Protocol Proving and Model Checking: A First-Look Experience

Petr Matoušek, Jaroslav Ráb, Pavel Výšek
Department of Computer Science and Informatics,
Technical University Brno, Czech republic
email: {matousp, rabj,vysek}@dcse.fee.vutbr.cz

April 7th, 2000

Roadmap

1. **Introduction**
2. **Verification tools**
   - PVS (Prototype Verification System)
   - SPIN
   - Isabelle
3. **Case-study - LDAP protocol**
4. **Future steps**

1 Introduction

1.1 Motivation

- Our research is focused on communication protocols from the application point of view:
  - Industry protocols as fieldbus protocols
  - Internet protocols - application level
- We are faced the problem how successfully verify systems based on these protocols.
- There are various verification tools to be used for this task.

2 Verification tools

- **Tools based on model checking and state exploration**
  - The system - a finite transition system
    - a required property as a formula in a temporal logic.

$$system \models requirements$$

  - Real-life protocols are not finite state and cannot be fully analyzed - state explosion problem.
    - Must generally abstract them or simplify them.
  - Often cannot express full requirements.
  - Mostly automatic.
  - State space searching is time and space consuming.

- **Tools based on theorem proving**
  - The system, required properties, assumptions and background theories specified as formula in a single logic

    $$background + assumptions + system \vdash requirements$$

  - They are not full automatic - they require interactive guidence.
  - Often require skill and patience.

- **Combination of these two techniques**
  - Proposed as useful methodology for protocol verification (Havelund, Shankar).
  - The steps of mechanized verification:
    1. A scaled-down version of the protocol is debugged using state exploration.
    2. This scaled-down version is generalized to recover the full version of the protocol for verification using theorem proving.
    3. Theorem proving is used to abstract out a finite-state protocol whose correctness (when established by model checking) implies the correctness of original protocol.

2.1 PVS - Prototype Verification System

2.1.1 Introduction

- PVS is verification system for writing formal specifications and checking formal proofs.
- The PVS specification language
  - features common programming language constructs (arrays, functions, records),
  - allows embedding of special logics, ie. property specifications can be expressed in temporal logics.
- PVS integrates a theorem prover, model checker, and ancillary tools.
- The PVS verifier is used to determine if the desired properties hold in the specification of the model.
- PVS provides model checking for properties expressed in CTL (Computation Tree Logic).

2.1.2 Distribution and installation

- Freely distributed under licencse by SRI from *http://www.csl.sri.com/pvs.html*.
- Implemented in Common Lisp.
- Runs on SunOS, IBM AIX, Linux.
- Needs 100 MB on disk, at least 50 MB RAM and 100 MB swap space.
- Required software:
  - Emacs v.19.34 or later,
  - Tcl/Tk v.7.3/3.6 or later,
  - LaTeX2.09 or 2.e.

2.2 Spin

2.2.1 Introduction

- Validation system for models of distributed software systems.
- Uses high level description language Promela to specify system.
- In course of verification process, SPIN reports on deadlock, unspecified reception, flags incompleteness, and race conditions.
- Correctness properties specified using assertion or linear temporal logic (LTL) theorems.

2.2.2 Distribution and installation

- SPIN is distributed with all sources (*ftp://netlib.bell-labs.com/netlib/spin*).
- Runs on Unix, Windows 95/98, Windows NT.
- Implemented in YACC and Ansi/POSIX C.
- Optional graphical interface (XSPIN) written in Tcl/Tk.

2.3 Isabelle

2.3.1 Introduction

- Theorem prover for using sizable applications.
- Isabelle language is based on lamda-calculus syntax.
- The Isabelle distribution includes:
  - HOL (Higher-order logic),
  - HOLCF (Scott's Logic for Computable Functions),
  - FOL (First-order logic),
  - ZF (Zermelo-Fraenkel set theory on top of FOL).
- Automated verification only for FOL.

2.3.2 Distribution and installation

- Free from non-commercial use (*http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/*).
- Runs on Linux, Unix.
- Needs about 32-64 MB of memory (depending on ML installed).
- Required software:
  - a full Standard ML Compiler,
  - GNU bash shell v.1.x or 2.x,
  - Perl 5.x.

