Protocol Proving Using PVS: A Case Study

Petr Matoušek *
matousp@dcse.fee.vutbr.cz

Abstract: Prototype Verification System (PVS) is a popular verification tool for writing formal specification and checking formal proofs. It consists of a specification language integrated with support tools and a theorem prover. This paper shows its application on the class of high-level communication protocols. Case study is demonstrated on a simple protocol for user database access. The paper discusses problems of formal specification of communication protocols, its representation using PVS language and a set of properties to be proved.

Keywords: communication protocol, formal specification, verification, PVS, theorem proving

1 Introduction

Many services today are built to be accessed via Internet. Their data exchange and communication is based on a standard client-server architecture. Communication sequences exchanged between two partners are called *a protocol*. The protocol creates an independent way of communication over Internet irrespective of operating systems, communication interfaces etc. It defines what kind of data is expected by a partner and what kind of response is to be replied.

For successful design and building of a real-world protocol formal specification of communication is required. The specification describes the system, shows its main features and behaviour and makes further analysis of the service possible. Formal specification is a good base for system verification and simulation that can reveal possible bottle necks and flaws in design.

This paper shows how Prototype Verification System (PVS) can be used for verification of high-level protocols. PVS [1] is a verification system for writing formal specifications and checking formal proofs. 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 paper presents an example of the protocol verification on a simple protocol for user database access. The first part of the paper discusses a choise of formal methods for protocol description. The example is described using a regular grammar. Then it is shown how this regular grammar is represented by PVS description language. The second part of the paper deals with verification by PVS. It shows how some protocol properties can be specified using PVS language and verified by the prover. The last part of the paper summarizes current results and proposed future steps.

* Dept. of Computer Science, FEI VUT Brno, Božetěchova 2, 612 66 Brno

2 Formal description of a protocol

In [8] there are many formal methods that are used for formal description of protocols. For instance, state-transition based models (finite-state machine), graph models (Petri Nets, Data Flow Graphs), algebras (CCS, CSP), formal languages (according to Chomsky), or languages of specific systems - SDL, ESTELLE, LOTOS etc.

These tools and methods are not necessarily suitable for a class of high-level communication protocols. This class has some specific properties:

- Protocols use a half-duplex communication channel.
- Protocols are of type *request/response*.
- Communication is mostly implemented over connection-oriented services - TCP.
- A protocol consists of a small set of commands.
- Data exchange is usually limited by time constraints, eg. timeout, maximal delay etc.

Most of the above mentioned methods is not applicable to a full-version protocol with parameters, time constraints, synchronization in distributed environment etc. For formal specification and verification specification of protocols is reduced. However, using that simplification the protocols are examined only from structural point of view. One possible solution to this problem is to use a more powerful specification system in contrast to, for instance, regular grammars. In [5], there is proposed a new approach *based on conditional grammars.*

For our study, we can abstract from a full-version protocols. Firstly, a simple version of a protocol is supposed. For formal description, regular grammar can be used.

The grammar is formally characterized by a tuple

$$G = (N, V, P, S),$$

where N is *a final set of non-terminals*, V is *a final set of terminals*, P is *a set of rules* and S is *the starting nonterminal* ($S \in V$).

2.1 Case study

In our case study a simple protocol is considered. Following requests and answers are implemented: *add_user, remove_user, bye, user_added, user_removed, already_exists, no_such_user*. These commands are a part of the terminal set V. In our case only five nonterminals are needed, so $N = \{A, B, C\}$, the starting symbol S and the end state E.

Rewriting rules P of the grammar can be written as follows:

$$\begin{array}{ll} S \rightarrow & \text{hello } A \\ A \rightarrow & \text{add\_user } B \mid \text{remove\_user } C \mid \text{bye} \\ B \rightarrow & \text{user\_added } A \mid \text{invalid\_user } A \mid \text{already\_exists } A \\ C \rightarrow & \text{user\_removed } A \mid \text{no\_such\_user } A \end{array}$$

This communication protocol can be visually represented as a finite-state machine - see Figure 1. Firstly, a TCP/IP connection between two partners is established (note that the request for a new connection is initiated by a client). Then, the server sends *hello* to confirm that a new dialogue will start. When data exchange is finished command *bye* is sent by the client and the TCP/IP connection is closed.

