Computer Algebra meets Automated Theorem Proving: Integrating Maple and PVS

Andrew Adams[1], Martin Dunstan[2], Hanne Gottliebsen[3], Tom Kelsey[3], Ursula Martin[3], and Sam Owre[4]

[1] School of CS, Cyb and EE, University of Reading, Reading RG6 6AY, UK
[2] NAG LTD, Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK
[3] School of Computer Science, University of St Andrews, St Andrews KY16 9SS, UK
[4] Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park CA 94025, USA

Abstract. We describe an interface between version 6 of the Maple computer algebra system with the PVS automated theorem prover. The interface is designed to allow Maple users access to the robust and checkable proof environment of PVS. We also extend this environment by the provision of a library of proof strategies for use in real analysis. We demonstrate examples using the interface and the real analysis library. These examples provide proofs which are both illustrative and applicable to genuine symbolic computation problems.

1 Introduction

In this paper we describe an interface between version 6 of the Maple computer algebra system, henceforth CAS, and the PVS automated theorem prover, ATP. CAS like Maple (which has 1 million users, mostly in the educational sector) incorporate a wide variety of symbolic techniques, for example for factoring polynomials or computing Gröbner bases, and increasingly some numerical elements also. They provide a mathematical programming environment and facilities such as graphics and document generation. They have enjoyed some outstanding successes: for example 't Hooft and Veltman received the 1999 Nobel Prize in Physics, Veltman for using computer algebra to verify 't Hooft's results on quantum field theory. The advantages of combining symbolic/numeric computation with logical reasoning are evident: improved inference capability and increased expressivity. A human performing symbolic mathematics by hand will make automatic use of the logical framework upon which the mathematics is based. A human performing similar calculations using a CAS has to supply the same logical framework; there is, in general, no sound reasoning system built in to the CAS. Our objective is to provide the CAS user with an interface to an ATP, so that certain side conditions which are implicit in many analytic symbolic computations can be highlighted, checked, and either verified or flagged as an error. The user interacts with the CAS, making calls to the ATP system which acts as a black box for the provision of formal proofs.

While there have been many projects aimed at combining the calculation power of CAS with the logical power of ATP systems, the basic approach is always from the

point of view of a user interested in completely correct developments. That is, the point of view of a theorem proving user who wishes to have powerful calculation available to generate existential witnesses or factorisations. Our approach takes the view point of a user of a CAS who simply wishes it to be more robust. Current users of CAS, while not pleased when incorrect results are generated, are nevertheless willing to accept a certain level of deficiency as a trade-off for the utility of the system. We aim to improve the existing support for users of CAS by adding to the correctness in an incremental way, with a black box ATP system supporting better computer algebra calculation. The work described in this paper is another step toward this goal, though much remains to be done, both in terms of developing the formalisations needed in ATP systems and in re-engineering the algorithms implemented in CAS to take advantage of the black box without losing the speed and power of existing CAS implementations.

The aims of this paper are (i) to describe the design and implementation of the interface (Section 3), and (ii) to illustrate its power by describing the automated verification of properties needed to ensure correctness of routines for differential equations in Maple (Section 4). The remainder of this introduction provides motivating examples, and briefly describe the systems involved and the background to the project. In Section 2 we discuss the extension to PVS which allows automated proofs of side conditions, continuity and convergence for elementary functions with parameters. One use of the interface is the checking of the validity of input to certain Maple procedures. For example, when solving a differential equation, one must ensure that the input function $f(x, y)$ is continuous (and thus defined) on the required region. This is not done by Maple, and so Maple might return an answer that looks like a plausible solution even if the input is not of the right form. However, using the interface one can verify the continuity of $f(x, y)$. Although our current PVS implementation of continuity checking relies on a fairly simple method for verifying continuity, it does handle a large class of functions well.

1.1 Motivating Examples

In this section we show that CAS output can be misleading or incorrect. A typical CAS will have no general mechanism for checking that results are mathematically correct. Hence errors can arise in the determination of equality of two expressions, or in the definedness of expressions with respect to symbolic parameters. By not formally checking each result, CAS output can propagate errors throughout a complicated symbolic calculation, and hence produce invalid results. The following examples illustrate typical problems which can occur.

