Applications of real number theorem proving in PVS

DOI 10.1007/s00165-012-0232-9

Formal Aspects of Computing (2013) 25: 993–1016

Formal Aspects of Computing

Applications of real number theorem proving in PVS

Hanne Gottliebsen[1], Ruth Hardy[2], Olga Lightfoot[1] and Ursula Martin[1]

[1] School of Electronic Engineering and Computer Science, Queen Mary, University of London, London, UK
[2] School of Computer Science, University of St Andrews, St Andrews, UK

Abstract. Real number theorem proving has many uses, particularly for verification of safety critical systems and systems for which design errors may be costly. We discuss a chain of developments building on real number theorem proving in PVS. This leads from the verification of aspects of an air traffic control system, through work on the integration of computer algebra and automated theorem proving to a new tool, NRV, first presented here that builds on the capabilities of Maple and PVS to provide a verified and automatic analysis of Nichols plots. This automates a standard technique used by control engineers and greatly improves assurance compared with the traditional method of visual inspection of the Nichols plots.

Keywords: Real number theorem proving, PVS, Maple, Control systems, Test suite, Air traffic control, Higher order theorem proving

1. Introduction

The purpose of this paper is to highlight a trajectory in the development of real number theorem proving, with applications to real world problems in engineering and design verification in mind. We describe a sequence of work carried out by ourselves and others in the theorem prover PVS over recent years, some presented here for the first time, and identify related work and a roadmap for future research. Our line of enquiry has led us to develop theories in PVS, and to use them in verifying safety-critical systems using the theorem prover directly. We have exploited the work on PVS by adding facilities to verify side conditions on calculations to commonly used mathematical software. We have also created NRV, a special purpose tool that applies theorem proving to automate a standard task in control engineering.

By real number theorem proving we mean the machine verification in a computational logic system of results about real valued functions, typically equalities, inequalities, and properties such as continuity or differentiability, over elementary functions (combinations of powers, cos, sin, exp, log) and subject to constraints on the variables.

Correspondence and offprint requests to: U. Martin, E-mail: ursula.martin@eecs.qmul.ac.uk

Here is an example of the kind of problem we are interested in taken from an air traffic control application:
Let a, b, c be the lengths of the edges of a triangle and let ϕ be the angle between the edges of length a and b, where a, b and c are functions of the variable t, and ϕ and min_sep are constants. Prove that

$$c^2(t) = a^2(t) + b^2(t) - 2 * a(t) * b(t) * cos(\phi) > min\_sep.$$

Where hand calculation or reference to textbooks does not suffice, the traditional approach to such questions has been to use numerical or symbolic computational methods. The problem above was initially investigated using the computer algebra system Maple [Map12]. Numeric and symbolic methods are supported by years of research into fundamental and applied aspects, and computer algebra systems typically offer a great variety of standard algorithms that have been used to develop tools to support specialist applications. However the correctness of such tools is a cause for concern, as it is often the user's responsibility to ensure that input data meets all the side conditions required to justify the use of the algorithm selected.

Decidability results for problems of this kind have been the subject of deep work in model theory by Macintyre, Wilkie and others [Mar96] in recent years. Tarski's theorem says that the first-order theory of reals with +, *, =, and > allows quantifier elimination and hence is decidable, and software such as QEPCAD [BEH+03] implements a cylindrical algebraic decomposition algorithm to do this effectively. However the problem of quantifier elimination over a real closed field is exponential in the number of bound variables to be eliminated, and implementations can, in general, only handle relatively small examples. Nonetheless the QEPCAD implementation is powerful enough to deal with real-life problems in some important applications domains where large formulas do not occur. For problems that only require linear arithmetic, the situation is better and systems for satisfiability modulo theories such as SRI's Yices [For] which combine SAT solvers and decision procedures for a variety of theories including linear arithmetic over integers and reals have been applied with problems involving hundreds of variables.

Machine verification aims to solve such problems by providing a formal proof, developed from axioms and rules of inference in a computational logic system. This draws on two major themes in twentieth century computer science: the science of programming and the formalisation of mathematics. Among the earliest work in computational logic was de Bruijn's 1955 system AUTOMATH, which checks a hand proof input by the user. In 1977, Jutting published a formalisation of Landau's Grundlagen der Analysis, a text on the foundations of mathematical analysis, done in AUTOMATH [vBJ77]. Milner and Gordon were among the first to develop the computational logic tools that made it practical to exploit the science of programming and apply the theoretical ideas of Hoare, Milner and others to obtain correctness proofs for software, hardware and systems, in domains where calculation or verification by hand would be totally infeasible.

As modern computational logic systems such as Coq, HOL, Isabelle and PVS have developed in logical power, availability of libraries, and capacity for automation, and as our understanding of how best to use them has grown, formalised proofs in many areas of mathematics have been developed. Substantial areas of analysis were developed in the MIZAR project, as far as the Hahn–Banach theorem [NT93], and a textbook on lattices [GHK+80], as well as Fleuriot's application of non-standard analysis in mechanising proofs from Newton's Principia [Fle00] done in Isabelle/HOL. Current work by Akbarpour and Paulson [AP08] integrates QEPCAD as an additional rule of inference in the resolution theorem prover Metis allowing automatic proof of non-linear inequalities involving functions such as ln, exp, $sine$ and $cosine$. While in general the capabilities of computational logic systems have lagged far behind the needs or inclination of research mathematicians, their value is now being recognised in handling proofs that are just too long or detailed for a traditional hand proof. Striking recent achievements are the mechanisation and simplification of the proof of the four colour theorem, in Coq [Gon], and the FlySpeck project [Hal] which has so far mechanised in HOL Light (with some proofs done in Isabelle and Coq) a substantial component of Hales's proof of the Kepler conjecture, including vital background facts such as the Jordan Curve theorem and the main purely combinatorial part of the proof comprising a result in graph theory involving an analysis of millions of graphs.

However it is engineering applications that have been the driving force behind much verification work, including our own. The demands of these applications have led to the developments in expressiveness and performance that have made our work possible. Many applications in areas such as aerospace, hardware verification, or hybrid systems involve reasoning about time, space or other continuous quantities. Hence the verification environment must provide a capability for reasoning about functions over real numbers, either as part of a freestanding computational logic system such as HOL, Isabelle or PVS, or as a specialist tool to be embedded in other software, or incorporated in a broader architecture of verification tools such as Rushby's "Evidential Tool Bus" [dMOR+05], along with SAT solvers, model checkers and so forth.

The first stage in developing such a capability is to formalise the underlying mathematics: for example, verifying identities between trigonometric functions requires formalising the basics of real analysis to the level of limits, convergence, power series representations and so on, using these to obtain proofs of basic properties, and then setting up automated tactics which can use the basic facts to deal with the identities that arise in practice. Applications make additional demands. Developing and verifying an algorithm may involve a specialist in repeated proofs of, or attempts to prove, variations of the same result, for example with slight variations on side-conditions. The discharge of routine verification requirements often requires the proof of a great many similar simple expressions by a non-specialist user. This makes it worthwhile to develop a high degree of automation so that the results can be effectively reused and so that the complexities of routine problems such as dealing with singularities and case splits are hidden from the user.

PVS is a system which includes both its own specification language and a powerful higher order theorem prover. In this spirit, we will describe the real number theorem proving capabilities that have been developed for the PVS engine and then consider a range of applications of PVS that exploit these capabilities. The rest of the paper is structured as follows:

- In Sect. 2 we discuss the initial development of *tools for real number theorem proving* in PVS by Dutertre [Dut96], Gottliebsen [Got00, Got01], and others. [ADG+01, GS].
- In Sect. 3 we consider real number theorem proving in *verification of algorithms*, exemplified by work at NASA on algorithms for self-controlled airport approaches [CM00, CGBK04, CM05].
- In Sect. 4 we discuss an example of *embedded verification support*, namely Maple–PVS, that provides external calls to PVS from the computer algebra system Maple.
- In Sect. 5, we turn to the main contribution of this paper, an example of *packaged tool support*, and describe NRV, a new tool for verifying aerospace engineering requirements. NRV combines Maple, QEPCAD, and PVS to discharge design requirements for control systems involving the Nichols plot, replacing the plotting and inspection of many hundreds of curves with a parametrised and verified analysis. The tool either proves that the system meets its requirements, produces a counter example to show that it does not or provides information to point the control engineer at the problematic areas that caused the automatic processing to fail.
- Finally, in Sect. 6, we draw some conclusions and discuss directions for future work.

2. Real number theorem proving in PVS

PVS [ORS92] is a higher order theorem prover with its own typed specification language and a lisp-like strategy language. PVS was first released in 1992 and has been continuously developed since, with the latest version being released as open source after being ported from allegro common lisp to CMU lisp. There is a strong tradition of the PVS user community contributing libraries to PVS, most of which are held in a publicly accessible collection maintained by the formal methods team at NASA Langley Research Center.

PVS is used in both academia and industry, with its particular strength being the powerful specification language, which gives considerable expressive power through dependent types, and is backed up the many built-in strategies for proof. PVS contains the following:

Theories which hold the specifications. Each theory covers a particular area, for example operations on lists. Theories may import other theories, much like libraries are used in a programming language.

Commands which are the basic proof commands of PVS. These are used for the proving of theorems in PVS.

Libraries which hold theories with proofs to be shared with other specifications. These may be polymorphic in that they may depend on values passed to the library theory.

Strategies which are programs combining commands and other strategies into new proof strategies. In some other theorem provers these are referred to as tactics. In PVS strategies are written in a lisp-like strategy language. One of the main strengths of PVS is the many built in strategies, for example grind which essentially does a brute-force search for a proof. In most cases it is possible, and advisable, to control the behaviour of the strategies using the command options available. Although not much used it is possible to share strategies like libraries, as can be seen with the strategy collections field [MM] and manip [Vit03], both of which provide strategies for handling expressions over the real numbers.

Typical application areas for PVS are hardware verification, control engineering, and air traffic control, including control of aircraft on the ground. These require both symbolic manipulations and calculations with actual numbers, handled either by approximate representations or by calculations to arbitrary precision. Theories in PVS to provide these capabilities have been developed by a number of different individuals and organisations, with different goals and approaches. We summarise this work in the rest of the section.