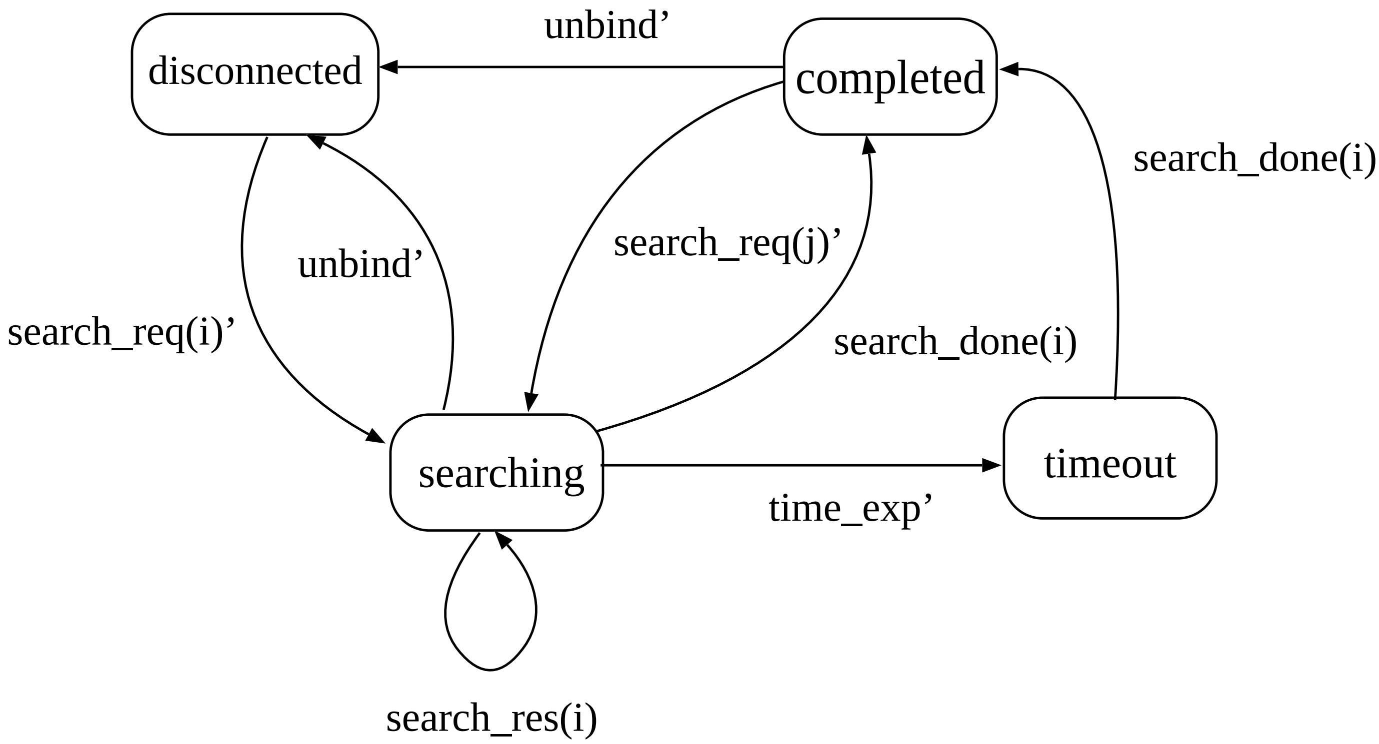
3 Case Study - LDAP Protocol

3.1 LDAP Protocol - Introduction

- Directory access protocol standardized in RFC 2251 (v.3).
- Client/server protocol.
- Application protocol based over TCP/IP.
- LDAP operation includes:
  - bind, modify, add, delete, compare, abandon,
  - unbind, search (tested)
- The LDAP protocol supports distributed directory by
  - data replication,
  - refferals.

3.2 Testing LDAP

- **The work reported here is very much in progress**.
- We model a *search* operation on the server side.
- Our specification is a derived from RFC 2251.
- Abstract model is described by Finite-State Machine.
- Formal specification to be written in
  - in PVS specification language,
  - in Promela,
  - in lambda calculus syntax of Isabelle.



4 Future steps

- At first we test just a simple properties modelled as theorems:
  - **A system eventually returns a response for every request that was sent (liveness)**.

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

- Next steps
  - to investigate more deeply verification techniques:
    * to compare model checking vs. theorem proving,
    * to apply the methodology of protocol testing for the downscaled version of the LDAP operation *search*,
    * to verify the original (full) search operation.
  - to extend the model by adding timeout requirements.
  - to model a distributed search operation (by following referrals).