

---

# Maple-PVS User Guide

Version 0.2 ■ December 2005

---

Clare M. So and Hanne Gottliebsen

Department of Computer Science  
Queen Mary, University of London

{cmso,hago}@dcs.qmul.ac.uk

## **Abstract**

Maple is a popular, powerful computer algebra system, but it could not guarantee the correctness of all problems. PVS can provide formal proofs to guarantee the correctness of problems, but it is difficult to use without specific knowledge of the system. To complement this weakness of PVS and Maple, an interface is built between the systems so that a Maple user can access some functionalities of PVS. This document describes the functionalities of this interface. Some non-trivial theorems are used to further demonstrate why this interface is useful.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>A Tour of the Maple-PVS Interface</b>	<b>2</b>
2.1	Before You Start . . . . .	2
2.2	Initializing the Interface . . . . .	2
2.3	Some Simple Proofs . . . . .	4
2.4	Using Libraries in Proofs . . . . .	5
2.5	Finishing Off . . . . .	6
<b>3</b>	<b>Applications to Computer Algebra</b>	<b>7</b>
3.1	A Non-Trivial Proof . . . . .	7
3.2	Example: Continuity Checking . . . . .	8
<b>4</b>	<b>Frequently Asked Questions</b>	<b>10</b>

# Chapter 1

## Introduction

Maple is a popular, powerful computer algebra system for solving mathematical problems, but it lacks of formal methods to ensure the correctness of answers. PVS is a verification system that use formal methods to ensure the correctness of theories. To complement this weakness of Maple, we enable a Maple user to access certain functionalities by our Maple-PVS interface. In this interface, a Maple user can access PVS in a Maple session.

This document is intended to be a tutorial on how this interface is used. For instruction on installing and configuring the software, please see the *Installation Guide*. For information on this software's architecture, please see the *Documentation*. This guide is not intended to serve as a PVS language tutorial nor a Maple programming language manual. The audience of this document is assumed to be familiar with the PVS language syntax and the Maple programming language.

# Chapter 2

## A Tour of the Maple-PVS Interface

In this chapter, we start by a brief outline of the Maple-PVS interface's requirements. This includes a list the software to be installed. Then we start to illustrate the functionalities of this interface by some simple examples. Finally, we show how libraries can be used in our proofs. Having access to the libraries is useful in proving non-trivial theorems.

### 2.1 Before You Start

Please be sure you have all of the following software installed and configured before continuing this tutorial:

- Linux (Example: Debian, Fedora Core) or Solaris
- PVS Specification and Verification System 3.2 (<http://pvs.csl.sri.com/>)
- Maple 9.5 or 10 (<http://www.maplesoft.com/>)

A complete list of the software requirement is listed in the *Installation Guide*.

The users of this interface is assumed to be familiar with the PVS syntax and the Maple programming language. A regular PVS-Emacs user should have enough background in PVS syntax. Elementary knowledge of Maple programming is adequate.

Either command-line or GUI version of Maple can be used: The Maple commands are the same under both versions. We use the command-line version of Maple throughout this document. A Maple worksheet containing the examples be found along with the Maple-PVS documentation.

### 2.2 Initializing the Interface

The first task to use the interface are to start Maple. Depending your machine's configuration, Maple may be started by typing `maple` in the shell's command prompt. An ASCII art of a Maple leaf, version and copyright information is displayed after Maple is started. Note that the prompt in Maple is denoted by `>`.

```

      |\~/|      Maple 9.5 (IBM INTEL LINUX)
._|\|\  |/\|_  Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /   All rights reserved. Maple is a trademark of
 <_---- _----> Waterloo Maple Inc.
      |      Type ? for help.
>

```

One part of the wrapper interface is implemented as a Maple package. This package serves as a part of the wrapper for PVS. Reading and loading the package is accomplished by “read” and “with” commands. Here we assume the package (named `pvsmodule.mpl`) is stored in the `../src/` directory.

```

> restart:
> read("../src/pvsmodule.mpl"):
> with(PVS):

```

We initialize the interface by the `PvsStart` command.

```

> pvs := PvsStart():

```

This command executes PVS, creates a communication channel between Maple and PVS and opens a Tcl/Tk window (Figure 2.1). The session identifier is returned by `PvsStart`. In our example, this identifier is held by the variable `pvs`. This identifier is absolutely necessary to execute the prove commands.