Simplification The expression $\sqrt{x^2}$ will not automatically simplify to x in a CAS, since this is incorrect for negative x. Maple provides a mechanism for assuming that variables (and functions) have properties such as negativity (or differentiability), but the system can still fail to simplify expressions containing square roots correctly. To see this, consider the expressions $a = |x - y|$ and $b = |\sqrt{x} + \sqrt{y}||\sqrt{x} - \sqrt{y}|$, where x and y are positive reals. Suppose a Maple user wishes to verify that $a = b$. The commands available are

```
> a := abs(x-y);
```

$$a := |x - y|$$

```
> b := abs(sqrt(x)+sqrt(y))*abs(sqrt(x)-sqrt(y));
```

$$b := \left|\sqrt{x} + \sqrt{y}\right| \left|\sqrt{x} - \sqrt{y}\right|$$

```
> assume(x > 0): assume(y > 0):
> verify(a,b);
```

$$false$$

```
> verify(a,b,equal);
```

$$FAIL$$

Note that issuing a command `assume (y > 0)` binds all future occurrences in this Maple session of the parameter `y` to be positive unless a new assumption about `y` is issued. However, not all algorithms will check the assumption database for information about the parameters included in their arguments. It is evident that there is a problem with the robustness of the Maple `verify` command: `verify(a,b)` returns `false` for this example; the heuristic simplification techniques used to reduce $a - b$ to zero are inadequate. `verify(a,b,equal)` uses a more sophisticated set of heuristic techniques, but still fails to provide the expected result, namely `true`. Hence, for this example, the output from Maple is either an admission of failure or misleading to the user.

Definedness and Parameters Consider the definite integral

$$\int_0^b \frac{1}{x-a} dx \tag{1}$$

where a and b are positive real parameters. Maple returns the solution $\log(b-a) - \log(a) - i\pi$, which, when $a = 1$ and $b = 2$, reduces to $-i\pi$ which is a complex number, and hence incorrect. Maple does not check that the function is defined everywhere in the interval of integration; in this case that a is not in $(0, b)$.

Continuity Checking Maple has an inbuilt procedure for checking properties of functions and expressions. This procedure does not always return correct results:

```
> is(x -> (1/x)+1-(1/x),continuous);
```

$$false$$

Even the assumption of positive arguments does not solve the problem:

```
> assume(y > 0):
> is(y -> (1/y)+1-(1/y),continuous);
```

$$false$$

The Maple/PVS interface described in this paper is designed to address the problem of the lack of verification of CAS results. The Maple user works in the usual Maple environment, and has access to a formal, checkable proof mechanism which has been extended to deal with properties of real valued expressions involving parameters, such

as those given in the above examples. In Section 4.1 we use the Maple/PVS interface to prove that

$$|x-y| = |\sqrt{x}+\sqrt{y}||\sqrt{x}-\sqrt{y}| \quad \text{for} \quad x,y>0 \tag{2}$$

and that

$$b > a > 0 \implies \exists x \in (0,b) \quad \text{such that} \quad x \notin \mathbb{R} \setminus \{a\}\,. \tag{3}$$

These proofs allow the user to correct, modify or disregard the incorrect results from Maple.

1.2 Background

Previous approaches to the combination of CAS and ATP are discussed and classified in [8]. We mention selected examples which are direct or indirect predecessors to this work. All approaches share the common problem of communicating mathematical information between the CAS and ATP systems. One solution is to define a standard for mathematical information that can, in principle, be understood by any mathematical computation system. Examples of this common knowledge approach include the OpenMath project [1, 11], and protocols for the exchange of information between generic CAS and ATP systems [5, 15]. Another solution is the sub-package approach, in which communication issues are side-stepped by building an ATP within a CAS. Examples include Analytica [4], REDLOG [12], the Theorema project [6, 7], and a logical extension to the type system of the AXIOM CAS [19, 26]. The third common approach involves the choice of preferred CAS and ATP environments, and the construction of a specific interface for communication between them. Examples include Maple-HOL [17], Maple-Isabelle [3], and Weyl-NuPrl [18].

The common-knowledge approach depends on wholesale acceptance of the protocols by both the CAS and ATP communities. Since this has not yet happened, we concentrate on the remaining two approaches. The sub-package approach has the advantage that communication is easy, and the disadvantage that implementation of an ATP in a language designed for symbolic/numeric computation can be hard. In particular, there may be problems with misleading or incorrect output from CAS routines, which adversely affect the soundness of the proof system. For example, the simplification error described in Section 1.1 could lead to an undetected division by zero, which could propagate logical errors.

The specific interface approach has the advantage that the target CAS and ATP are implemented and developed by experts in the respective technologies, and the disadvantage that communication problems remain. Another issue is the relationship between the systems. In the examples mentioned above, the primary system in use was the ATP, and the CAS was used as an oracle to provide calculation steps. By contrast the motivation for our work is to support the users of computer algebra systems in their work by giving them the opportunity to use the rigour of theorem prover when they wish, completely automatically in some cases. This has the advantage that users can use all the facilities of the CAS, but the theorem prover implementation can be restricted. Since Maple is a programming language calls to a prover can be embedded in procedures and made invisible to the user.