A language $L = L(G)$ is a language of communication sequences of the system. Valid sentences of the language express possible communication sequences between a client and a server.

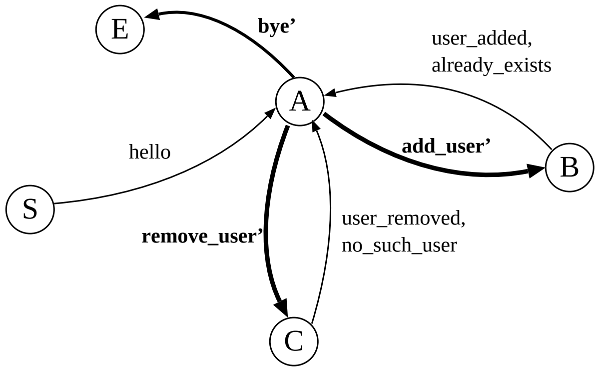

Fig. 1: FSM of the example.

For verification of the system PVS theorem prover can be exploited. At first, the system specification has to be expressed as a theory in PVS language. Then, properties of the system are to be written as formulas in PVS language. Using theorem prover and tools integrated in PVS it can be proved if specified property holds or not. A successfully proved theory is displayed in LaTeX format.

3 Prototype Verification System

PVS is a verification system with a general purpose specification language integrated with a theorem prover, model checker, and ancillary tools. Specification language is a higher-order logic with

- parameterized theories,
- computer-science type-constructors (records, lists etc.),
- predicate subtypes and dependent types,
- definitional forms include recursive functions, abstract data types, inductive definitions etc.

PVS theorem prover is based on Gentzen's formal axiomatic system, see further in [3]. Basic idea behind theorem proving is *to describe a system by high-level functional description and specify desirable properties as theorems to be proved.* These theorems are called *putative theorems.*

The point is that the process of proving putative theorems quickly highlights the gaps, errors, and inadequacies in the functional description. *In some cases, such a proof could also prove that the system as described meets its complete specification, in other words, that it is* correct. But it is rare that the proof succeeds. The typical proof attempt fails, but such failure usually yields valuable insights that can be used to correct oversights in the specification or formulation of the putative theorems.

The PVS proof checker is intended to serve as a skeptical party in a proving dialogue. The user supplies the steps in the argument and PVS applies them to the goal of the proof progressively breaking them into simpler subgoals or to obvious truths or falsehoods. If all the subgoals are reduced to obvious truths, the proof attempt has succeeded.

Several aspects of the language are built into the proof checker:

- the automatic use of the type constraints by the decision procedures,

– the simplifications given by the abstract datatype axioms,
– forms of beta-reduction, and
– extensionality.

The most direct consequence of this is that the trivial, obvious, or tedious parts of the proof are often entirely hidden from the displayed proof.

4 Case study

4.1 Protocol specification using PVS

The first part of the project is focused on PVS as a tool for protocol specification and testing. A specification of the protocol described in the previous section was implementented in PVS as a theory using PVS language [7]. The specification is based on the Finite State Machine theory (FSM) - see Figure 2.

```
state: TYPE = {A, B, C, S, E}          % states

input: TYPE = {hello,remove_user, add_user, bye, user_removed, no_such_user, user_added,
already_exists}

automaton(st:state, inp: input): state = COND
    st = S AND inp = hello → A,
    st = A AND inp = remove_user → C,
    st = A AND inp = add_user → B,
    st = A AND inp = bye → E,
    st = B AND inp = user_added → A,
    st = B AND inp = already_exists → A,
    st = C AND inp = user_removed → A,
    st = C AND inp = no_such_user → A
ENDCOND
```