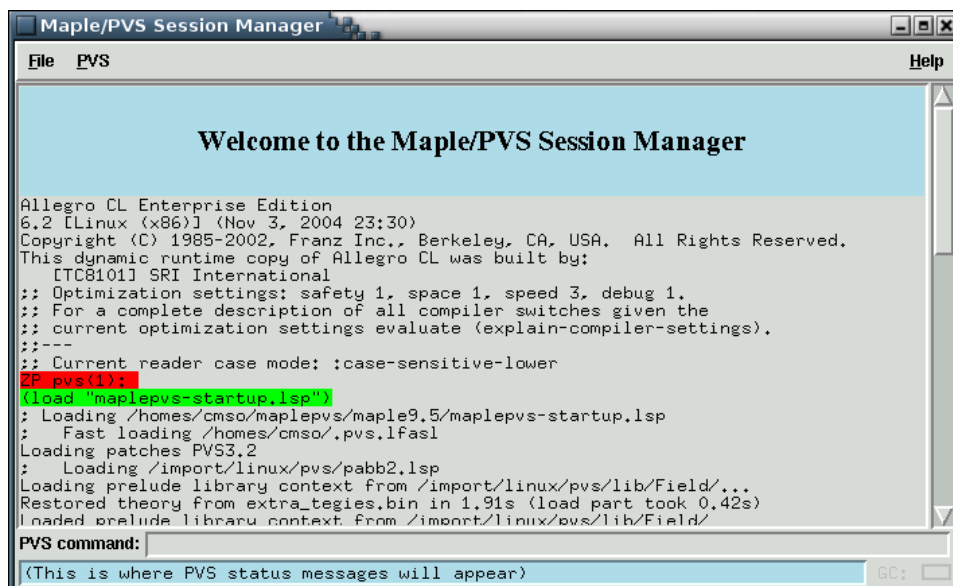


Figure 2.1: Tcl/Tk window for the Maple-PVS interface.

## 2.3 Some Simple Proofs

We are now ready to use PVS from this Maple session. The simplest command is `PvsProveTrivial`. It tries to prove theorems with the standard PVS Prelude library. It takes (1) a PVS session identifier, (2) a formula in PVS and (3) a strategy. If the strategy is an empty string, the default `ASSERT` strategy is used. This command returns a Maple table containing the proof of the query. Let's try to prove boolean formula  $((p \Rightarrow q) \text{ AND } p) \Rightarrow q$  with the `(grind)` strategy.

```
> ex1 := PvsProveTrivial(pvs, "f: FORMULA FORALL (p,q:bool): \
                               ((p => q) AND p) => q","(grind)");
```

Note that `PvsProveTrivial` and other prove commands do not print the proof or the result by default. We can use `PvsQEDfind` to find out if the theorem is proved true by PVS or not. It takes the Maple table returned by the `PvsProveTrivial`. In this case, `ex1` has already stored the proof of our query.

```
> PvsQEDfind(ex1);

                               true
```

Optionally, you can ask Maple-PVS to print out the proof by issuing the `PvsPrintLines` command. This command also takes the Maple table returned by `PvsProveTrivial`.

```
> PvsPrintLines(ex1):
-----
| f :
|   |-----
| {1}  FORALL (p, q: bool): ((p => q) AND p) => q
| Rerunning step: (grind)
| Trying repeated skolemization, instantiation, and if-lifting,
| Q.E.D.
| Run time  = 0.08 secs.
| Real time = 0.20 secs.
| :proved
-----
```

The proof does not only give us information on the steps taken, it also shows the proof's run time and status. The status, which is in the last line of the proof, can be one of the following: `proved`, `disproved`, `unknown`, `unproved`.

Let's try to prove some formula that is obviously wrong. In this example, since the last argument is an empty string, the default `ASSERT` strategy is used.

```
> ex2 := PvsProveTrivial(pvs,"f: FORMULA 1 + 1 = 3","");
```

We can ask Maple-PVS if the formula is proven or not by `PvsQEDfind`. If we are curious of the steps, we can ask Maple-PVS to print out the proof by `PvsPrintLines`.

```
> PvsQEDfind(ex2);

false
```

```
> PvsPrintLines(ex2);
```

```
-----
| f :
| |-----
| {1}  1 + 1 = 3
| Rerunning step: nil
| No change on: (skip)
| f :
| |-----
| [1]  1 + 1 = 3
| Postponing f.
| f :
| |-----
| {1}  1 + 1 = 3
| Run time = 0.01 secs.
| Real time = 0.01 secs.
| :unknown
-----
```

## 2.4 Using Libraries in Proofs

PVS Libraries can be used for our proofs. The `reals` library is used in the examples of this section.