The objective of this project is to develop a specific interface which links the major commercial CAS Maple [20] to the widely used ATP PVS [22] in a way which minimises the inherent problems of linking two unrelated systems. We also utilise recent extensions to PVS [14, 2] which allow reasoning in the theory of real analysis, so that properties of symbolic expressions can be described and checked. The project is in approach similar to the PROSPER toolkit [10], which allows systems designers using CAD and CASE tools access to mechanised verification. The PROSPER paradigm involves the CAD/CASE system as the master, with a slave proof engine running in the background. We have also adopted the PROSPER architectural format: a core proof engine which is integrated with a non-formal system via a script interface. Our target, however, is the engineering/scientific/mathematical community of CAS users.

1.3 Maple

Maple [9] is a commercial CAS, consisting of a kernel library of numeric, symbolic and graphics routines, together with packages aimed at specific areas such as linear algebra and differential equations.

The key feature of version 6 of Maple is that it was designed to run a subset of the NAG numerics library. We utilise this ability to extend Maple by running PVS as a subprocess.

1.4 PVS

PVS [24] supports formal specification and verification and consists of a specification language, various predefined theories and a theorem prover which supports a high level of automation. The specification language is based on classical, typed higher-order logic and supports predicate and dependent sub-typing.

PVS is normally run using Gnu or X emacs for its interface, but can also be run in batch mode or from a terminal interface. However, running PVS via emacs provides many useful features such as abbreviations for prover commands and graphical representations of proofs and library structures. The core of PVS is implemented in Allegro Common Lisp.

1.5 Design Issues

The overall view of our system is one in which the user interacts with the CAS, posing questions and performing computations. Some of these computations may require ATP technology, either to obtain a solution or to validate answers obtained by existing computer algebra algorithms. In such cases the CAS will present the theorem prover with a number of lemmas and request it to make some proof attempts. The results of these attempts, whether successful or otherwise, will guide the rest of the computation within the CAS: the theorem prover acts as a slave to the CAS.

The system has a tightly coupled architecture. In such a system, the theorem prover shares the same resources as the computer algebra system and is directly controlled by it.

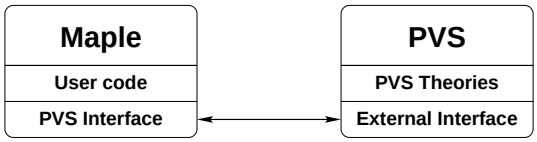

Fig. 1. Tightly Coupled System

2 The PVS Real Analysis Library

Our goal is to provide automatic support for reasoning about real valued CAS expressions. The basic PVS libraries are insufficient for proving properties such as continuity or convergence of functions. It was therefore necessary to extend PVS with (i) libraries of real analysis definitions and lemmas, and (ii) specialist proof strategies.

A description of a PVS library of transcendental functions was provided in [14]. The library can be used to check continuity of functions such as

$$e^{x^2+|1-x|}\,. \tag{4}$$

The library is based on Dutertre's real analysis library [13] and contains a development of power series, which is used to describe and reason about functions such as exp, cos, sin and tan and their inverses. The library also contains a large set of lemmas about properties of these functions as provided by mathematics textbooks.

A particular focus of this library is supporting automation. There are two ways of automating proofs in theorem provers: writing purpose-built strategies; or using predefined widely applicable strategies. The latter method (such as `blast-tac` in Isabelle [25] or `grind` in PVS) is the one we have primarily used, although we have also written some special purpose strategies. Application of the PVS generic tactic `grind` can be quite difficult. In order to improve the performance of `grind` (both in terms of speed of proof and in the number of conjectures it will prove automatically) we have introduced various type judgements [23] into our development. These judgements allow grind to apply generic theorems to our specific tasks, effectively by giving hints to the matching algorithm searching for appropriate theorems in the development.

The method used for *continuity checking* is what one might call the High School method. It is based on the theorems that the constant functions and the identity function are continuous everywhere, and that well-founded combinations using the following operators are also continuous: addition, subtraction, multiplication, division, absolute value and function composition. Also, the functions exp, cos, sin and tan are continuous everywhere in their domains,[1] which means that we can prove that functions such as (4) are continuous on the whole of their domain. The continuity checker is invoked by using the strategy `cts`, which performs all the necessary theory instantiations.

Since [14] some changes have been made to the basic analysis library as some definitions were discovered to be unsuitable. This particularly concerns the definition

[1] Note that the domain of tan excludes all points with value of the form $(2n+1)\,\pi/2$ and that it is continuous everywhere else.

of convergence of functions. Dutertre's definition of convergence was unusual in that it coincided with the definition of continuity. This is seen by the "theorem" in Dutetre's development:

```
continuity_def2 : THEOREM
   continuous(f, x0) IFF convergent(f, x0) .
```

We changed the definition of convergence of functions to the more usual one, so that the above is no longer a theorem, and also made other necessary changes to the rest of the theory.

As well as changes to the underlying theories we have extended the implementation of the continuity checker. In particular, it is now possible to check functions such as $1/(cos(x) + 2)$ for continuity. This has been implemented using judgements to assert that cos and sin are always within $[-1; 1]$ and that adding (or subtracting) something strictly greater that 1 will return a non-zero real value.

