The fault tolerance applications described above employ synchronous algorithms. Other distributed algorithms are often asynchronous and are generally modeled as transition relations. Safety properties are traditionally verified by invariance arguments, and generation of suitably strong invariants is the major methodological challenge. More recent approaches employ abstraction to a finite-state (or other tractable) system that can be model checked. PVS has a model checker integrated with its theorem prover, so that it is able to perform all the stages of such approaches. Examples include communications protocols [19] and garbage collection algorithms, parallel simulation algorithms [44] and parallelizing techniques [8], and operating system buffer-cache management [34].

Current research focusses on methods for automating the generation of abstractions and invariants [1, 5, 41].

7 Semantic Embeddings and Backend Support

For some applications it is convenient to use a customized logic for both specification and reasoning. Such logics can be encoded in the higher-order logic of PVS using either shallow or deep semantic embeddings. Examples include the Duration Calculus [42], DisCo [26], the B method [29], and coalgebraic treatments of Java classes [25]. An advantage of these embeddings over dedicated verification support is that the full expressiveness and power of PVS is available for all the auxiliary concepts and data types that are required.

An API for semantic embeddings of other logics is currently under development; this will allow specifications and proofs to be presented directly in the notation of the embedded logic.

An alternative to semantic embedding is to use PVS to discharge proof obligations generated by the support tool for another language. This route has been explored at Michigan State [20] and Bremen [6] universities.

8 Real-Time and Hybrid Systems

Formal treatments of real-time systems often employ special temporal or Hoare logics. Some of these have been supported by semantic embedding in PVS, as described above; others include timed automata [4], the language Trio [2], and the compositional method of Hooman [22]. Applications include several standard test-pieces, such as the Fisher's mutual exclusion algorithm, the Generalized Railroad Crossing, and the Steam Boiler, as well as some realistic protocols.

A real-time kernel for supporting Ada95 applications on a uniprocessor embedded system has also been developed in PVS at the University of York [14].

9 Security and Safety

Strong protection of data belonging to different processes is required for both security and safety in several applications. A formulation of this property in terms of "noninterference" forms one of the PVS tutorial examples. More elaborate and realistic treatments based on the same idea have been developed for security at Secure Computing Corporation [21], and for safe "partitioning" in avionics at NASA [45] and Rockwell Collins [49].

Ongoing work at SRI is developing an efficient approach for the verification of cryptographic protocols, while the special security problems arising in active networks have been formalized at the University of Cincinnati [11].

10 Compiler Correctness

In most system developments, correctness of the translation from source code to object code is not a source of major concern. Testing is performed on object code, which is fortuitously effective in finding errors introduced during compilation, assembly and linking. For critical developments, however, further assurance may be required.

PVS has been used to perform the verification of a compiler for a small safety critical language [43], and to reason about object code in terms of flow graphs [47]. The Verifix project (`http://i44s11.info.uni-karlsruhe.de/~verifix/`) at the Universities of Karlsruhe, Kiel, and Ulm has verified several compilation and optimization algorithms (including some expressed as abstract state machines, ASMs, where errors were found) and has also developed a collection of PVS theories for reasoning about operational and denotational semantics in this context. Another application related to programming language implementation is the security of Java style dynamic linking [10].

11 Summary

The applications sketched above give an idea of the range of projects for which PVS has been used and also provide a resource for those undertaking similar work. Additional descriptions can be found in the PVS Bibliography, which provides over 250 citations [38].

The development of PVS has been strongly influenced by practical applications and by feedback from users, and we expect this to continue. Enhancements currently in progress include direct and very fast execution for a substantial subset of the PVS language (this supports computational reflection [46], as well as improved validation of specifications [16]), and faster and more automated theorem proving. Those planned for the near future include support for refinement and a more open system architecture.

References

1. Parosh Aziz Abdulla, Aurore Annichini, Saddek Bensalem, Ahmed Bouajjani, Peter Habermehl, and Yassine Lakhnech. Verification of infinite-state systems by combining abstraction and reachability analysis. In Halbwachs and Peled [17], pages 146–159.
2. Andren Alborghetti, Angelo Gargantini, and Angelo Morzenti. Providing automated support to deductive analysis of time critical systems. In Mehdi Jazayeri and Helmut Schauer, editors, *Software Engineering—ESEC/FSE '97: Sixth European Software Engineering Conference and Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, volume 1301 of *Lecture Notes in Computer Science*, pages 211–226, Zurich, Switzerland, September 1997. Springer-Verlag.
3. Rajeev Alur and Thomas A. Henzinger, editors. *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, New Brunswick, NJ, July/August 1996. Springer-Verlag.
4. Myla Archer and Constance Heitmeyer. Mechanical verification of timed automata: A case study. In *IEEE Real-Time Technology and Applications Symposium (RTAS'96)*, pages 192–203, Brookline, MA, June 1996. IEEE Computer Society.

