# Protocol verification using PVS

Petr Matoušek, 2. ročník PDS, ÚIVT FEI
školitel: Doc.Ing.Miroslav Švéda

***Abstrakt***

*For successful design and building of a real-life distributed system based on communication protocols further analysis of the specified communication protocol is required. This analysis can reveal possible bottle-necks in real service and it can prove that the protocol specification satisfies real-time properties and requirements of the service. It is not feasible to verify systems like communication protocols by hand. There are automated techniques and tools which can help the designer to prove required properties. One of the popular verification system is Prototype Verification System (PVS) developed at SRI International's Computer Science Laboratory (CSL).*

*This paper introduces first-look experiences with the PVS system. It shows how the system can be used for the verification of communication protocols.*

**Keywords:** protocol specification, formal methods, verification, theorem prover

## 1 Introduction

Communication protocols are an important class of concurrent algorithms that pose a difficult challenge for existing verification technologies [Hav96]. Tools based on model checking and state exploration are effective and widely used for protocol verification, but many real-life protocols are not finite state and cannot be fully analyzed by these methods.

The state spaces of real-life protocols are either infinite or too large for finite-state verification techniques like model checking and state exploration. For that reason theorem proving seems to be a good solution to prove real-life communication protocols.

In this paper the PVS system is presented as a verification tool that can help a designer to prove protocol properties. It is a theorem prover based on higher-order logic. It includes decision procedures for arithmetic, automatic rewriting, induction, and other relatively large units of deduction. Proving is interactive and manage by the user which select appropriate rules and techniques.

In the course of a proof, PVS builds up a tree of sequent where each sequent is a subgoal generated from its parent sequent by a PVS proof command. At any point in a proof attempt, the control is at a leaf sequent of such a proof tree. At this point a PVS proof command can either

- cause control to be transferred to next proof sequent in the tree
- undo a subtree by causing control to move up to some ancestor node in the proof tree
- prove the current sequent causing control to move to the next remaining leaf sequent in the tree
- generate subgoals so that control moves to the first of these subgoals, or
- leave the proof tree unchanged while providing some useful status information.

A proof is completed when there are not remaining unproved leaf sequents in the proof tree. The resulting proof is saved and can be edited and rerun on the same or a different conjecture.

## 2 Verification techniques

Two well-established approaches to verification are model checking and theorem proving. They go one step beyond specification; these formal methods are used to analyze a system for desired properties.

### 2.1 Model Checking

Model checking is a technique that relies on building a finite model of a system and checking that a desired property holds in that model:

$$system \models requirements$$

Roughly speaking the check is performed as an exhaustive state space search which is guaranteed to terminate since the model is finite.

In contrast to theorem proving, model checking is completely automatic. Model checking can be used to check partial specifications, and it can provide useful information about a system's correctness even if the system has not been completely specified. Above all, model checking produces counterexamples, which usually represent subtle errors in design, and thus can be used to aid in debugging.

### 2.2 Theorem Proving

Theorem proving is a technique where both the system and its desired properties are expressed as formulas in some mathematical logic. This logic is given by a *formal system*, which defines a set of axioms and a set of inference rules. Theorem proving is the process of finding a proof of a property from the axioms of the system. Steps in the proof appeal to the axioms and rules, and possibly derived definitions and intermediate lemmas.

$$background + assumpt. + system \vdash requirements$$

In contrast to model checking, theorem proving can deal directly with infinite state space. It relies on techniques like structural induction to prove over infinite domains. Interactive theorem provers, by definition, require interaction with a human, so the theorem proving process is slow and often error-prone. In the process of finding the proof, however, the human user often gains invaluable insight into the system or the property being proved.

## 3 PVS

PVS (Prototype Verification System) [Crow95] is a verification system for writing formal specifications and checking formal proofs. It consists of a specification language integrated with support tools and a theorem prover. It provides an integrated environment for the development and analysis of formal specifications, and supports a wide range of activities involved in creating, analyzing, modifying, managing and documenting theories and proofs.

The specification language of PVS is a highly expressive language based on higher-order logic. The language can describe computer systems; however, it concentrates on abstract descriptions rather than detailed prescriptions (*what* rather than *how*). A specification can be constructed using definitions or axioms, or a combination of the two.