Dutertre constructed the first PVS library for real analysis [Dut96], covering the basics of calculus, such as limits, continuity and differentiation. Traditionally, in mathematics, the next step might be infinite series and functions defined by power series. This was worked on in parallel by Gottliebsen [Got00] and by the NASA LaRC team [CM00], with Gottliebsen taking a more foundational approach, using equational definitions, and the NASA team developing an axiomatic version. This illustrates quite clearly two different approaches to library development in general: Gottliebsen was concerned with having an exposition of the traditional mathematical development of transcendental functions as well as the library being usable, whereas the NASA LaRC team was more focused on usability. For example, the NASA LaRC version contained from the very beginning many lemmas about bounds of the trigonometric functions, since this is what they needed for their applications work on air traffic management [CM00].

The NASA LaRC repository [But] now contains PVS theories for limits of functions over reals, continuity and differentiation, finite and infinite sums, and functions such as sine, cosine and exp defined by their power series. In each case, there is an extensive set of lemmas and theorems expressing the "usual" properties. The theories of this repository have been developed over years by a group of people somewhat larger than the NASA LaRC/NIA team. Later, Lester proved many of the axioms in the axiomatic version of the library, leading to a foundational version of the applications oriented NASA LaRC library.

It is still the case, however, that Gottliebsen's library contains many lemmas about general properties of analysis, particularly on power series and transcendental functions that are not included in either of the NASA LaRC versions. Hardy has subsequently extended this library for her work on Nichols plots (Sect. 5), continuing the foundational approach.

The merging of the Gottliebsen/Hardy library with the NASA LaRC library is ongoing, raising the interesting issue of simultaneous library development by different groups. A basic problem here is that one library considers infinite series to start at index 0, whereas the other library considers them to start at index 1. The two definitions are provably (even in PVS) equivalent, however working with both representations at the same time is rather cumbersome, and even transferring proofs between representations is no small task. In order to help the PVS community avoid such parallel developments in the future, NASA LaRC provides a web space where users may post their ongoing developments while they are still fairly raw.

There has also been much work in PVS and other systems on theories supporting non-symbolic manipulations, for example interval arithmetic [ML05] and floating point arithmetic [BM06]. Support for verifying floating point arithmetic has been available for example in HOL-Light for some time [Har98] and this has been used by Intel to verify both hardware and software implementations of floating point algorithms. Moore et al. and Russinoff have verified floating point operations of the AMD K5 processor using ACL2 [MLK98, Rus99]. Jacobi has formalised the IEEE floating point standard and a floating point unit implementing it using PVS [Jac02].

We are concerned here, however, with verification at a higher level than floating point operations. Our goal is to apply theorem proving to the problems in real algebra and analysis that arise in engineering applications. In the various formulations of the real numbers and of real analysis in PVS the following areas are covered:

- Series (finite and infinite)
- Power series
- Trigonometric functions and their inverses
- Exponentiation and the inverse
- Continuity
- Differentiability
- Approximation using Taylor series
- Vectors over the real numbers (2-dimensional, 3-dimensional and N-dimensional)

Alongside the theories for real analysis and transcendental functions, all the libraries provide strategies which offer automation for proofs involving real numbers. Most noteworthy are Muñoz and Di Vito's `manip` [Vit03] and `field` [MM] which allow for basic equational manipulation of expressions with real numbers. Without these two

sets of strategies, one needs to specifically invoke lemmas in order to e.g. subtract a number or variable from both sides of an equation. However, with the strategies, the work is little more difficult than when done with pen and paper. Gottliebsen developed strategies for use with DITLU (Sect. 4), in particular for proving continuity (`cts`) and differentiability (`diff`) of functions over intervals. These implementations are simple in that they implement only basic "high school" methods for proving continuity and differentiability—essentially that the identity and constant functions are continuous (differentiable), and that combinations (using addition, subtraction, multiplication and composition) of functions preserve continuity (differentiability). However, naive as this may seem, it turns out to serve very well in many practical examples, e.g., Hardy's work on Nichols plots (Sect. 5).

3. Small aircraft transportation system

As we have seen, many potential application areas for PVS require theorem proving with real numbers. In this section we will discuss some of these. Over the years support in PVS has been developed for these applications by providing libraries, strategies and external "hooks" for other systems to use. As a result, PVS has become very useful both as a stand-alone tool for modelling and verifying systems and as a back-end to other tools, e.g., to provide extra assurance in the results of computer algebra systems or specialised tools to solve specific classes of engineering problems. The examples discussed in this section will illustrate the development of back-end support for computer algebra systems and the use of real number theorem proving in PVS to model and verify a safety-critical system. NRV is our example of a tool to solve a specific class of problems in control engineering and we will go on to describe it in Sect. 5.

Both DITLU and Maple–PVS are examples of applications of PVS where the emphasis is on doing relatively simple proofs in high volume, with the actual specifications mostly relying upon built-in or library specifications. In this section we will discuss the verification of some air traffic management procedures. This involves a much more complicated specification and proofs, many of which are not obvious candidates for reuse.

The Small Aircraft Transportation System (SATS) project [KS] led by NASA LaRC was concerned with investigating new procedures for air traffic management for small airports within the USA. The small airports considered typically have neither radar coverage nor a local control tower: instead they are controlled by air traffic controllers located at nearby larger airports. In many cases, the small airports are out of direct radio contact with the controlling tower due to geographical features. This implies that the procedures ensuring the safety and fair access to the small airports cannot rely on active radar control or direct radio contact. In fact the traffic must be controlled by automated systems, both on the ground and on the aircraft.

Evidently air traffic management procedures are safety critical—if the procedures are not safe, it may cost lives. As part of SATS, theorem proving with PVS was used to ensure that the suggested procedures, developed by air traffic management engineers, were indeed safe in principle. It is important to realise here the difference between verifying the procedures and verifying a finished system. The procedures are basically the design, described by graphics, formulae and rules for interaction between different aircraft, remote controllers and the system. A finished system would be an implementation of the procedures, involving the actual hardware and software running the system. The verification for SATS dealt with the procedures, since if the procedures are flawed, an implementation of them will be unsafe. Of course, proving the procedures (the design) safe does not give a guarantee that an implementation will be safe—that would require verification that the implementation then corresponds to the procedures, without adding any implementation specific problems.

As with many engineering projects, there were several attempts at getting the SATS procedures just right, and two of the main iterations were modelled in PVS [CGBK04, CM05]. The first procedure proposed was based on an algorithm developed by air traffic control experts that grants landing permission based on predicted separation in time.

Figure 1 shows the plan of a SATS airport. Aircraft arriving from above the dotted line fly directly to the Intermediate Fix (*IF*) and then via the Final Arrival Fix (*FAF*) to the runway, while aircraft arriving from below the dotted line fly to the nearer of the two Initial Arrival Fixes (*IAF_L, IAF_R*) and then turn in the area marked *Virtual IAF* before flying on to the runway.

Aircraft inside the circle referred to as the Self Controlled Area (SCA) accept responsibility for separation. The algorithm is responsible for granting permission to enter the SCA. It uses current speed, direction, position, and aircraft-specific landing speed profiles to predict whether aircraft requesting access will maintain their separation in time until landing. Permission is granted if the predicted separation is acceptable. The verification task was to prove that this separation in time ensures continued spatial separation. The proof approach was to consider each combination of entry regions for pairs of aircraft requesting permission to land.

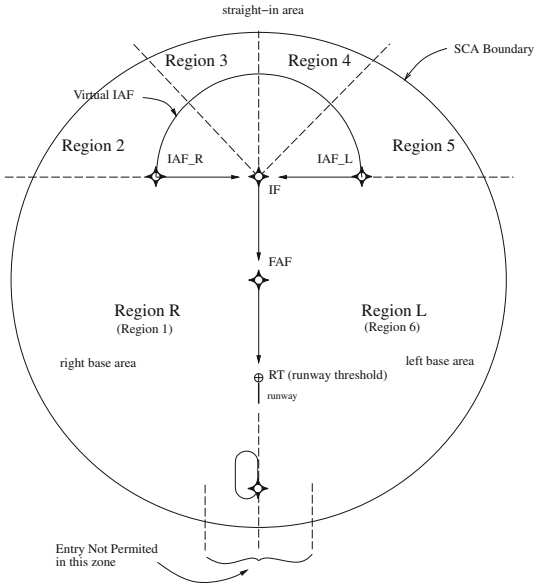


Fig. 1. Layout of a SATS airport [CGBK04]

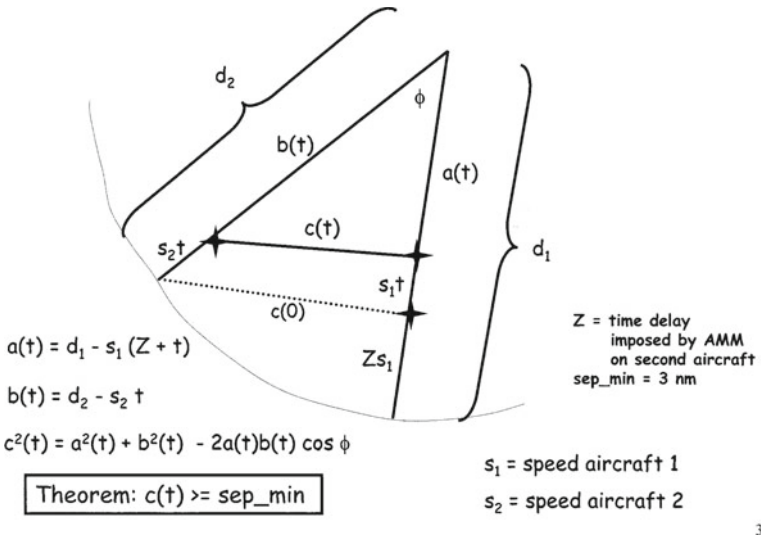


Fig. 2. Paths of two aircraft entering Region R[CGBK04]

Since there is no *stacking* of aircraft in this algorithm, in PVS the airspace was modelled as two-dimensional, ignoring altitude all together. The `reals` library in PVS was used extensively throughout the proof effort. One very useful observation was in the case where both aircraft are entering through Regions R or L. Figure 2 shows a diagram of the Region R case. We are concerned with the distance between two aircraft given as a function of time, $c(t)$. This distance is calculated as:

$$c^2(t) = a^2(t) + b^2(t) - 2 * a(t) * b(t) * cos(\phi),$$

where ϕ is the angle between the flight paths and a and b are given functions of t. It turned out that the time separation chosen for the algorithm was large enough to ensure spatial separation $2 * a(t) * b(t) * cos(\phi)$ is approximated by $2 * a(t) * b(t)$. Thus in this case, human insight greatly simplified the formal reasoning about transcendental functions.