Let's try to prove our lemma `LEMMA sqrt(4) = 2` using the some strategies from the `reals` library. We can use the `PvsProve` command to direct the proving process. This command takes (1) the identifier, (2) the formula in PVS syntax, (3) the `.pvs` file from which the strategies are imported and (4) the strategy.

```
> ex3 := PvsProve(pvs,"f: LEMMA sqrt(4) = 2", "reals@sqrt", \
                "(use \"sqrt_lem\") (assert)")
```

In argument (2), the string before “@” is the name of subdirectory under the default library path. The string after “@” is the name of `.pvs` file. More than one `.pvs` files can be specified: Each filename is to be separated by commas. When specifying libraries' special strategies in argument (3), don't forget to put a backslash (“\”) before the quotation marks.

As mentioned in section 2.3, the result of the proof is not displayed by default. We can access the result of this proof by `PvsQEDfind` and `PvsPrintLines`:

```
> PvsQEDfind(ex3);

true
```

```

> PvsPrintLines(ex3);
-----
| f :
|   |-----
| {1}  sqrt(4) = 2
| Rerunning step: (use "sqrt_lem")
| Using lemma sqrt_lem,
| this simplifies to:
| f :
| {-1} sqrt(4) = 2 IFF 2 * 2 = 4
|   |-----
| [1]  sqrt(4) = 2
| Rerunning step: (assert)
| sqrt_4 rewrites sqrt(4)
|   to 2
| Simplifying, rewriting, and recording with decision procedures,
| Q.E.D.
| Run time  = 0.33 secs.
| Real time = 0.50 secs.
| :proved
-----

```

## 2.5 Finishing Off

When you are finished using the Maple-PVS interface, shut down the interface gracefully by `PvsEnd`. This command takes the session identifier that we have gotten from `PvsStart`. In our case, the variable `pvs` holds this session's identifier.

```
> PvsEnd(pvs) :
```

After `PvsEnd` is entered, PVS exits and the Tcl/Tk window closes. There is no more communication between Maple and PVS. If you wish to use PVS from Maple, you have to start the interface again by `PvsStart`.

# Chapter 3

## Applications to Computer Algebra

This chapter explains how the Maple-PVS interface can help to solve non-trivial problems. To demonstrate the power of the interface, we start by giving an example showing the level of complexity this interface handles. This example exposes a weakness of this interface: Leaving the user to specify complex proof steps decreases the usability. To increase the usability, sets of strategies can be predefined for a specific type of mathematical problem. Since we are interested in using PVS to check the continuity of functions, we have derived a set of special PVS strategies for this purpose. (maple proc!)

The readers of this chapter are assumed to be familiar with initializing and closing the interface, using commands, and proving simple lemma.

### 3.1 A Non-Trivial Proof

Having the users to specify the strategies can be advantageous: Expert PVS users can freely choose their preferred strategies. This feature can also decrease the usability of the interface: Specifying every step of the proof could be cumbersome for non-expert PVS users. To illustrate this point, let's use the `reals` library to prove that

$$|\sqrt{x} + \sqrt{x_0}| |\sqrt{x} - \sqrt{x_0}| < \sqrt{x_0} \delta \Rightarrow |\sqrt{x} - \sqrt{x_0}| < \epsilon$$

The proof consist of several PVS prove commands. These commands may not be trivial for most PVS users.

```
> ex4 := PvsProve(pvs, "f: LEMMA FORALL (eps,x,y: posreal): \
    abs(sqrt(x)-sqrt(y))*abs(sqrt(x)+sqrt(y)) < eps*sqrt(y) \
    IMPLIES abs(sqrt(x)-sqrt(y)) < eps", \
    "reals@top", \
    "(skosimp)(lemma \"both_sides_times_pos_lt1\" (\x\" \
    \"abs(sqrt(x!1)-sqrt(y!1))\" \"pz\" \
    \"abs(sqrt(x!1)+sqrt(y!1))\" \"y\" \"eps!1\"))(grind)");
```

This example does not imply that Maple cannot prove this theorem. Instead, it shows us how this interface may not be useful for complex problems: Most users don't know how to specify complex PVS strategies.

## 3.2 Example: Continuity Checking

We discovered that certain type of problems can be solved by the same strategies. This means we can predefine sets of strategies for specific types of problems.

A PVS library for continuity checking had been developed by Hanne Gottliebse. The motivation of building this PVS library was that Maple's continuity checking function is not reliable. Consider the example  $f(x) = \frac{1}{2+\cos(x)}$  is continuous everywhere since  $2 + \cos(x) \neq 0$  for all  $x \in \mathbf{R}$ . Maple 9.5's `iscont` returns `false`:

```
> iscont(1/(2+cos(x)), x=-infinity..infinity);
      false
```

Continuity checking has useful applications in other areas. One of the application is to check the results given by Maple's `dsolve` (find exact solutions for systems of ordinary differential equations). This Maple function relies on the fact that the input functions are continuous over a certain interval. It does not guarantee the validity of the solution: All valid solution should be continuous over the interval.