PVS includes a powerful interactive theorem prover or proof checker. The PVS proof checker manages the proof construction process by prompting the user for a suitable command for a given subgoal. The execution of the given command can either generate further subgoals or complete a subgoal and move control over to the next subgoal in a proof.

PVS differs from most other interactive theorem provers in the power of its basic steps: these can invoke decision procedures for arithmetic, automatic rewriting, induction, and other relatively large units of deduction; it differs from other highly automated theorem provers in being directly controlled by the user.

Model-checking capabilities used for automatically verifying temporal properties of finite-state systems have recently been integrated into PVS.

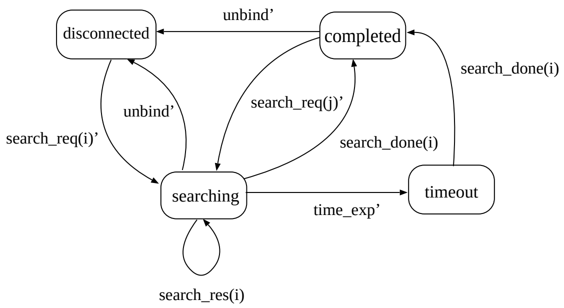
PVS is implemented in Common Lisp - with ancillary functions provided in C, Tcl/Tk, and LaTeX, and uses GNU Emacs for its interface. It is configlured either for Sun SPARC Workstations running SunOS 4.1.3, or for Intel x86 with Redhat Linux 4 or 5.

PVS, developed at SRI, can be downloaded from *http://pvs.csl.sri.com/*. It can be freely used for non-commercional purposes.

## 4 Case study

As a case study to demonstrate verification of a high-level communication protocol the LDAP protocol has been chosen. LDAP protocol is a client/server application protocol defined by RFC - see [Wahl97]. It is based on X.500 standard while improving its features (simple implementation over TCP/IP, authentication, replication etc.).

At the first phase the simplified version of the *search* command is considered. The abstract model is described by a finite-state machine (see figure) and has been derived from the RFC standard [Wahl97].




Specified model is described by PVS specification languages and its consistency is verified by PVS system.

Temporal properties of the model can be specified as theorems that has to be proved. An example of liveness property is ***"A system eventually returns a response for every request that was sent."*** That can be formally described using standard temporal logic as

$$\vdash search\_req \Rightarrow \diamond(search\_done \vee error)$$

The formula written in standard temporal logic has to be firstly translated to temporal logic supported by the verification tool and then verified.

# 5 Conclusion

The work presented here is a part of my PhD thesis. It deals with verification of communication protocols by verification tools. There are a few case studies demonstrated usage of these tools on protocols. However, the formal description of communication properties is limited by expressive power of a tool. The aim of my project is to propose a method for verification of real-time properties of the system.

The kernel of the project will be developing a method supporting the specification and design of distributed applications based on high-level communication protocols. Theoretical results will be applied on a specific case-study - LDAP protocol, which is a typical example of distributed real-time communication protocols.

Verification of real-time properties is a semidecidable problem. The success of the proving depends mostly on the proof strategy. In this work I would like to test some strategies and make a recommendation for other studies.

# Reference

[Cla96] Clarke, E.M., Wing, J.M.: *Formal Methods: State of the Art and Future Directions*, CMU-CS-96-178, August 1996.

[Crow95] Crow J., Owre S., Shankar N., Srivas M.: *A Tutorial Introduction to PVS*, SRI Int., 1995, http://pvs.csl.sri.com/.

[Hav96] Havelund, K., Shankar, N.: *Experiments in Theorem Proving and Model Checking for Protocol Verification*, LNCS 1051, Springer-Verlag, March 1996, Pages 662-681.

[MRV00] Matoušek, P., Ráb, J., Výšek, P.: *Protocol Proving and Model-Checking: A First-Look Experience*, In Proceedings of IEEE/IFIP FSCBS Joint Workshop, April 2000, Edinburgh, Pages 71-75.

[Rush97] Rushby, J.: *Specification, Proof Checking, and Model Checking for Protocols and Distributed Systems with PVS*, FORTE/PSTV, Osaka, November 1997.

[Wahl97] Wahl, M., Howes, T., Kille, S.: *Lightweight Directory Access Protocol (v3)*, RFC 2251.