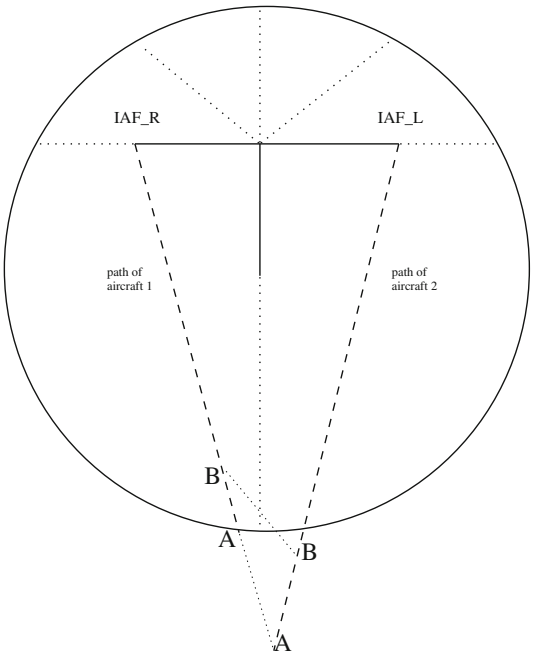


Fig. 3. Paths of two aircraft entering near the take-off path [CGBK04]

The proof work found a significant error in the algorithm that extensive simulation had failed to detect. Failed proof attempts revealed the scenario illustrated in Fig. 3. Here two incoming aircraft are separated while outside the SCA. As the paths are diverging, one might assume that the aircraft will continue to be separated. However, the assumption is false: if the second aircraft to enter the SCA is flying faster than the first aircraft, the *point of closest approach* can occur after penetration of the SCA. As illustrated in the figure, at time A both aircraft are just outside the SCA, while at time B one is inside but the distance between the aircraft has reduced.

This error detected by proof made the algorithm unsafe. It was resolved by banning entry through the region around the take-off path, leading to the *no-entry zone* shown at the bottom of Fig. 1.

This example clearly illustrates both that careful simulation and testing cannot ensure that all significant cases have been covered and that human intuition about even quite simple dynamic systems is often faulty. Theorem proving and other formal methods are vital for mitigating these issues in the design of complex safety critical systems.

The following are the main conclusions from the verification done for SATS:

- A problem with the original algorithm was identified by the lack of a proof, despite not being discovered during extensive testing.
- The project also shows the importance of analysing the domain before attempting verification with the capabilities of the tools at hand in mind: the abstraction from dealing with trigonometric functions over vectors in a coordinate system to dealing with a simple second order polynomial equation greatly decreased the difficulty of a particular part of the proof given the support available to us in PVS.
- The project provided a very clear example of a major difference between testing/simulation and verification, namely that illustrated in Fig. 3. This is very useful as a practical example of why such safety critical systems should be analysed formally as well as tested for validation.
- New PVS code was developed, particularly to improve the support for vector spaces, covering 2- and 3-dimensional spaces and a generalisation to n-dimensional spaces. Basic versions of these were in PVS before, but they were expanded and standardised during the SATS project. This also includes development of PVS theory for calculations on point of closest approach, although this was ultimately not used in the final set of proofs.
- In general it is quite hard to do theorem proving with numeric values: however it is not always possible to stay within the realm of symbolic verification, and better support for theorem proving with real numbers is needed.

4. Maple–PVS

In this section we discuss our work on using PVS to augment the capabilities of a computer algebra system. This began with Definite Integration Table Look-Up (DITLU) [AGLM99b, AGLM99a], a prototype tool designed to mitigate the problems caused in computer algebra systems, such as Maple and Mathematica, that use look-up tables in standard calculations such as definite integrals, but fail to verify that all the side-conditions for the chosen table entry actually hold. DITLU used PVS as a back-end to a computer algebra system to carry out symbolic integration by table look-up with all side-conditions verified by PVS.

DITLU clearly showed the value of using a theorem prover as a back-end to a computer algebra system, so coupling Maple more closely with PVS seemed like a natural next step. We envisaged Maple providing the user with all its standard capabilities augmented with an option to have PVS verify properties that the Maple algorithms assume but do not check. Our motivation and approach was quite different from approaches such as that described in [BHC95] where the computer algebra system provides computational support to an interactive theorem prover.

Maple–PVS [ADG+01] provides an interface between the computer algebra system Maple and the theorem prover PVS, with PVS acting as a back-end to Maple. PVS is used to check properties such as continuity which the Maple algorithms assume but do not check. The core of the interface comprises a set of restricted functions written in Maple which formulate the necessary checks on definedness, continuity, differentiability and the like. The actual checks are then carried out by PVS.

The idea is that a user runs Maple, and uses the restricted functions. The functions use the interface to PVS to check preconditions on the inputs to the Maple algorithms that are not checked by Maple itself. In order for PVS to be (mostly) hidden from the Maple user, it is essential that there be adequate automation support in PVS and that any new Maple–PVS specific Maple functions are written to take advantage of this automation. Within the interface, it is possible to do some interaction with PVS, but this rather takes away from the idea of hiding PVS behind Maple, and we believe that in most cases, Maple users would rather not have to do theorem proving, even if they are happy to have the greater assurance that comes from having a theorem prover checking necessary preconditions that are not checked by Maple alone.

As an example, we will discuss the Initial Value Problem. The initial value problem is concerned with solving equations of the form

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

where r and q are real valued functions. A solution exists under certain restrictions:

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 $[a, b]$.

In this case, Maple returns a value for y, but does not consistently and fully check that the requirements on the standard solution are met. We can use the interface to check these requirements on inputs (bearing in mind that each input can be a complicated symbolic expression involving parameters).

Of course, to run these checks in PVS, we rely on automation being available within PVS for these specific types of problems. Answers to these questions provide a formal check on the existence and uniqueness of solutions for the given finite range.

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

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

That is, we can check properties which determine if the proposed solution is indeed a valid solution.

```
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
```

Fig. 4. Maple procedure using PVS to enhance the built-in differential equations solver

Figure 4 shows a prototype Maple procedure which takes as input $r(x)$, $q(x)$, a, η and b, and supplies answers to some of the above questions using the inbuilt Maple `dsolve` procedure for obtaining $y(x)$. This prototype is somewhat naive in that it does not take into account for example non-termination of the call to PVS. Also, an implicit assumption on the semantics of operations such as multiplication and subtraction being the same in PVS and in Maple is made. Maple does have built in procedures for answering many of these questions, but 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.

Another application of the Maple–PVS interface is that of verifying equations and inequalities, forming the basis of the tool NRV used for asserting properties of Nichols plots [Har06]. This work is described in Sect. 5.

The Maple–PVS interface was a natural follow-on from DITLU, in that it allows us to use Maple directly to implement new procedures, mimicking standard Maple procedures, but adding additional checks of side conditions.

In order to make good use of this interface, such procedures must be written in Maple, requiring a certain level of expertise from the Maple user. However, given procedures like the one for solving differential equations above, any Maple user may use these procedures as safer alternatives to the original ones.

In short:

- This was the first actual interface to use PVS as a back-end, with all the issues of getting two systems to work together. In particular, we had to consider how to handle any need for hand driven proofs in PVS.
- The implementation allows for some interaction with PVS, which is then used more as a grey box than a truly black box.
- The standard use of the interface uses PVS as a black box, requiring a high degree of automation within PVS to be useful. We provided the automation needed for examples such as the Initial Value Problem.

- The interface may be used in a power user vs user fashion: the power user develops new procedures relying on PVS, and those procedures are then available to regular users. Since the new procedures make calls to PVS, it is necessary for the power user to have some understanding of PVS.
- The Maple–PVS interface spurred the PVS developers into improving the interface to PVS in order to make it more feasible to use it as a back-end. These changes are now in the standard distribution of PVS.

5. Control laws

In this section, we present a new tool, the Nichols plot Requirements Verifier (NRV) [Har06]. The tool provides a symbolic, fully verified, check of properties of Nichols plots, something which is traditionally done only by visual inspection. NRV is built using Maple–PVS and QEPCAD, and has a graphical user interface, which is very simple to use and completely shields the user from the intricacies of PVS. Furthermore, the decision procedures implemented in Maple and used in the tool have been modelled and formally verified using PVS.

First we discuss some general background of control engineering, the use of Nichols plots, the mathematics involved and the commonly used methods for deciding on the correctness. We then discuss the decision procedure used in NRV, and finally present the new tool, using an inverted pendulum as a case study.

The process for designing control systems is well established and well documented [BKM01, Oga97, Pra00]. In classical control engineering mathematical formulae modelling the behaviour of dynamical and control systems are developed, often as difference or differential equations. These formulae or *models* can be considered to be specifications for systems, which may later be implemented in software or hardware. In control engineering, the model rather than its implementations is referred to as the control system. Dynamical models are usually considered in terms of their response or behaviour when exposed to some input or force. In general, systems are modelled mathematically in one of three domains: (1) the time domain, (2) the complex domain (also known as the *s-domain*), (3) the frequency domain. Control systems are then analysed with respect to some design criteria to ensure they display the correct behaviour. This analysis classically uses numerical techniques, often involving the visual inspection of a number of plots, or the performance of a number of simulations. Systems are analysed for a number of sample inputs and the assumption is made that the results of this analysis also hold for all the values between sample points. This assumption is not sound and may lead to incorrect conclusions being drawn about a system if it behaves unexpectedly between sample points. This problem is amplified when the precise values of parameters are unknown (*uncertain parameters*); the system is only analysed for a limited number of values of the parameters (and combinations of values) and the assumption is made that the results of the analysis hold for *any* values of the parameters (and combination of values). Conclusions are drawn about the entire system without formal justification.