Fig. 2: Protocol specification in PVS as a transition system

The specification of the protocol and auxiliary functions are written in PVS language. Then, it can be verified if an input sequence of commands belongs to language L(G), i.e. if it is a valid sentence of the protocol. In PVS a sequence of protocol commands becomes *a theorem* to be proved. If the proof is successful (the result is TRUE) the sequence is a valid protocol sentence according to the protocol specification. We implemented sequences of commands as lists of protocol requests and responses - see following example:

```
% two valid protocol sequences
adduser: THEOREM IsValidSentence?((: hello, add_user, user_added, bye :))
add2users: THEOREM IsValidSentence?((: hello, add_user, user_added, add_user, already_exists, bye :))

% two invalid protocol sequences
short : THEOREM IsValidSentence?((: hello, bye, hello, add_user :))
short2 : THEOREM IsValidSentence?((: hello :))
```

The specification of the protocol in PVS in Figure 2 looks nice but it does not make use of special PVS verification features (high-order logic, mu-calculus etc.). The same result can be reached by other tools, for instance, FSM implementation in a programming language. A positive thing is that an implementation in PVS is short and clear.

In next step, the implementation of the protocol was extended by adding protocol parameters. *The goal of our effort is to use PVS system for protocol verification AND simulation.* That means not only test if an input sequence is a valid protocol sentence but also simulate behaviour of the protocol communication from a view of one dialogue partner. For example, we want to simulate what is the answer of the server to a specific request (sequence of commands) of the client.

```
state: TYPE = {A, B1, B2, C1, C2, S, E, Error}        % states
command: TYPE = {hello,remove_user,add_user,bye, user_removed,no_such_user,user_added,
already_exists}                                        % requests and responces
State: TYPE = [# st: state, db: userdb #]              % State - record type

automaton(cstate:State, inp: command, args: t): State =
COND
    st(cstate) = S → IF inp = hello THEN cstate WITH [ st:=A ]
                     ELSE cstate WITH [ st:=Error ]
                     ENDIF,
    st(cstate) = A → IF inp = add_user THEN
                        IF IsIndb?(db(cstate),args) THEN cstate WITH [ st:=B1 ]
                        ELSE cstate WITH [ st:=B2, db:= AddUser(db(cstate),args)) ]
                        ENDIF
                     ELSIF inp = remove_user THEN
                        IF IsIndb?(db(cstate),args) THEN
                           cstate WITH [ st:=C1, db:= DelUser(db(cstate),args)) ]
                        ELSE cstate WITH [ st:=C2 ]
                        ENDIF
                     ELSIF inp = bye THEN cstate WITH [ st:=E ]
                     ELSE cstate WITH [ st:=Error ]
                     ENDIF,
    st(cstate) = B1 → IF inp = already_exists THEN cstate WITH [ st:=A ]
                     ELSE cstate WITH [ st:=Error ]
                     ENDIF,
    st(cstate) = B2 → ...
    st(cstate) = C1 → ...
    ...
ENDCOND
```

Fig. 3: Protocol specification in PVS with parameters

Now, the protocol specification becomes more complicated - see Figure 3. States B and C are split into two new states B_1, B_2 and C_1, C_2. New states and transitions are added and the non-deterministic FSM is tranformed to a deterministic FSM. Transition from A to B_1 depends not only on an input string but also on a condition which is specified as a predicate *IsIndb?*. If the predicate is true state B_1 is reached. Otherwise, state B_2 is reached. The new FSM is depicted on Figure 4.

Communication sequences of the system with parameters are specified in form of

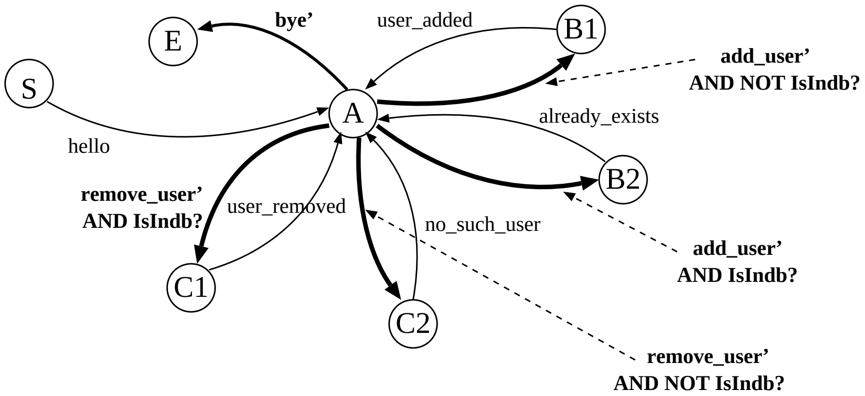

Fig. 4: FSM of the protocol with parameters.