As well as a continuity checker we now have a convergence checker; this will check if a certain function has a limit at some point (or indeed everywhere in its domain). We can prove, for example, that the function

$$-\pi - 1 + \pi * e^{1-cos(x)} \tag{5}$$

has a limit at the point

$$arccos(1 - log(\frac{1+\pi}{\pi})) \; . \tag{6}$$

The convergence checker is implemented in the strategy `conv-check`, and it works in the same syntax directed way as the continuity checker, and so has similar capabilities and limitations.

3 Implementation

In this section we describe the work undertaken to develop our Maple/PVS system. We have created a tightly coupled system under UNIX with PVS being controlled by Maple as if it was a "normal" user. That is, PVS believes that it is interacting with someone entering commands from a terminal rather than, for example, a pipe to another program.

3.1 Extending Maple

Although Maple provides its own programming language, it was necessary to make use of the Maple interface to external functions written in C. By writing a small amount of "glue" in C to handle the creation and management of new processes, low-level PVS interactions and other support facilities, we are able to present a simplified PVS interface to users of the Maple language.

An example of the Maple code that we use to import C functions is:

```
PvsSend:= define_external(
   'pvsSend',
      handle :: integer[4],
      message :: string[],
      RETURN :: boolean[4],
      LIB = "maple_pvslib.so"
);
```

Here the C function `long pvsSend(long handle, char *message)` is imported into Maple as an external procedure called `PvsSend`. The compiled C for `pvsSend` is stored in the shared object called `maple_pvslib.so`.

Only seven of our C functions are imported into Maple in this way and provide a basic low-level interface to PVS. Built on top of these are a number of higher-level functions written in the Maple language. For example, the Maple function used to start a PVS session with a particular context is:

```
PvsStart := proc(dir::string)
   # Start PVS using imported C function
   pvs := PvsBegin();

   # Send data to PVS via imported C function
   PvsSend("(change-context \"" || dir || "\")");

   # Wait for prompt using another imported C function
   PvsWaitForPrompt(pvs);

   pvs;
end proc
```

These Maple procedures are supplied as a module, which allows users to import them as a library package. The library must be added to Maple by each user before the interface can be used. The library can then be accessed at each session by the command:

```
> with(PVS);
```

$[PvsLineFind,\ PvsPrintLines,\ PvsProve,\ PvsQEDfind,\ PvsReadLines,$
$PvsSendAndWait,\ PvsStart]$

which loads the listed commands from the Maple module PVS.

3.2 Maple/PVS Communication

Maple/PVS communication has been implemented in two parts. Firstly, a simple lexical analyser (**pvs-filter**) recognises PVS output and translates it into a format that is easier to parse by other tools (such as the Maple interface). Secondly, a small Tcl/Tk application (**pvs-ctl**) acts as a broker between Maple, PVS and the Maple/PVS user. This broker application is launched from Maple as the child process instead of the PVS-Allegro LISP image and uses **pvs-filter** to simplify the processing of PVS-Allegro output:

Under normal use, Maple sends PVS commands to **pvs-ctl** which passes them directly to PVS-Allegro. Responses are translated by **pvs-filter** and examined by **pvs-ctl**.

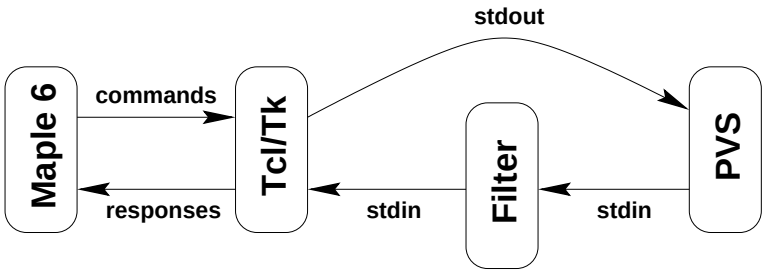

Fig. 2. Maple/PVS System

Anything that needs the user's attention is handled interactively by **pvs-ctl** allowing the user to respond directly to PVS-Allegro without Maple being aware of it. Status messages from PVS-Allegro are displayed graphically by **pvs-ctl** along with other output describing how proof attempts are progressing. Again, none of this reaches Maple—only the information that Maple actually needs. At present, Maple only needs to know about PVS prompts (so that it can send the next command), and Q.E.D. messages indicating that a proof attempt was successful.

The benefits of this system are significant: for the Maple programmer PVS appears as a black-box which is controlled via a trivial protocol. Maple sends a command to the black-box and reads response lines until a prompt is found. If any of these lines contains a Q.E.D. line then the command was successful, otherwise it was not. Simplicity is important because Maple is designed for solving computer algebra problems, and not for text processing.