Symbolic techniques provide a clear benefit over numerical techniques as they allow analysis to be performed over a range of values rather than at specific sample points; however, they do not provide guarantees of correctness unless implemented in a suitably formal setting. Formal theorem proving techniques provide guarantees that results provided are correct; however, these techniques are largely unfamiliar to control engineers and are considered generally quite difficult to use, especially for those with little or no background in formal methods. This complexity of formal methods makes their integration into existing control system development infeasible without high levels of automation.

5.1. Modelling control systems

Systems of ordinary or partial differential equations (ODEs or PDEs) are used to model the outputs of continuous-time systems in terms of the rates of change of their state variables over time. A general linear differential equation of order n for a system with input $u(t)$, output $y(t)$ and initial conditions $y(0) = y_0$, $y'(0) = y_1, \ldots, y^{n-1}(0) = y_{n-1}$, is shown in Eq. 1.

$$a_n y^n(t) + a_{n-1} y^{n-1}(t) + \cdots + a_1 y'(t) + a_0 y(t) = u(t). \tag{1}$$

The Laplace transform provides a method for deriving a representation of a system in the complex or s domain from equations representing it in the time domain, with

$$\mathcal{L} : \mathbf{R}^{\mathbf{R}} \longrightarrow \mathbf{C}^{\mathbf{C}}.$$

Given a continuous-time linear system, the resulting equation is a rational equation in the complex variable s, which is comparatively easy to solve.

The Laplace transform of a function f of time is defined in terms of a *transformation integral*

$$\mathcal{L}[f] = F = \lambda s.\ \int_0^\infty f(t)e^{-st}\,dt.$$

The Laplace transform is defined only if this integral converges. A sufficient condition for convergence is that

$$\int_0^\infty |f(t)|e^{-\sigma t}\,dt < \infty$$

holds for some real positive σ. If there exist real M and $\alpha > 0$ such that $|\ f(t)\ | < Me^{\alpha t}$ for all positive t, then the latter integral will converge for $\sigma > \alpha$ [DB01] and the Laplace transform of f will certainly be defined.

Using the transformation integral, the Laplace transform of the n-th derivative of a signal over time is calculated as

$$\mathcal{L}[f^n] = \lambda s.\ s^n F(s) - s^{n-1}f(0) - s^{n-2}f'(0)\cdots - f^{n-1}(0).$$

This is an important transformation and can be substituted directly into sets of differential equations to give the Laplace transform. For instance, given a general linear differential equation (see Eq. 1) with all initial conditions set to zero (i.e, $f(0) = 0, f'(0) = 0, \ldots, f^{n-1}(0) = 0$), the Laplace transform is

$$a_n s^n\, Y(s) + a_{n-1}s^{n-1}\, Y(s) + \cdots + a_1 sY(s) + a_0 Y(s) = U(s).$$

The Laplace transform can be used to produce the complex valued *transfer function*, which represents the ratio of the output variable to the input variable in the s-domain. The transfer function $F(s)$ is simply calculated as the quotient of the Laplace transform of the output of the system $Y(s)$ and the Laplace transform of the input of the system $U(s)$ with all initial conditions assumed to be zero.

The transfer function of a general linear differential equation (Eq. 1) is

$$F(s) = \frac{Y(s)}{U(s)} = \frac{1}{a_n s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0}.$$

Thus the Laplace transform turns differential equations into multiplication by s, and hence maps the problem of solving linear differential equations into the problem of solving polynomial equations.

Transfer functions are usually given as a description of the behaviour of the blocks in control diagrams, which is a standard graphical way of modelling dynamical systems.

5.2. Background mathematics

The work presented in this section relies on analysing functions for convexity, the definitions of which appear in most introductory calculus text books [Ada95, Spi73]. Although the definitions (and terminology)[1] vary slightly, fundamentally they represent the same geometric idea; that is that in some convex set $D \subseteq \mathbf{R}^n$ a function $f : D \to \mathbf{R}$ is convex if for all $\mathbf{x}, \mathbf{y} \in D$ the line segment between $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ lies on or above the graph of f. A function $f : D \to \mathbf{R}$, where D is a convex set in $\mathbf{R}^n$, is defined as *convex* (as illustrated in Fig. 5) if the following inequality holds for all $\mathbf{x}, \mathbf{y} \in D$ and $0 \leq \theta \leq 1$:

$$f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta)f(\mathbf{y}) \tag{2}$$

Reversing the inequality of Eq. 2 gives the definition of a *concave function*. A function is *strictly convex* (or *strictly concave*) if the inequality is strict. A curve $\{(\mathbf{x}, y) :\ y = f(\mathbf{x})\}$ defined by a function $f : D \to \mathbf{R}$, where $D \subseteq \mathbf{R}^n$, is *convex/concave* in the regions in which the function is convex/concave. A function that is linear in all variables is both convex and concave but neither strictly convex nor strictly concave.

A differentiable function f in one variable is convex on the convex set $D \subseteq \mathbf{R}$ iff $\forall\, \mathbf{x},\mathbf{y} \in D\ f'(x)(y-x) \leq f(y) - f(x)$ and is concave on D iff the inequality is reversed. This is equivalent to the condition for convexity that f' is not decreasing on D and for concavity that f' is not increasing on D. A twice differentiable function f in one variable is convex on D iff $f'' \geq 0$. These theorems and their proofs have been formalised in PVS [Har06], following a treatment in [Art64].

[1] [Spi73] refers to functions as *convex* or *concave*, whereas, [Ada95] uses *concave up* and *concave down*.

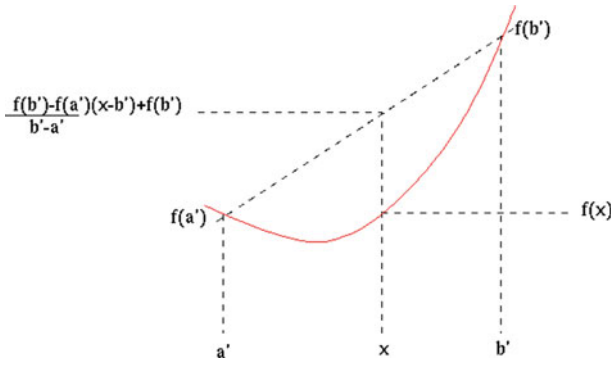


Fig. 5. Convexity of a curve

The sum of convex functions is a convex function and (equivalently) the sum of concave functions is a concave function. From this it follows that addition and subtraction of a function that is linear in all variables preserves convexity. The sum of a convex function and a concave function need be neither convex nor concave. However if $f(x)$ is any polynomial function of one real variable x, then the real line can be divided into a finite set of closed intervals on each of which $f(x)$ is convex or concave. E.g., $f(x) = x^4 - x^2$ is the sum of the convex function x^4 and the concave function $-x^2$ and is not itself either convex or concave over the entire real line, but it is convex on the intervals $(-\infty, -c]$ and $[c, \infty)$ and concave on the interval $[-c, c]$ where $c = \sqrt{1/6}$. For a function in several variables, even a polynomial function, there may be regions in which the function is neither convex nor concave. E.g., $f(x, y) = x^2 - y^2$ fails to be convex or concave in any open convex subset of the plane, since at each point it is convex in some directions and concave in others.

5.3. Current methods for analysis of control diagrams

There are three main graphical analysis techniques used in the analysis of systems in the frequency or complex plane: the Nyquist plot (complex plane), Bode diagrams (frequency domain) and Nichols plots (frequency domain). We will discuss in particular analysis using Nichols plots.

The *Nichols plot* [DB01, p.435] (e.g. see Fig. 6) (also known as a *Nichols chart*) plots the gain (in decibels) against the phase–shift of the output sinusoid as the frequency varies. Nichols plots often show *exclusion regions* that are used in the analysis of systems. Exclusion regions are used to represent various properties of a system, such as stability, 'good handling' and 'fast response', and differ depending on the precise requirements for the system being anaylsed.

A Nichols plot can be constructed by calculating the gain and phase–shift of a system F explicitly using Eqs. 3 and 4 and plotting the gain against the corresponding phase–shift

$$\text{gain} = 20\log_{10}(|F(j\omega)|) \tag{3}$$

$$\text{phase–shift} = \arg(F(j\omega)) = \begin{cases} \arctan\left(\frac{\Im(F(j\omega))}{\Re(F(j\omega))}\right) + k\pi & [\Re(F(j\omega)) \neq 0] \\ \pi/2 + k\pi & [\Re(F(j\omega)) = 0] \end{cases} \tag{4}$$

where F is the Laplace transform of a system; ω is frequency; k is some integer; $j = \sqrt{-1}$; and, if $z = x + jy = r(\cos\theta + j\sin\theta)$ is a complex number (where x, y, r and θ are real numbers with $r \geq 0$), then $\Re(z) = x$ and $\Im(z) = z$ are the real and imaginary parts of z, while $\arg(z) = \theta$ and $| z |= r$ are its argument and modulus.

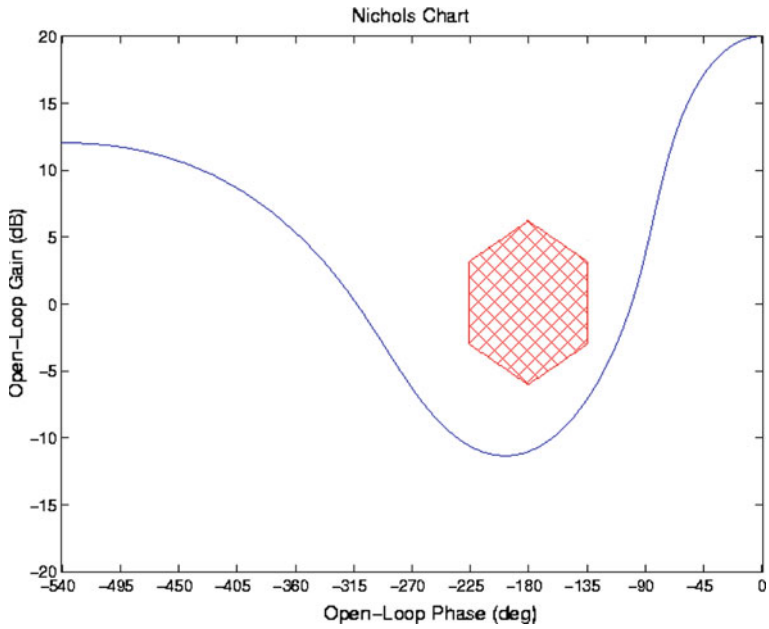


Fig. 6. An example Nichols plot showing an *exclusion region*(*shaded*)

Using a Nichols plot, we can then determine whether the system it represents has the desired properties of stability etc. This is traditionally done by checking by human inspection that the graph plotted does not cross into the exclusion zone. Note that this kind of property is readily expressed using quantifiers and so is a natural candidate for automation via theorem-proving. Our goal is to replace the human inspection process by an automated reasoning process that is more reliable, more cost-effective and more scaleable. We do not seek to verify the control theory that relates the Nichols plot to the physical system, since that is not a matter for significant concern in the overall engineering problem.