theorems as follows. An input is a pair of a command and a parameter. In the following example symbol e means empty parameter.

```
% two valid protocol sequences
adduser: THEOREM LET db:userdb = null IN
   CheckSentence((db,(: (hello,e), (add_user,a), (user_added,e), (add_user,a), (already_exists,e),
(bye,e) :), null))
deluser: THEOREM LET db:userdb = null IN
   CheckSentence((db,(: (hello,e), (add_user,a), (user_added,e), (remove_user,a), (user_removed,e),
(bye,e) :), null))

% two invalid protocol sequences
adduser2: THEOREM LET db:userdb = null IN
   CheckSentence((db,(: (hello,e), (add_user,a), (add_user,a), (already_exists,e), (bye,e) :), null))
adduser3: THEOREM LET db:userdb = null IN
   CheckSentence((db,(: (hello,e), (add_user,a) :), null))
```

4.2 Protocol properties in PVS

Now we want to test properties of the specified protocol. What kind of behaviour is expected for communication protocols ?

- *Fairness*, i.e. any state of the system is reachable from the starting state S.
- *Correctness.* The correctness criterium depends on the type of a protocol. In our case we defined correctness as a theorem that *after adding a new user to database and removing him by next request the database has to have the same contents.*
- others ...

Process of proving of the properties is still in progress. We try to prove theorems that contain higher order logic (quantifiers, for instance) and temporal logic. We want to find a strategy for protocol proving using PVS. Our goal is to show how PVS system can be exploited for protocol proving and what type of properties, proving strategies can be succesffully used. This paper demonstrates partial results of our research.

5 Conclusion

This paper presents a first-look experience dealing with formal specification and verification of high-level protocols using PVS. It shows an example of steps for creating the verified version of the protocol. At first, the formal description using regular grammar (or FSM) is made. Then this description is translated to PVS specification language. When specified as PVS theory, verification process can be applied to verify some properties of the system. We demonstrated how the protocol description can be extended by protocol parameters. The paramaters help us to model real behaviour of the protocol.

The paper shows an example of a simple protocol and its description by PVS language. Some properties of the protocol are specified as theorems in PVS language. Using PVS system their validity has been proved.

Future steps go further in protocol specification. High order logic and temporal logic are used to specify general and temporal propeties of the protocol. Their proving is not simple. Our goal is to show how PVS system can be exploited for protocol proving, and to propose some strategies for protocol proving and specification of the properties.

Acknowledgements

The work introduced in the paper is a part of a project supported by grant FRVŠ no. 99/2001 *"Methods selection for formal specification and verification of communication protocols"* of Czech Ministry for Education.

Bibliography

1. Crow, J., Owre, S., Shankar, N., Srivas, M.: *A Tutorial Introduction to PVS*, SRI Int., 1995. Available at URL http://www.csl.sri.com/sri-csl-fm.html.
2. Cerny, Eduard, Song, Xiaoyu: *Formal Verification Methods*, Notes of the Lecture, 1996.
3. Janák, V.: *Základy formální logiky*, SPN Praha, 1976.
4. 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.
5. Matoušek, P.: *Real-world protocols specified by Conditional Grammars*, to be published in Proceedings of ESM 2001.
6. Murphy, S.L., Shankar, A.U.: *Service Specification and Protocol Construction for the Transport Layer*, ACM Communication architectures and protocols, 1988, Pages 88-97.
7. Owre, S., Shankar, N., Stringer-Calvert, D.W.: *PVS Language Reference*, Version 2.3, SRI Int., September 1999. Document available on URL http://pvs.cls.sri.com.
8. Tarnay, K.: *Protocol Specification and Testing*, Akademia Kiado es Nyomda Vallalat, Budapest, 1991.