For the Maple/PVS user the **pvs-ctl** application provides a graphical display of the current state of the system. Not only can progress be monitored, but interaction with PVS-Allegro is available when needed. If PVS stops in the middle of a proof attempt and asks for a new rule to apply, the novice user can respond with `(quit)` while the expert user might be able to guide the proof further.

3.3 Simple Examples

We now provide straightforward examples of the use of the interface. Neither of the examples involves our PVS real analysis library; they demonstrate only the mechanics of the interface, and illustrate the master/slave relationship between Maple and PVS.

We assume that the Maple user has installed locally the C code, shell scripts and Maple library described in Section 3, and PVS. The first task is to initialise the interface, and check that we can prove that $2 + 2 = 4$:

```
> pvs := PvsStart("../pvslib"):
```

The `PvsStart` command launches a Tcl/Tk window and opens communications with a PVS session using libraries found in the given directory.

```
> ex1 := PvsProve(pvs, "g: FORMULA 2 + 2 = 4", "", ""):
```

The PvsProve command (i) takes a PVS session identifier, (ii) a formula in PVS syntax, (iii) a PVS library - the default is the prelude, and (iv) a PVS proof command - the default is ASSERT. The result of this command is shown in Figure 3.

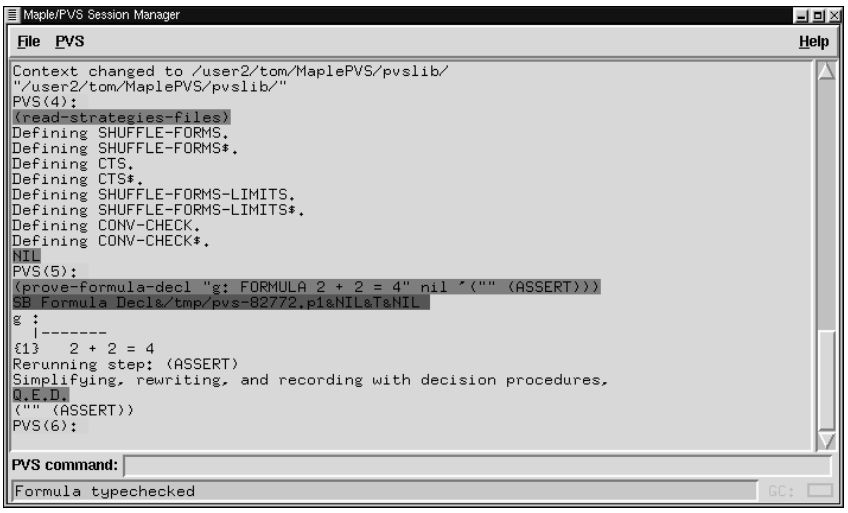

Fig. 3. Tcl/Tk window for the Maple/PVS interface

We confirm in Maple that the proof was successful using the PvsQEDfind command:

```
> PvsQEDfind(ex1);
```

$$true$$

The second elementary example is a proof that the definite integral given in Section 1.1 is undefined for certain values of its parameters. We prove property (3) via the command:

```
> PvsProve(pvs, "def: LEMMA FORALL (a,b:posreal) :
> b > a IMPLIES EXISTS (x:real) : 0 <= x AND x <= b AND
> not(member[real](x,({z:real|z /=a})))",
> "", "then (skosimp*)(then (inst 1 \"a!1\")(grind))");
```

For this example the proof argument to PvsProve is rather more complicated, and represents a single statement of the sequential PVS proof by repeated Skolemisation and flattening, explicit instantiation, and use of the grind tactic.

The above examples show that the Maple user controls the interface using Maple commands in a Maple session. The user can check that proof attempts have succeeded without needing to interact with (or even view) the Tcl/Tk window. This is present only as a gateway to PVS, used when proof attempts fail, or when a record of the logical steps used in a proof is needed.

4 Applications

In this section we demonstrate the use of the Maple/PVS interface to obtain proofs which use lemmas from the real analysis library discussed in Section 2. We motivate the use of the proof strategies contained in the library by first proving results without them.

4.1 Continuity of $\sqrt{x}$ for $x > 0$

Consider the following proof that $\sqrt{x}$ can be shown to be continuous for $x > 0$ by choosing $\delta = \epsilon\sqrt{x_0}$ in the definition of continuity:

$$\begin{aligned}
\delta &= \epsilon\sqrt{x_0} && \Longrightarrow \\
\delta &\leq \epsilon(\sqrt{x} + \sqrt{x_0}) && \Longrightarrow \\
|\sqrt{x} - \sqrt{x_0}| < \frac{\delta}{\sqrt{x} + \sqrt{x_0}} &\Longrightarrow |\sqrt{x} - \sqrt{x_0}| < \epsilon && \Longrightarrow \\
|\sqrt{x} + \sqrt{x_0}||\sqrt{x} - \sqrt{x_0}| < \delta &\Longrightarrow |\sqrt{x} - \sqrt{x_0}| < \epsilon && \Longrightarrow \\
|x - x_0| < \delta &\Longrightarrow |\sqrt{x} - \sqrt{x_0}| < \epsilon
\end{aligned} \tag{7}$$