This library was initially developed using a previous version of Maple. Adjustments are to be made so that it is compatible to PVS 3.2.

### An Application to IVPs

We here present how the improved `dsolve` can be used for checking the input and output of IVP. Before presenting the pseudocode, we first define the definition of the problem. Consider a function in the form

$$y' = r(x) - q(x)y(x), y(a) = eta, x \in [a, b]$$

In order for the solution to be valid, the following conditions must be true for the inputs  $r(x)$  and  $q(x)$ :

- $r(x)$  and  $q(x)$  are continuous over  $[a, b]$
- $r(x) - q(x)y(x)$  is continuous over  $[a, b]$

The final solution,  $y$ , should satisfy the following properties:

- $y' = f(x, y)$
- $y(a) = eta$
- $y(x)$  is continuous over  $[a, b]$

PVS can help Maple to check the conditions of the inputs and the solution. The first part of the pseudocode checks the conditions of the inputs. Then we ask Maple's `dsolve` to solve this problem. Finally, the solution is checked for each of the three properties. Note that checking properties of inputs and solution should be done by PVS instead of Maple. "`top_analysis`" is the library for continuity checking. "`cts`" is one of the PVS strategies defined by this library. "`grind`" is one of the PVS built-in strategies.

```

Input:  r(x) and q(x) -- Two functions
        a, b          -- Endpoints of the interval
        eta          -- A point between a and b

Output: sol          -- Valid solution

pvs := PvsStart

cond_1 := PvsProve(pvs, "g: FORMULA FORALL (v:I[a,b]):
                        continuous(lambda(x:I[a,b]): r(x), v)",
                        "top_analysis", "(cts)")

cond_2 := PvsProve(pvs, "g: FORMULA FORALL (v:I[a,b]):
                        continuous(lambda(x:I[a,b]): q(x), v)",
                        "top_analysis", "(cts)")

if not (PvsQEDfind(cond_1) and PvsQEDfind(cond_2))
then
  Error "Either r(x) or q(x) is not continuous over [a,b]"

else
  sol := dsolve({diff(y(x),x) = r(x) - q(x)y(x), y(a) = eta}, y(x))
  diffsol := diff(sol,x)

  cond_3 := PvsProve(pvs, "f: FORMULA FORALL (v:I[a,b]):
                          diffsol(v) = r(v)-q(v)*sol(v)",
                          "top_analysis", "(grind)")

  cond_4 := PvsProve(pvs, "f: FORMULA sol(a) = eta", "top_analysis", "(grind)")

  cond_5 := PvsProve(pvs, "f: FORMULA FORALL (v:I[a,b]):
                          continuous(lambda(x:I[a,b]): sol(x), v)",
                          "top_analysis", "(cts)")

end if

if not (PvsQEDfind(cond_3) and PvsQEDfind(cond_4) and PvsQEDfind(cond_5))
then
  ERROR "Invalid solution"
else
  return sol
end if

```

# Chapter 4

## Frequently Asked Questions

1. PVS is behaving strangely. What can I do?  
Never execute more than one PVS session in your computer, even if the sessions are from PVS-Emacs, PVS raw or different Maple worksheets. Doing so may cause unpredictable results and loss of proof information.
2. I got a repeated “`excp() failed`” message after I typed `PvsStart`. What does this mean?  
This means PVS is not started successfully. A possibility is that Maple-PVS cannot find the local installation of PVS. Please refer to the *Installation Guide* for details.
3. Maple is frozen after the Maple-PVS interface is initialized.  
Please see answer of (2).
4. There is no Tcl/Tk window!  
Please check the access permission of `pvs-ctl` found under `src/`. Please refer to the *Installation Guide* for details on setting access permission of files.

# Bibliography

- [1] PVS Verification and Specification System. <http://pvs.csl.sri.com>
- [2] Maple Computer Algebra System. <http://www.maplesoft.com>
- [3] Monagan, Michael B., et al. *Maple Advanced Programming Guide*. Toronto: Maplesoft, a division of Waterloo Maple Inc., 2005.
- [4] Monagan, Michael B., et al. *Maple Introductory Programming Guide*. Toronto: Maplesoft, a division of Waterloo Maple Inc., 2005.
- [5] Andrew Adams, Martin Dunstan, Hanne Gottliesen, Tom Kelsey, Ursula Martin, Sam Owre. *Computer Algebra meets Automated Theorem Proving: Integrating Maple and PVS*. TPHOLS 2001. (?)