5.4. Decision procedure for positivity and negativity of functions

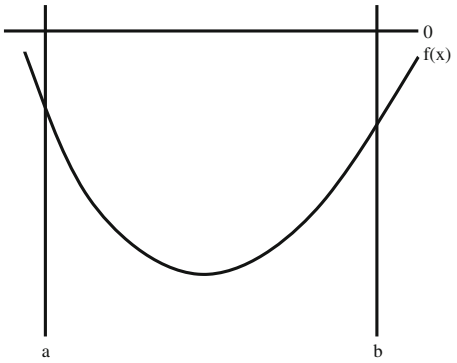
Many classical methods for control system analysis can be reduced to the problem of determining whether a given function is positive or negative in a given interval. This is the case for Nichols plot analysis as described above. We consider the plotted function and any exclusion zones in a piecewise manner, and can now present the basics of a decision procedure which determines this positivity or negativity. This decision procedure was proved correct using PVS.

5.4.1. Conditions for piecewise positivity and negativity

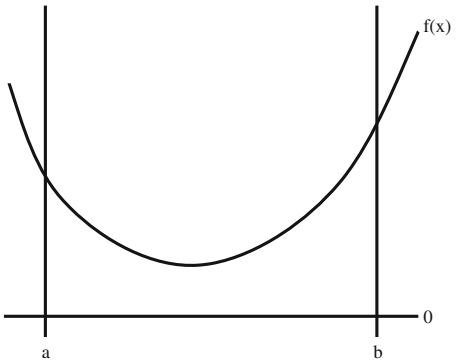
The following details a set of conditions that allow the positivity or negativity of a convex function $f : D \to \mathbf{R}$, where D is the closed interval $[a,\ b]$, to be determined. The conditions for a concave function are simply reflections of the conditions for a convex function and can be worked out by looking at $-f$ in the appropriate intervals.

Suppose the function f is continuously differentiable and convex on the closed convex set $D = [a,\ b]$, then:

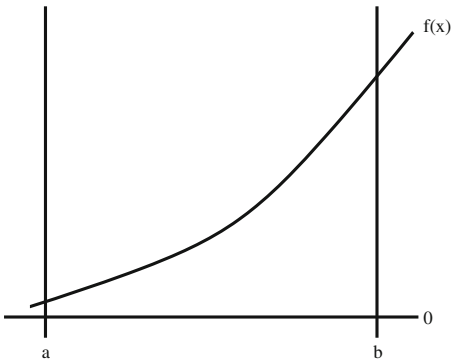
1. The function is negative on D iff the function is negative at the end points of D, i.e, $f(a) < 0$ and $f(b) < 0$



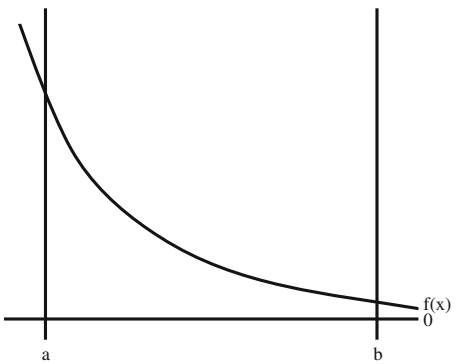
2. The function is positive on D iff one of the following two mutually exclusive conditions holds:



   (a) the gradient of the graph of f is equal to zero at some point within D and f is positive at that point, i.e, $\exists x \in D.\, f'(x) = 0 \wedge f(x) > 0$



   (b) the gradient of the graph of f does not equal zero at any point within D and the function is positive at the end points of D, i.e, $\forall x \in D.\, f'(x) \neq 0, f(a) > 0$, and $f(b) > 0$



If none of the above conditions hold for a convex function then there is at least one point within the interval at which the function is equal to zero. These conditions have been formalised in PVS [Har06] along with proof of correctness and coverage of the cases.

The conditions can be used to determine the positivity or negativity of an arbitrary reasonable function f. The domain of f can be split into intervals $[a_i, b_i]$ over which f is either convex or concave. By applying the set of conditions detailed here to f in each interval $[a_i, b_i]$, one can determine whether f is positive or negative on its domain.

5.4.2. Decision procedure

The decision procedure described in this section was developed to take those minimal isolated formulae (see [Har06], essentially normalised formulae with no nested quantifiers) in which all functions are finitely inflective and the scope of any quantifier is a literal formula in the language $\mathcal{L}_1$, and output the truth of the formula.[2]

The decision procedure takes a sentence ϕ in minimal isolated form, in which the scope of any quantifier is a literal formula in the language $\mathcal{L}_1$ and all functions f_i in ϕ are finitely inflective, and performs the following steps:

1. Convert existential quantification in ϕ to universal quantification giving ϕ'. This is a syntactic conversion to simplify the algorithm: $\exists x.P(x)$ becomes $\neg \forall x.\neg P(x)$
2. Take each quantified formula A in ϕ' containing a single literal $f_i \sim_i 0$ then determine the intervals D_{ij} of convexity and concavity for f_i.
3. For each of the intervals D_{ij} within $\mathbf{dom}(f_i)$ apply the appropriate case from the set of conditions given in Sect. 5.4.1. If the correct conditions hold for all these intervals then the i-th formula has the value TRUE. If the condition fails to hold for any interval then the formula has the value FALSE.
4. Construct the truth value for the sentence ϕ' (and thus ϕ) by applying the propositional operators within it to the truth values of Step 3.

The procedure presented in this section is applicable to functions of one variable that are finitely inflective, that is reasonable functions that have a finite number of intervals in which the curve is convex or concave. An extension to the decision procedure to handle functions of two variables has also been developed [Har06].

PVS was used to model the decision procedure and to verify its termination and soundness: the syntax and semantics of the language $\mathcal{L}_1$ were defined in PVS as was the decision procedure formulated as a function mapping $\mathcal{L}_1$ syntax to truth values; it was then proved that the truth value calculated by the decision procedure agrees with the semantics.

5.5. Nichols plot requirement verifier

In typical current practice, Nichols plots are created using tools such as Simulink or MATLAB [TM] and then inspected visually. This process successfully hides the mathematics from the engineers using the tools. When using a formal, symbolic approach, it is desirable that the mathematical details remain hidden from the engineer.

NRV (Nichols plot Requirements Verifier) was developed to allow the replacement of the informal visual analysis with formal symbolic analysis of Nichols plots. The tool is designed to fit into the development process and require no extra work to be performed by the control engineer. The tool requires the user to provide the transfer function representing the system and the Nichols plot requirements in terms of the boundary lines for the exclusion region. The tool applies the decision procedure of Sect. 5.4 and produces a formal proof that the system meets its requirements, a counter example to show that the system does not meet its requirements, or, if it can do neither of these, highlights the particular areas for which the process failed for closer examination by the control engineer.

[2] The decision procedure cannot be applied to formulae in which the scope of any quantifier is a compound expression, e.g. $\forall x.\ A(x) \vee B(x)$.

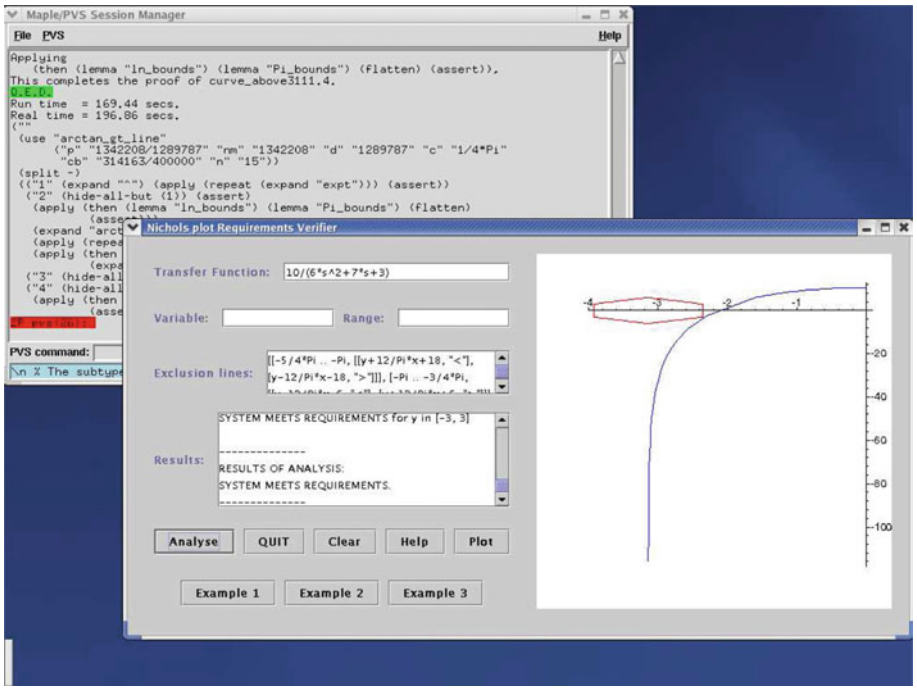


Fig. 7. The input/output window of NRV

5.5.1. Overview of NRV

The Nichols plot Requirements Verifier (NRV) is implemented in the Maple–QEPCAD–PVS system. Maple is used to perform the bulk of the calculations as well as providing the user interface. PVS is used to verify the results of Maple's calculations, using, when necessary, QEPCAD routines.

The minimum amount of data required to perform Nichols plot analysis is a representation of the system to be analysed, usually in the form of a transfer function, and some representation of the exclusion/desired region. The decision procedure of Sect. 5.4 can not be applied directly to this input, thus it is necessary to perform some pre-processing to correctly formulate the problem. This pre-processing requires both symbolic manipulation of the input and numerical calculation and is a task ideally suited to Maple.