The equality used is the example given in Section 1.1. The proof using the Maple/PVS interface is obtained with the command:

```
> ex3 := PvsProve(pvs, "sqrt_eq: LEMMA FORALL (x,y:posreal)
:
> abs(root(x,2)-root(y,2))*abs(root(x,2)+root(y,2)) =
> abs(x-y)", "roots", "then (skolem!)(then (use
> \"sqrt_difference\")(then (use \"abs_mult\")(assert)))");
```

Here we are using the PVS `roots` theory. We use this theory again to prove that

$$|\sqrt{x} + \sqrt{x_0}||\sqrt{x} - \sqrt{x_0}| < \sqrt{x_0}\delta \implies |\sqrt{x} - \sqrt{x_0}| < \epsilon \tag{8}$$

```
> ex4 := PvsProve(pvs, "sqroot_ineq: LEMMA
> FORALL (eps,x,y:posreal) :
> abs(root(x,2)-root(y,2))*abs(root(x,2)+root(y,2))
> < eps*root(y,2) IMPLIES abs(root(x,2)-root(y,2)) < eps",
> "roots", "then (skosimp)(then (lemma
> \"both_sides_times_pos_lt1\" (\"x\"
> \"abs(root(x!1,2)-root(y!1,2))\"
> \"pz\" \"abs(root(x!1,2)+root(y!1,2))\" \"y\"
> \"eps!1\"))(grind))");
```

This example demonstrates the greater proof power of the interface. Maple alone fails to obtain the identities needed in the above proof. Unfortunately, we need several explicit commands to guide PVS in each proof. The next stage of the development of the interface is the provision of a suite of strategies for performing analysis proofs with much less user guidance.

4.2 Using the Real Analysis Library

The previous example involved the sequential proof of properties of a standard ϵ-δ real analysis definition. By using the real analysis library described in Section 2, we can prove continuity and convergence directly for a wide range of functions. To illustrate this we use the `PvsProve` command with `top_analysis` as the library, and with either `cts` or `conv-check` as the proof strategy. For example, consider the initial value problem (IVP)

$$y'(x) = f(x, y), \quad y(a) = \eta \tag{9}$$

where y' denotes denotes the derivative of $y(x)$ with respect to x. Let D denote the region $a \leq x \leq b$ and $-\infty < y < \infty$. Then Equation 9 has a unique differentiable solution, $y(x)$, if $f(x, y)$ is defined and continuous for all $(x, y) \in D$, and there exists a positive constant L such that

$$|f(x, y_0) - f(x, y_1)| \leq L|y_0 - y_1| \tag{10}$$

holds for every (x, y_0) and $(x, y_1) \in D$.

Our intention is to use the interface and the real analysis library to verify conditions on the input function, $f(x, y)$, and the Maple solution, $y(x)$, of the IVP. For example, we prove that

$$f(x, y) = \frac{1}{e^{\pi - |6cos(x)|}} \tag{11}$$

is defined and continuous on the real number line using the `cts` strategy:

```
> ex5 := PvsProve(pvs, "g: LEMMA FORALL (y:real) :
> continuous(lambda (x:real) : 1/exp(pi - abs(6*cos(x))),y)",
> "top_analysis", "cts");
```

For an example of the verification of properties of a Maple solution, we consider the IVP

$$y'(x) = sin(x)y(x) + sin(x)y(x)^2, \quad y(0) = \pi\,. \tag{12}$$

We can obtain a closed form solution using a Maple procedure:

$$y(x) = \frac{-\pi\, e^{(-\cos(x)+1)}}{-1 - \pi + \pi\, e^{(-\cos(x)+1)}} \tag{13}$$

Since the IVP is of generalised Riccati type [21], we can check that the solution has only removable poles, i.e. that the solution has a limit at those points at which it is undefined. In other words, we prove that

$$\pi e^{1-cos(x)} - \pi - 1 \tag{14}$$

is convergent (i.e. has a limit) at

$$x = arccos(1 + \log(\frac{1+\pi}{\pi})) \tag{15}$$

via the `conv-check` strategy:

```
>  ex6 := PvsProve(pvs, "g: LEMMA convergent(LAMBDA (x:real)
:
>  -pi-1+pi*exp(1-cos(x)),acs(1+ln((1+pi)/pi)))",
>  "top_analysis", "conv-check");
```

We can also verify the the solution in (13) can never be zero, using the `grind` strategy:

```
>  ex1 := PvsProve(pvs, "g: FORMULA FORALL (x:real) :
>  -pi*exp(-cos(x)+1)/= 0", "top_analysis", "grind :defs
NIL");
```

These examples demonstrate the inference capability and expressivity of the interface augmented with a library of analytic proof strategies. The results can not be proved within Maple, and are not easy to prove by hand.