5. Saddek Bensalem, Yassine Lakhnech, and Hassen Saïdi. Powerful techniques for the automatic generation of invariants. In Alur and Henzinger [3], pages 323–335.
6. Bettina Buth. PAMELA + PVS. In Michael Johnson, editor, *Algebraic Methodology and Software Technology, AMAST'97*, volume 1349 of *Lecture Notes in Computer Science*, pages 560–562, Sydney, Australia, December 1997. Springer-Verlag.
7. Ricky W. Butler and Jon A. Sjogren. A PVS graph theory library. NASA Technical Memorandum 1998-206923, NASA Langley Research Center, Hampton, VA, February 1998.
8. Raphaël Couturier and Dominique Méry. An experiment in parallelizing an application using formal methods. In Hu and Vardi [24], pages 345–356.
9. Judith Crow and Ben L. Di Vito. Formalizing Space Shuttle software requirements: Four case studies. *ACM Transactions on Software Engineering and Methodology*, 7(3):296–332, July 1998.
10. Drew Dean. Static typing with dynamic linking. In *Fourth ACM Conference on Computer and Communications Security*, pages 18–27, Zurich, Switzerland, April 1997. Association for Computing Machinery.
11. Darryl Dieckman, Perry Alexander, and Philip A. Wilsey. ActiveSPEC: A framework for the specification and verification of active network services and security policies. In Nevin Heintze and Jeannette Wing, editors, *Workshop on Formal Methods and Security Protocols*, Indianapolis, IN, June 1998. Informal proceedings available at `http://www.cs.bell-labs.com/who/nch/fmsp/program.html`.
12. Bruno Dutertre and Victoria Stavridou. Formal requirements analysis of an avionics control system. *IEEE Transactions on Software Engineering*, 23(5):267–278, May 1997.
13. Steve Easterbrook, Robyn Lutz, Richard Covington, John Kelly, Yoko Ampo, and David Hamilton. Experiences using lightweight formal methods for requirements modeling. *IEEE Transactions on Software Engineering*, 24(1):4–14, January 1998.
14. Simon Fowler and Andy Wellings. Formal development of a real-time kernel. In *Real Time Systems Symposium*, pages 220–229, San Francisco, CA, December 1997. IEEE Computer Society.
15. Ganesh Gopalakrishnan and Phillip Windley, editors. *Formal Methods in Computer-Aided Design (FMCAD '98)*, volume 1522 of *Lecture Notes in Computer Science*, Palo Alto, CA, November 1998. Springer-Verlag.
16. David Greve. Symbolic simulation of the JEM1 microprocessor. In Gopalakrishnan and Windley [15], pages 321–333.
17. Nicolas Halbwachs and Doron Peled, editors. *Computer-Aided Verification, CAV '99*, volume 1633 of *Lecture Notes in Computer Science*, Trento, Italy, July 1999. Springer-Verlag.
18. David Hardin, Matthew Wilding, and David Greve. Transforming the theorem prover into a digital design tool: From concept car to off-road vehicle. In Hu and Vardi [24], pages 39–44.
19. Klaus Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. In *Formal Methods Europe FME '96*, volume 1051 of *Lecture Notes in Computer Science*, pages 662–681, Oxford, UK, March 1996. Springer-Verlag.
20. Mats P. E. Heimdahl and Barbara J. Czerny. Using PVS to analyze hierarchical state-based requirements for completeness and consistency. In *IEEE High-Assurance Systems Engineering Workshop (HASE '96)*, pages 252–262, Niagara on the Lake, Canada, October 1996.
21. John Hoffman and Charlie Payne. A formal experience at Secure Computing Corporation. In Hu and Vardi [24], pages 49–56.