The front end of NRV is implemented using Maple *Maplets*, a feature for providing graphical user interfaces similar to applets in Java. Figure 7 shows the interface for data entry and display of results and error messages. A simple type check mechanism ensures that the input is of the correct type and format. Maple processes the input to form the appropriate sentences for use in the decision procedure and invokes PVS to perform the required verification. During the verification PVS must show that various polynomial inequalities hold. To do this, PVS invokes QEPCAD routines within the proof. The PVS–QEPCAD interface takes the sequents and consequents from the current PVS sub-proof, formulates an equivalent sentence in the syntax of QEPCAD and invokes the QEPCAD routines. QEPCAD returns true if the sentence holds, false otherwise. The results of the QEPCAD routines are trusted by PVS. If PVS fails to provide proofs then attempts to find counter examples are made and the process continues. Maple records appropriate messages depending on the results of the PVS calls. Once the process is complete Maple displays the results of the decision procedure, the messages recorded as a result of PVS calls, and a plot showing the bounding lines for the specified region along with the plot of the curve representing the system. This allows the analyst to view the Nichols plot as classically used tools allow but also provides either assurances that the system meets or does not meet its requirements, or suggestions of areas that require closer examination.

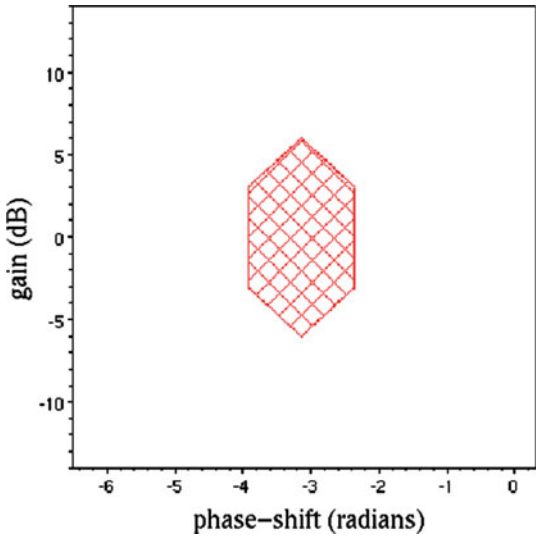


Fig. 8. Nichols plot showing a hexagonal exclusion region around $(-\pi, 0)$

NRV is fully automatic, relying on Maple procedures and high level PVS strategies to perform all the calculations and proofs. Thus the added complexity of the formal symbolic verification is hidden to the user of the tool.

5.6. Case studies

In general, a system is considered stable if its Nichols plot does not enter a certain hexagonal region about the point $(-\pi, 0)$ as shown in Fig. 8. This requirement can be expressed in terms of the lines bounding the region in particular intervals. The Nichols plot for the system must lie below the line between the points $(-\frac{5}{4}\pi, -3)$ and $(-\pi, -6)$ or above the line between the points $(-\frac{5}{4}\pi, 3)$ and $(-\pi, 6)$. It must lie below the line between the points $(-\pi, -6)$ and $(-\frac{3}{4}\pi, -3)$ or above the line between the points $(-\pi, 6)$ and $(-\frac{3}{4}\pi, 3)$. It must lie to the left of the line between the points $(-\frac{5}{4}\pi, -3)$ and $(-\frac{5}{4}\pi, 3)$ or to the right of the line between the points $(-\frac{3}{4}\pi, -3)$ and $(-\frac{3}{4}\pi, 3)$. This can then be entered into NRV together with the transfer function of the system being investigated. NRV can then attempt to prove automatically that the system meets its stability requirements as illustrated in the case study of Sect. 5.6.1 below. If the proof attempt fails, NRV lets us use PVS to investigate the cause of the failure, typically coming up with a counterexample showing that the Nichols plot criterion is not satisfied, as in some of the further examples discussed in Sect. 5.6.2.

5.6.1. Inverted pendulum

The inverted pendulum is a classic example from control engineering. An inverted pendulum is balanced on a cart (see Fig. 9 and Table 1); when a force F is applied to the cart the pendulum and the cart move.

There are two outputs of interest: the displacement of the cart x and the angle of the pendulum θ. When concerned only with the angle of the pendulum, the behaviour of the system can be represented using the following transfer function (see Table 1)

$$G_1 = \frac{mls}{(MI + Mml^2 + mI)s^3 + (bI + bml^2)s^2 - (Mmgl + m^2gl)s - bmgl}$$

where $G_1(s)$ is the Laplace transform of $\theta(F)$. The system can be modelled as the sequential combination (see Fig. 10) of a controller G_c and the system G_1. The controller G_c used in this example is a PID (Proportional/Inte-

gral/Derivative) controller, which is a commonly used form of controller

$$G_c = \frac{K_d s^2 + K_p s + K_i}{s}$$

designed to make the system produce the desired response.

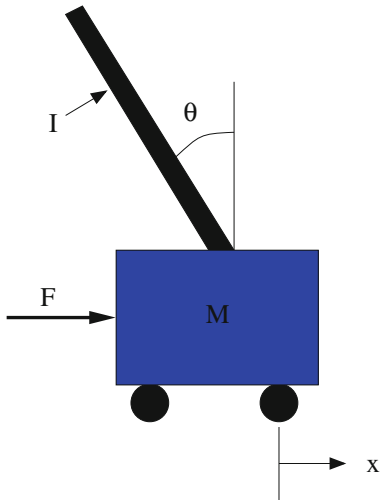


Fig. 9. Inverted pendulum

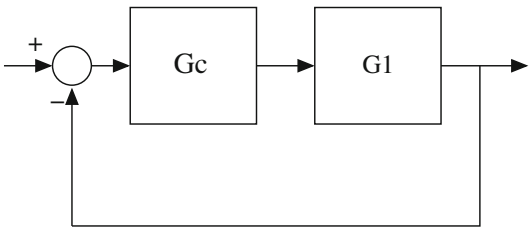


Fig. 10. Block diagram for an inverted pendulum system

Table 1 shows the values for the parameters of the system chosen for this example. The value of the mass of the pendulum m is left undecided.

In order to analyse the inverted pendulum system with regards to the Nichols plot requirements, one must provide NRV with the transfer function of the system along with the correct formalisation of the exclusion region as mentioned above.

Assuming that the mass of the pendulum is 0.2 kg, the open loop transfer function for the inverted pendulum system is

$$G = G_c G_1 = \frac{-25(2s^2 - 7s + 2)}{11s^3 + 2s^2 - 343s - 49}.$$

The Nichols plot for the system, showing the exclusion region is shown in Fig. 11. Note the difficulty of ensuring that the Nichols plot requirements are met by visual inspection.

Table 1. Values for parameters in an inverted pendulum system

Mass of cart	M	0.5 kg
Friction of the cart	b	0.1 N/m/s
Length to the pendulum's centre of mass	l	0.3 m
Inertia of the pendulum	I	0.006 kgm^2
Force of gravity	g	9.8 ms^2
Proportional coefficient	K_p	3.5
Integral coefficient	K_i	−1
Derivative coefficient	K_d	−1

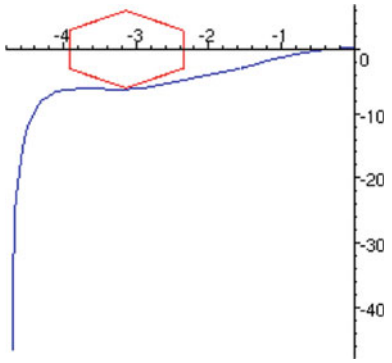


Fig. 11. Nichols plot for an inverted pendulum system

The bounds of the exclusion region can be represented as follows

$$y = \begin{cases} -\frac{12}{\pi}x - 18 & \text{for } x \in [-\frac{4}{5}\pi, -\pi] \\ \frac{12}{\pi}x + 18 & \text{for } x \in [-\frac{4}{5}\pi, -\pi] \\ \frac{12}{\pi}x + 6 & \text{for } x \in [-\pi, -\frac{3}{4}\pi] \\ -\frac{12}{\pi}x - 6 & \text{for } x \in [-\pi, -\frac{3}{4}\pi] \end{cases}$$

$$x = \begin{cases} -\frac{3}{4}\pi & \text{for } y \in [-3, 3] \\ -\frac{5}{4}\pi & \text{for } y \in [-3, 3] \end{cases}$$

In order to show that the system meets the specified requirements, i.e., that the curve does not enter the exclusion region, NRV must verify the following assertions for each point (x, y) on the curve:

$$-\tfrac{4}{5}\pi \le x \wedge x \le -\pi \Rightarrow y < -\tfrac{12}{\pi}x - 18 \vee \tfrac{12}{\pi}x + 18 < y$$
$$-\pi \le x \wedge x \le -\tfrac{3}{4}\pi \Rightarrow y < \tfrac{12}{\pi}x + 6 \vee -\tfrac{12}{\pi}x - 6 < y$$
$$-3 \le y \wedge y \le 3 \Rightarrow -\tfrac{3}{4}\pi < x \vee x < -\tfrac{5}{4}\pi$$

NRV considers the relationship between the curve and the bounds of the exclusion region in each of the three intervals.