4.3 A Generic Application to IVPs

We now describe a methodology for validating and improving Maple procedures for solving IVPs of the form

$$y'(x) = r(x) - q(x)y(x), \quad y(a) = \eta, \quad x \in [a, b] \tag{16}$$

We can use the interface to check the following requirements on inputs (bearing in mind that each input can be a complicated symbolic expression involving parameters):

1. $r(x)$ and $q(x)$ are continuous over $[a, b]$;
2. $r(x) - q(x)y(x)$ is continuous, Lipschitz, and/or differentiable over $[a, b]$.

Answers to these questions provide a formal check on the existence and uniqueness of solutions for the given finite range. For example, we proved the continuity of $\sqrt{x}$ for all positive x in Section 4.1. We can prove that $\sqrt{x}$ is not Lipschitz for $x \in (0, 1)$ using the interface. Information regarding existence and uniqueness can be used to fine tune the procedure used to obtain a solution, by using relevant `assumes` clauses in Maple (e.g. `assume(r(x) - q(x)y(x), continuous)`) so that specialised solution techniques can be safely used.

Once a solution, $y(x)$, has been obtained, we can use the interface to check properties such as

1. $y'(x) - f(x, y) = 0$;
2. $y(a) = \eta$;
3. $y(x)$ has removable poles, non-removable branch points and/or is itself continuous.

The following prototype Maple procedure takes $r(x)$, $q(x)$, a, η and b, and supplies answers to some of the above questions using the inbuilt Maple `dsolve` procedure for obtaining $y(x)$.

```
qsolve := proc(r, q, a, η, b)
local pvs, z1, z2, z3, z4, z5, z6, sol, diffsol;
pvs := PvsStart( "../pvslib") ;
z1 := PvsProve(pvs,
    "g: FORMULA FORALL (v:I[a,b]) : continuous(lambda (x:I[a,b]) : r(x), v)",
    "top_analysis", "cts");
z2 := PvsProve(pvs,
    "g: FORMULA FORALL (v:I[a,b]) : continuous(lambda (x:I[a,b]) : q(x), v)",
    "top_analysis", "cts");
if not (PvsQEDfind(z1) and PvsQEDfind(z2)) then ERROR(`invalid input`)
else
    sol := dsolve({diff(y(x), x) = r(x) − q(x)y(x), y(a) = η}, y(x)) ;
    diffsol := diff(sol, x) ;
    z4 := PvsProve(pvs,
        "g: FORMULA FORALL (v:I[a,b]) : diffsol(v) = r(v) - q(v)*sol(v)",
        "top_analysis", "grind");
    z5 :=
        PvsProve(pvs, "g: FORMULA sol(a) = eta", "top_analysis", "grind");
    z2 := PvsProve(pvs,
        "g: FORMULA FORALL (v:I[a,b]) : continuous(lambda (x:I[a,b]) : sol(x),v)",
        "top_analysis", "cts")
fi;
if not (PvsQEDfind(z4) and PvsQEDfind(z5) and PvsQEDfind(z6)) then
    ERROR(`invalid solution`)
else sol
fi
end
```

Maple does have built in procedures for answering many of these questions, but, as shown in Section 1.1, can fail to detect the equality of two straightforward expressions and the continuity of a simple function. Using the interface helps the user to validate both the input and output of problems, and hence leads to improved use and understanding of the CAS.

5 Conclusions

We have presented an interface between Maple and PVS. The interface gives Maple users access to the proof capabilities of PVS, thus providing a means to gain more formal knowledge of Maple results.

We have created a tightly coupled interface between Maple and PVS, under which PVS is controlled by Maple. A small Tcl/Tk application sits between Maple and PVS, so that PVS looks like a black box to the Maple user. However, it also allows a more experienced user to interact directly with PVS.

In Section 1.1 we saw that Maple can fail to recognise a seemingly obvious equality, which could lead to an undetected division by zero, and also that Maple might apply

standard procedures without checking validity of the input. In Section 4 we showed how the interface can be used to correct these errors.

Our aim is to extend the applicability of the interface in two ways. Firstly the extension of the real analysis library discussed in Section 2 by adding new strategies, and secondly by providing Maple procedures which automate the checking of validity of input and output, as described in Section 4.3. These extensions will require an improvement in the communication between the two systems, both in terms of syntax of expressions, and in decision procedures based on failed proof attempts.