22. Jozef Hooman. Compositional verification of real-time applications. In Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors, *Compositionality: The Significant Difference (Revised lectures from International Symposium COMPOS'97)*, volume 1536 of *Lecture Notes in Computer Science*, pages 276–300, Bad Malente, Germany, September 1997. Springer-Verlag.
23. Ravi Hosabettu, Mandayam Srivas, and Ganesh Gopalakrishnan. Proof of correctness of a processor with reorder buffer using the completion functions approach. In Halbwachs and Peled [17], pages 47–59.
24. Alan J. Hu and Moshe Y. Vardi, editors. *Computer-Aided Verification, CAV '98*, volume 1427 of *Lecture Notes in Computer Science*, Vancouver, Canada, June 1998. Springer-Verlag.
25. Bart Jacobs, Joachim van den Berg, Marieke Huisman, Martijn van Berkum, Ulrich Hensel, and Hendrick Tews. Reasoning about Java classes. In *Proceedings, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'98)*, pages 329–340, Vancouver, Canada, October 1998. Association for Computing Machinery. Proceedings issued as ACM SIGPLAN Notices Vol. 33, No. 10, October 1998.
26. Pertti Kellomäki. Verification of reactive systems using DisCo and PVS. In *Formal Methods Europe FME '97*, volume 1313 of *Lecture Notes in Computer Science*, pages 589–604, Graz, Austria, September 1997. Springer-Verlag.
27. Paul S. Miner and James F. Leathrum, Jr. Verification of IEEE compliant subtractive division algorithms. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods in Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 64–78, Palo Alto, CA, November 1996. Springer-Verlag.
28. Abdel Mokkedem, Ravi Hosabettu, and Ganesh Gopalakrishnan. Formalization and proof of a solution to the PCI 2.1 bus transaction ordering problem. In Gopalakrishnan and Windley [15], pages 237–254.
29. César Muñoz. PBS: Support for the B-method in PVS. Technical Report SRI-CSL-99-1, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1999.
30. S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In Alur and Henzinger [3], pages 411–414.
31. Sam Owre, John Rushby, and N. Shankar. Integration in PVS: Tables, types, and model checking. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '97)*, volume 1217 of *Lecture Notes in Computer Science*, pages 366–383, Enschede, The Netherlands, April 1997. Springer-Verlag.
32. Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.
33. Seungjoon Park and David L. Dill. Verification of cache coherence protocols by aggregation of distributed transactions. *Theory of Computing Systems*, 31(4):355–376, 1998.
34. N.S. Pendharkar and K. Gopinath. Formal verification of an O.S. submodule. In V. Arvind and R. Ramanujin, editors, *18th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1530 of *Lecture Notes in Computer Science*, pages 197–208, Madras, India, December 1998. Springer-Verlag.
35. Holger Pfeifer, Detlef Schwier, and Friedrich W. von Henke. Formal verification for time-triggered clock synchronization. In Weinstock and Rushby [48], pages 207–226.

36. Amir Pnueli and Tamara Arons. Verification of data-insensitive circuits: An in-order-retirement case study. In Gopalakrishnan and Windley [15], pages 351–368.
37. S. P. Rajan, M. Fujita, K. Yuan, and M. T-C. Lee. ATM switch design by high level modeling, formal verification, and high level synthesis. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 3(4):554–562, October 1998.
38. John Rushby. PVS bibliography. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA. Constantly updated; available at `http://www.csl.sri.com/pvs-bib.html`.
39. John Rushby. Formal methods and their role in the certification of critical systems. In Roger Shaw, editor, *Safety and Reliability of Software Based Systems (Twelfth Annual CSR Workshop)*, pages 1–42, Bruges, Belgium, September 1995. Springer. Also to be issued as part of the *FAA Digital Systems Validation Handbook* (the guide for aircraft certification).
40. John Rushby, Sam Owre, and N. Shankar. Subtypes for specifications: Predicate subtyping in PVS. *IEEE Transactions on Software Engineering*, 24(9):709–720, September 1998.
41. Hassen Saïdi and N. Shankar. Abstract and model check while you prove. In Halbwachs and Peled [17], pages 443–454.
42. Jens U. Skakkebæk and N. Shankar. Towards a Duration Calculus proof assistant in PVS. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 660–679, Lübeck, Germany, Sept. 1994. Springer-Verlag.
43. David W. J. Stringer-Calvert. *Mechanical Verification of Compiler Correctness*. PhD thesis, University of York, Department of Computer Science, York, England, March 1998. Available at `http://www.csl.sri.com/~dave_sc/papers/thesis.html`.
44. Kothanda Umamageswaran, Krishnan Subramani, Philip A. Wilsey, and Perry Alexander. Formal verification and empirical analysis of rollback relaxation. *Journal of Systems Architecture (formerly published as Microprocessing and Microprogramming: the Euromicro Journal)*, 44(6–7):473–495, March 1998.
45. Ben L. Di Vito. A model of cooperative noninterference for integrated modular avionics. In Weinstock and Rushby [48], pages 269–286.
46. Friedrich von Henke, Stephan Pfab, Holger Pfeifer, and Harald Rueß. Case studies in meta-level theorem proving. In Jim Grundy and Malcolm Newey, editors, *Theorem Proving in Higher Order Logics: 11th International Conference, TPHOLs '98*, volume 1479 of *Lecture Notes in Computer Science*, pages 461–478, Canberra, Australia, September 1998. Springer-Verlag.
47. M. Wahab. Verification and abstraction of flow-graph programs with pointers and computed jumps. Research Report CS-RR-354, Department of Computer Science, University of Warwick, Coventry, UK, November 1998. Available at `http://www.dcs.warwick.ac.uk/pub/reports/rr/354.html`.
48. Charles B. Weinstock and John Rushby, editors. *Dependable Computing for Critical Applications—7*, volume 12 of *Dependable Computing and Fault Tolerant Systems*, San Jose, CA, January 1999. IEEE Computer Society.
49. Matthew M. Wilding, David S. Hardin, and David A. Greve. Invariant performance: A statement of task isolation useful for embedded application integration. In Weinstock and Rushby [48], pages 287–300.

The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.