1. NRV uses Maple to calculate that the interval $[-\frac{5}{4}\pi, -\pi]$, in terms of x, corresponds to the interval $[\frac{157}{128}, \frac{129}{32}]$, in terms of ω and uses PVS to show that

   $$\forall\omega.\frac{157}{128} \ge \omega \vee \omega \ge \frac{129}{32} \Rightarrow -\frac{5}{4}\pi \ge \arg(G) \vee \arg(G) \ge -\pi.$$

   Within this interval there is one point of inflection, which lies in the interval $[\frac{569}{256}, \frac{1139}{512}]$. The curve is convex for $\omega \in [\frac{157}{128}, \frac{569}{256}]$ and concave for $\omega \in [\frac{1139}{512}, \frac{129}{32}]$. NRV uses the conditions given in Sect. 5.4.1 to determine points at which the curve should be examined to ensure that it lies below the line $-\frac{12}{\pi}x - 18$ or above the line $\frac{12}{\pi}x + 18$. PVS proves that the curve lies below $-\frac{12}{\pi}x - 18$ at $\frac{157}{128}$, $\frac{569}{256}$ and $\frac{1139}{512}$, and thus that it lies outwith the exclusion region for $x \in [-\frac{5}{4}\pi, -\pi]$.
2. NRV uses Maple to calculate that the interval $[-\pi, -\frac{3}{4}\pi]$, in terms of x, corresponds to the interval $[\frac{57}{128}, \frac{629}{512}]$, in terms of ω and uses PVS to show that

   $$\forall\omega.\frac{57}{128} \ge \omega \vee \omega \ge \frac{629}{512} \Rightarrow -\pi \ge \arg(G) \vee \arg(G) \ge -\frac{3}{4}\pi.$$

Within this interval there are no points of inflection. The curve is convex for $\omega \in [\frac{57}{128}, \frac{629}{512}]$. NRV uses the conditions given in Sect. 5.4.1 to determine points at which the curve should be examined to ensure that it lies below the line $\frac{12}{\pi}x + 6$ or above the line $-\frac{12}{\pi}x - 6$. PVS proves that the curve lies below $\frac{12}{\pi}x + 6$ at $\frac{57}{128}$ and $\frac{629}{512}$, and thus that it lies outwith the exclusion region for $x \in [-\pi, -\frac{3}{4}\pi]$.

Given the nature of the exclusion region, the final condition:

$$-3 \leq y \wedge y \leq 3 \Rightarrow -\frac{3}{4}\pi < x \vee x < -\frac{5}{4}\pi,$$

could be excluded, as it is implied by the first two conditions. However, for completeness this condition is also proved. In order to prove this condition is met, rather than considering the curve represented by the set of parametric equations $x = \arg(G)$, $y = \text{gain}(G)$, one must consider the curve represented by the set of parametric equations $x = \text{gain}(G)$, $y = \arg(G)$.

3. NRV uses Maple to calculate that the interval $[-3, 3]$, in terms of y, corresponds to the interval $[0, \frac{101}{512}]$, in terms of ω and uses PVS to show that

$$\forall \omega.\omega \geq \frac{101}{512} \Rightarrow -3 \geq \text{gain}(G) \vee \text{gain}(G) \geq 3.$$

Within this interval there are no points of inflection. The curve is convex for $\omega \in [0, \frac{101}{512}]$ NRV uses the conditions given in Sect. 5.4.1 to determine points at which the curve should be examined to ensure that it lies below $-\frac{5}{4}\pi$ or above $-\frac{3}{4}\pi$. PVS proves that the curve lies above $-\frac{3}{4}\pi$ at $\frac{101}{512}$ and thus that it lies outwith the exclusion region for $y \in [-3, 3]$.

5.6.2. Further examples

The example of Sect. 5.6.1 is one of a series discussed in more detail in [Har06]. The second example is an analysis of an inverted pendulum that fails to meet its requirements. This example was just as in Sect. 5.6.1 except that the mass of the pendulum was reduced to 0.17. The new Nichols plot just crosses into the excluded hexagon at the lowest point. As before, the decision procedure of Sect. 5.6.1 tells us at what point we need to examine the Nichols plot, and would need to prove the properties given in 5.6.1, to show that the Nichols plot lay outside the exclusion region in the interval $[-5/4\pi, -\pi]$. NRV is unable to complete the proof, so PVS is used to evaluate the Nichols plot at this point, so that a counterexample is generated showing that in fact at this point the curve lies inside the exclusion region.

In a third example, NRV is applied to the second case study, in the case of the interval $[-\pi, -3/4\pi]$, to show again that the requirements are not met in this interval.

The fourth example concerns a case study of a disk drive reader. This is more sophisticated than the previous examples, in that one of the parameters lies in a numeric range, instead of being a numeric constant. This type of problem was suggested by our industry partners as it is difficult to analyse using classical Nichols plot techniques, as it is a three dimensional rather than a two dimensional problem. The classic approach is to generate a suite of Nichols plots for different values of the variable parameter within the range. If the system meets the requirements in all of these plots the assumption is made that the system meets them throughout the numeric range.

Our method used the extension of our decision procedure to functions of two variables, as described in Sect. 5.4.2 above. NRV showed that the Nichols plot lay outside the exclusion region for all values of the parameter in the numeric range.

5.7. Results

NRV exploits the symbolic computation provided by the computer algebra system Maple, the formal techniques provided by PVS and the quantifier elimination routines provided by QEPCAD. NRV is highly automated and provides a graphical user interface, similar in appearance to a java applet, which allows the user of the system to have no knowledge of the underlying decision procedure, formal methods or the Maple or PVS syntax.

NRV may fail to provide a proof or counter example automatically. In these cases, NRV attempts to produce useful feedback to the analyst indicating where the analysis failed and, more for the benefit of the tool developer, whether the failure was within a Maple computation or in a PVS proof. NRV indicates which, if any conditions were proven and which failed, and attempts to highlight areas on the Nichols Plot that may require closer inspec-

tion by the analyst. Several case studies have been performed [Har06] and NRV has produced promising results rarely failing to find proofs. The case studies also show that NRV easily performs analysis of control systems of up to degree 5.

- NRV is a fully working prototype of an industrial strength tool.
- NRV relies on existing interfaces Maple–PVS and QEPCAD–PVS to provide a user friendly, highly specialised tool.
- It uses some extensions to the real analysis library [Got00] in PVS.
- The algorithms used for the decision procedures in NRV have been proved correct in PVS.
- NRV easily handles systems of a size of industrial interest, successfully hiding both the theorem proving and the quantifier elimination behind a graphical user interface.
- Nearly all the infrastructure, including automation, for the mathematical specifications and proofs were already present in PVS from previous work on DITLU and Maple–PVS.
- NRV clearly demonstrates the added value of combining several systems.

6. Conclusions and future work

The work on SATS showed the practical utility of real number theorem proving in verifying safety-critical systems. However, this kind of verification activity is likely to remain the province of formal methods experts interacting directly with a theorem prover such as PVS.

Maple–PVS illustrates one approach to making theorem proving accessible to a wider audience, by extending the popular computer algebra system Maple to provide new, safer versions of regularly used functions, for example for solving differential equations. PVS was used to discharge side conditions that are not checked within Maple.

With NRV, the Maple–PVS interface is used together with QEPCAD–PVS to produce a highly specialist tool for analysing Nichols plots, something designers of control systems would normally do by visual inspection. The progression from our early work on DITLU, which had a very simplistic manual user interface, the sleek interface of NRV, with the advances of the theory implementation in PVS working behind the scenes is a clear advance in making theorem proving accessible to ordinary engineers.

In general, we may consider three different modes of using theorem proving:

Hidden Verification These tools are completely automatic with a user friendly front-end. NRV, presented for the first time in this paper, is an example of this.

Semi-exposed Verification With foundations and some automation already in existence, these tools require the user to apply some guidance when doing verification. Both the calls from DITLU and the basic calls from Maple using the Maple–PVS are in general example of this. Also, specialist packages may be developed, such as the spatial reasoning package suggested below. Often, the applications will have many similar lemmas, facilitating repeated application of the same strategies.

Exposed Verification Some specifications and their proofs are highly specialised, like those for SATS, and are basically one-off developments, even though some library development may follow from these.

It is clear from both NRV, the projects involving verification of air traffic management procedures and other industrial applications, such as Harrison's floating point verification [Har00] that the use of theorem proving for industrial applications is now appropriate. It is, however, also still the case that using higher order theorem provers in particular is very much a task for experts. This stresses the need for systems such as NRV with a limited application area permitting specialists tools to be developed. Whether such a specialist tool has a separate front end (as NRV does), or is simply a set of well-developed theories with appropriate automation in a theorem prover, is a separate issue. So of the three modes mentioned above, we might expect the first to be immediately usable by experts in the application area, even if they have little or no experience using formal methods. The second mode should be usable by someone with some but limited experience of formal verification, whereas the last mode is in general only feasible for experts in theorem proving, preferably in collaboration with experts in the application area.

Amongst users of theorem provers, it is generally recognised that the learning curve for using any one of the provers is quite steep. This of course means that for any new project one undertakes, one is likely to continue to use the prover one is most familiar with. However, as the use of higher order theorem provers increases, it is nonetheless interesting to compare the capabilities of different provers in various areas. As opposed to for first order theorem provers, where one may use TPTP as a standard for testing, there is no recognised test collection for

higher order theorem provers applied to problem domains such as real number theorem proving. Some progress on defining a suitable test suite has been made by one of us [Lig06], but much further work remains to be done.

Using a theorem prover as a back-end to another tool, such as Maple in the application NRV, provides a way to hide the specifics of the theorem prover from users less interested in the details. However, for more diverse verification problems, it is still necessary to work directly with the prover. One suggestion for how to construct convincing arguments of correctness for larger systems/problems is the "evidential tool bus", suggested by Rushby et al. [dMOR+05]. This provides an architecture for combining verification tools, with the main proviso that their output much be in such a form that evidence can be reproduced. Despite running under Maple, not a formal tool, NRV could be added to such a tool bus with the specific service of verifying certain properties of transfer functions, since only small modifications would be needed to allow NRV to return for example the PVS proof scripts used.

There are several interesting directions in which to continue the research described here.

Firstly, having seen how powerful the combination of tools in NRV is, we would like to see a move towards making such integration of tools easier. One of the biggest issues in this area has nothing to do with theorem proving, but rather is a systems issue: changing versions of the various systems involved makes it difficult to implement an interface which is forward compatible or even easy to update when new versions of the systems are released. The evidential tool bus mentioned above could alleviate these compatibility issues. With each system responsible for its own interface to the bus, this should be available with each new release.

There are a number of different systems we might imagine connected to such an evidential tool bus, for example:

Real Number Theorem Proving This should support basic verification over the reals, including numerics and handling instantiation.

Automation Specialised automation packages, perhaps including proof planning.

Specialist packages Like the suggested spatial reasoning package.

QEPCAD Quantifier elimination packages.

Visualisation Although not evidential, visualisation may help in understanding both problems and proofs.