References

1. ABBOTT, J., VAN LEEUWEN, A., AND STROTMAN, A. Objectives of OpenMath. Available from `http://www.rrz.uni-koeln.de/themen/Computeralgebra-/OpenMath/`, Jan. 1995.
2. ADAMS, A., GOTTLIEBSEN, H., LINTON, S., AND MARTIN, U. Automated theorem proving in support of computer algebra: symbolic definite integration as a case study. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation, Vancouver, Canada* (1999), ACM Press.
3. BALLARIN, C., HOMANN, K., AND CALMET, J. Theorems and algorithms: An interface between Isabelle and Maple. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation* (1995), A. Levelt, Ed., ACM Press, pp. 150–157.
4. BAUER, A., CLARKE, E., AND ZHAO, X. Analytica - an experiment in combining theorem proving and symbolic computation. *Journal of Automated Reasoning 21* (1998), 295–325.
5. BERTOLI, P., CALMET, J., GUINCHIGLIA, F., AND HOMANN, K. Specification and integration of theorem provers and computer algebra systems. In *Artificial intelligence and symbolic computation (Plattsburgh, NY, 1998)*, no. 1476 in Lecture Notes in Computer Science. Springer-Verlag, 1998.
6. BUCHBERGER, B. Symbolic computation: Computer algebra and logic. In *Frontiers of Combining Systems* (1996), F. Baader and K. Schultz, Eds., Applied Logic Series, Kluwer Academic Publishers, pp. 193–220.
7. BUCHBERGER, B., JEBELEAN, T., KRIFTNER, F., MARIN, M., TOMUTA, E., AND VASARU, D. A survey of the theorema project. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation* (1997), W. Kuechlin, Ed., ACM Press, pp. 384–391.
8. CALMET, J., AND HOMANN, K. Classification of communication and cooperation mechanisms for logical and symbolic computation systems. In *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)* (1996), F. Baader and K. U. Schulz, Eds., Applied Logic, Kluwer, pp. 221–234.
9. CHAR, B. W. *Maple V language Reference Manual*. Springer-Verlag, 1991.
10. DENNIS, L. A., COLLINS, G., NORRISH, M., BOULTON, R., SLIND, K., ROBINSON, G., GORDON, M., AND MELHAM, T. The PROSPER Toolkit. In *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2000), vol. 1785 of *Lecture Notes in Computer Science*, Springer-Verlag.
11. DEWAR, M. Special Issue on OPENMATH. *ACM SIGSAM Bulletin 34*, 2 (June 2000).
12. DOLZMANN, A., AND STURM, T. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bulletin 31*, 2 (June 1997), 2–9.
13. DUTERTRE, B. Elements of mathematical analysis in PVS. In *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs '96* (Turku, Finland, Aug. 1996),

J. von Wright, J. Grundy, and J. Harrison, Eds., vol. 1125 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 141–156.

14. GOTTLIEBSEN, H. Transcendental Functions and Continuity Checking in PVS. In Harrison and Aagaard [16], pp. 198–215.
15. GRAY, S., KAJLER, N., AND WANG, P. S. MP: A Protocol for Efficient Exchange of Mathematical Expressions. In *Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'94), Oxford, GB* (July 1994), M. Giesbrecht, Ed., ACM Press, pp. 330–335.
16. HARRISON, J., AND AAGAARD, M., Eds. *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000* (2000), vol. 1869 of *Lecture Notes in Computer Science*, Springer-Verlag.
17. HARRISON, J., AND THÉRY, L. Reasoning about the reals: the marriage of HOL and Maple. In *Logic Programming and Automated Reasoning* (1993), A.Voronkov, Ed., no. 698 in Lecture Notes in Artificial Intelligence, LPAR'93, Springer-Verlag, pp. 351–359.
18. JACKSON, P. *Enhancing the NUPRL Proof Development System and Applying it to Computational Abstract Algebra*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, Apr. 1995.
19. JENKS, R. D., AND SUTOR, R. S. *AXIOM: the scientific computation system*. Numerical Algorithms Group Ltd., 1992.
20. MONAGAN, M., GEDDES, K., HEAL, K., LABAHN, G., VORKOETTER, S., AND MCCARRON, J. *Maple6 Programming Guide*. Waterloo Maple Inc., 2000.
21. MURPHY, G. M. *Ordinary differential equations and their solutions*. D. van Nostrand Company, Inc., 1960.
22. OWRE, S., SHANKAR, N., AND RUSHBY, J. M. *User Guide for the PVS Specification and Verification System*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.
23. OWRE, S., SHANKAR, N., RUSHBY, J. M., AND STRINGER-CALVERT, D. W. J. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, Sept. 1999.
24. OWRE, S., SHANKAR, N., RUSHBY, J. M., AND STRINGER-CALVERT, D. W. J. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, Sept. 1999.
25. PAULSON, L. C. *The Isabelle Reference Manual*. Computer Laboratory, University of Cambridge, February 2001.
26. POLL, E., AND THOMPSON, S. Adding the axioms to AXIOM: Towards a system of automated reasoning in Aldor. Tech. Rep. 6-98, Computing Laboratory, University of Kent at Canterbury, May 1998.