In order to facilitate more general industry take-up of theorem proving, we know that automation and accessibility is essential. For the case of control law design, NRV allows for a fully automatic formal verification of something which until this point was "proved" by visual inspection. We suggest the development of a PVS package for *spatial reasoning* relevant not only to air traffic management, but also other areas concerned with reasoning about moving bodies. This package would encompass not only an implementation of relevant theories in both 2 and 3 dimensions, but also PVS strategies to support the automation of proofs within this area.

It would be tempting to try to generalise NRV to handle any curves, which can be plotted in Maple. However, such a generalisation may mean that the functions covered are too complicated to be handled by the current PVS strategies for continuity and the like. Of course, one can extend the strategies, but they currently work by the very simple "high school method", and so an extension is likely to mean a complete redesign.

In general, when working with real data, one does not have precise numbers, but works instead with values in intervals. There is a theory of interval arithmetic in PVS [ML05], but it does not currently support much automation. It would be very useful to have a good strategy package for interval arithmetic.

The idea of combining visual inspection and formal verification, as allowed by NRV, seems useful, particularly in fields where the current standard method of arguing correctness is by visual inspection. For those less familiar with formal verification, confidence in the formal proof is strengthened by the connection to the visual argument. For specific domains, it may be very useful to include tools for visualisation along with a PVS package containing theories and automation.

Acknowledgements

Our thanks are due to the many colleagues who have helped with our research and with the writing of this paper, especially Rob Arthan. Parts of the work were sponsored by QinetiQ and DSTL, and we are grateful to them for financial support, the involvement, patience and insights they brought to the research, and for their suggestion of Nicholls plots as an application. Support was also provided by EPSRC under grants EP/H500162, EP/F02309X and GR/S31242. We are grateful to the referees for their insightful and encouraging comments, which have been of great help in shaping the paper, and to the editor for his patience.

References

[Ada95] Adams RA (1995) Calculus: a complete course, 3rd edn. Addison-Wesley, Reading

[ADG+01] Adams A, Dunstan M, Gottliebsen H, Kelsey T, Martin U, Owre S (2001) Computer algebra meets automated theorem proving: integrating maple and PVS. In: Boulton RJ, Jackson PB (eds) 14th international conference on theorem proving in higherOrder logics. Lecture notes in computer science, vol 2152. Springer-Verlag, pp 27–42

[AGLM99a] Adams AA, Gottliebsen H, Linton SA, Martin U (1999) Automated theorem proving in support of computer algebra: symbolic definite integration as a case study. In: Dooley S (ed) ISSAC '99: proceedings of the 1999 international symposium on symbolic and algebraic computation. Vancouver, British Columbia, 1999. Simon Fraser University, ACM Press, pp 253–260

[AGLM99b] Adams AA, Gottliebsen H, Linton SA, Martin U (1999) VSDITLU: a verified symbolic definite integral table look-up. In: Ganzinger H (ed) Automated deduction—CADE-16. Lecture notes in artificial intelligence, vol 1632. Trento, Italy, 1999. ITC-irst, Springer-Verlag, pp 112–126

[AP08] Akbarpour B, Paulson LC (2008) Metitarski: an automatic prover for the elementary functions. In: Autexier S, Campbell J, Rubio J, Sorge V, Suzuki M, Wiedijk F (eds) AISC/MKM/Calculemus Lecture notes in computer science, vol 5144. Springer, pp 217–231

[Art64] Artin E (1964) The gamma function. Holt, Rinehart and Winston, Inc, New York

[BEH+03] Brown CW, Encarnacin MJ, Hong H, Johnson J, Werner Kr, Liska R, Mccallum S (2003) QEPCAD B: a program for computing with semi-algebraic sets using cads. SIGSAM Bull 37:108

[BHC95] Ballarin C, Homann K, Calmet J (1995) Theorems and algorithms: an interface between Isabelle and Maple. In: ISSAC, pp 150–157

[BKM01] Bosgra OH, Kwakernaak H, Meinsma G (2001) Design methods for control systems: notes for a course of the Dutch Institute of Systems and Control, Winter term 2001–2002. Department of Systems, Signals and Control, University of Twente

[BM06] Boldo S, Muñoz C (2006) A formalization of floating-point numbers in PVS. Report NIA Report No. 2006-01, NASA/CR-2006-214298, NIA-NASA Langley, National Institute of Aerospace, Hampton, VA

[But] Butler R NASA LaRC PVS libraries. http://shemesh.larc.nasa.gov/fm/larc/PVS-library/pvslib.html

[CGBK04] Carreño V, Gottliebsen H, Butler R, Kalvala S (2004) Formal modeling and analysis of a preliminary small aircraft transportation system (SATS) concept. Technical Report NASA/TM-2004-21, NASA Langley Research Center, NASA LaRC, Hampton VA 23681-2199, USA

[CM00] Carreño V, Muñoz C Aircraft trajectory modeling and alerting algorithm verification. In: Harrison and Aagaard [HA00], pp 90–105

[CM05] Carreño V, Muñoz C (2005) Safety verification of the Small Aircraft Transportation System concept of operations. In: Proceedings of the AIAA 5th aviation, technology, integration, and operations conference, AIAA-2005-7423. Arlington, Virginia

[DB01] Dorf RC, Bishop RH (2001) Modern control systems, 9th edn. Prentice-Hall, Englewood Cliffs

[dMOR+05] de Moura L, Owre S, Rue H, Rushby J, Shankar N (2005) Integrating verification components. http://www.csl.sri.com/cgi-bin/rushby/ps2pdf.pl?~rushby/papers/vstte05

[Dut96] Dutertre B (1996) Elements of mathematical analysis in PVS. In: von Wright J, Grundy J, Harrison J (eds) Theorem proving in higher order logics: 9th international conference. Lecture notes in computer science, vol. 1125. Springer-Verlag, pp 141–156

[Fle00] Fleuriot JD On the mechanization of real analysis in Isabelle/HOL. In: Harrison and Aagaard [HA00], pp 146–162

[For] SRI Formalware. Yices. http://yices.csl.sri.com/

[GHK+80] Gierz G, Hofmann KH, Keimel K, Lawson JD, Mislove M, Scott DS (1980) A compendium of continuous lattices (CCL). Springer-Verlag, Berlin

[Gon] Gonthier G A computer-checked proof of the Four Colour Theorem. http://research.microsoft.com/~gonthier/4colproof.pdf

[Got00] Gottliebsen H Transcendental functions and continuity checking in PVS. In: Harrison and Aagaard [HA00], pp 198–215

[Got01] Gottliebsen H (2001) Automated theorem proving for mathematics: real analysis in PVS. PhD thesis, University of St Andrews

[GS] Gottliebsen H, So CM The Maple–PVS interface. http://www.dcs.qmul.ac.uk/~hago/Maple-PVS/

[HA00] Harrison J, Aagaard M (eds) (2000) In: Theorem proving in higher order logics: 13th international conference, TPHOLs 2000. Lecture notes in computer science, vol 1869. Springer-Verlag

[Hal] Hales T The flyspeck project. http://www.math.pitt.edu/~thales/flyspeck/

[Har98] Harrison J (1998) Theorem proving with the real numbers. Springer-Verlag, Berlin

[Har00] Harrison J Formal verification of IA-64 division algorithms. In: Harrison and Aagaard [HA00], pp 234–251

[Har06] Hardy R (2006) Formal methods for control engineering: a validated decision procedure for Nichols plot analysis. PhD thesis, University of St Andrews

[Jac02] Jacobi C (2002) Formal verification of a fully IEEE compliant floating point unit. PhD thesis, University of the Saarland, 2002. http://engr.smu.edu/~seidel/research/diss-jacobi.ps.gz

[KS] Kemmerly GT, Syrett NE Small aircraft transportation system (SATS). http://sats.nasa.gov/main.html

[Lig06] Lightfoot O (2006) A real arithmetic test suite for theorem provers. In: 13th workshop on automated reasoning. ARW, pp 21–23

[Map12] Maplesoft (2012) The maple documentation center. http://www.maplesoft.com/documentation_center/

[Mar96] Marker D (1996) Model theory and exponentiation. Not Am Math Soc 43:753–759

[ML05] Muñoz C, Lester D (2005) Real number calculations and theorem proving. In: Hurd J, Melham T (eds) Proceedings of the 18th international conference on theorem proving in higher order logics, TPHOLs 2005. Lecture notes in computer science, vol 3603. Oxford, UK, 2005. Springer-Verlag, pp 195–210

[MLK98] Moore JS, Lynch TW, Kaufmann M (1998) A mechanically checked proof of the $AMD5_K86^{TM}$ floating point division program. IEEE Trans Comput 47(9):913–926

[MM] Muñoz C, Mayero M Real automation in the field. http://research.nianet.org/~munoz/Field/

[NT93] Nowak B, Trybulec A (1993) Hahn–Banach theorem. J Formaliz Math. http://markun.cs.shinshu-u.ac.jp/Mirror/mizar/JFM/Vol5/hahnban.html

[Oga97] Ogata K (1997) Modern control engineering, 3rd edn. Prentice-Hall, Englewood Cliffs

[ORS92] Owre S, Rushby JM, Shankar N (1992) PVS: a prototype verification system. In: Kapur D (ed) 11th international conference on automated deduction (CADE). Lecture notes in artificial intelligence, vol 607. Saratoga, NY, June 1992. Springer-Verlag, pp 748–752

[Pra00] Pratt RW (ed) (2000) Flight control systems: practical issues in design and implementation. IEE control engineering series, vol 57. The Institution of Electrical Engineers. Copublished by The American Institute of Aeronautics and Astronautics

[Rus99] Russinoff DM (1999) A mechanically checked proof of correctness of the AMD K5 floating point square root microcode. Form Methods Syst Des. 14(1):75–125

[Spi73] Spivak M (1973) Calculus. Addison-Wesley, Reading

[TM] Inc. The Mathworks. MATLAB and Simulink. http://www.mathworks.com/

[vBJ77] van Benthem Jutting LS (1977) Checking Landau's "Grundlagen" in the AUTOMATH system. PhD thesis, Eindhoven University of Technology

[Vit03] Di Vito BL (2003) Strategy-enhanced interactive proving and arithmetic simplification for PVS. In: 1st international workshop on design and application of strategies/tactics in higher order logics (STRATA 2003). Rome, Italy

Received 9 May 2007

Accepted in revised form 23 April 2012 by David Duce and Jim Woodcock

Published online 8 June 2012