HAL
open science

Guaranteed Proofs Using Interval Arithmetic

Marc Daumas, Guillaume Melquiond, César Muñoz

To cite this version:

Marc Daumas, Guillaume Melquiond, César Muñoz. Guaranteed Proofs Using Interval Arithmetic. 17th IEEE Symposium on Computer Arithmetic, 2005, Cape Cod, Massachusetts, United States. pp.188-195, 10.1109/ARITH.2005.25 . hal-00164621

HAL Id: hal-00164621

https://hal.science/hal-00164621v1

Submitted on 22 Jul 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Guaranteed Proofs Using Interval Arithmetic

Marc Daumas, Guillaume Melquiond, and César Muñoz*

Abstract

This paper presents a set of tools for mechanical reasoning of numerical bounds using interval arithmetic. The tools implement two techniques for reducing decorrelation: interval splitting and Taylor's series expansions. Although the tools are designed for the proof assistant system PVS, expertise on PVS is not required. The ultimate goal of the tools is to provide guaranteed proofs of numerical properties with a minimal human-theorem prover interaction.

1 Introduction

Deadly and disastrous failures [4, 7, 12] confirm the shared belief that the traditional peer-review process is not sufficient to guarantee correctness of published proofs and software [11]. Despite this belief, mechanical theorem provers and proof assistants are not widely used in the applied mathematics community. Part of the problem is the lack of user friendly interfaces that results on steep learning curves. This paper presents a set of tools for mechanical reasoning of numerical bounds using interval arithmetic. The goal is to provide guaranteed formal proofs of numerical properties with a minimum effort.

Interval arithmetic has been used for decades as a standard tool for numerical analysis on engineering applications [8, 15]. In interval arithmetic, operations are evaluated on range of numbers rather than on real numbers. Formally, an *interval* $\mathbf{x} = [a, b]$ is the set of real numbers between a and b, i.e.,

$$[a, b] \quad = \quad \{x \mid a \leq x \leq b\}.$$

The bounds a and b are called the *lower bound* and *upper bound* of $\mathbf{x}$, respectively. In this paper, we are interested in *rational* interval arithmetic, i.e., the bounds a and b are assumed to be rational numbers. In the following, we use the first letters of the alphabet $a, b, \ldots$ to denote rational numbers, and the last letters of the alphabet $\ldots x, y, z$ to denote arbitrary real variables. We use boldface for interval variables. Furthermore, if $\mathbf{x}$ is an interval variable, $\underline{\mathbf{x}}$ denotes its lower bound and $\overline{\mathbf{x}}$ denotes its upper bound. By abuse of notation, and when it is clear from the context, a rational number a is identified with the interval $[a, a]$.

The four basic operations in interval arithmetic are defined such that they satisfy the *inclusion property*:

$$\mathbf{x} \otimes \mathbf{y} \quad = \quad \{x \otimes y \mid x \in \mathbf{x},\ y \in \mathbf{y}\},$$

where $\otimes = \{+, -, \times, \div\}$.[1] This property is fundamental to interval arithmetic. It guarantees that the evaluation of an expression using interval arithmetic is a correct approximation of the exact real value.

Interval arithmetic is *sub-distributive*, i.e., $\mathbf{x} \times (\mathbf{y} + \mathbf{z}) \subseteq \mathbf{x} \times \mathbf{y} + \mathbf{x} \times \mathbf{z}$. In the general case, the inclusion is strict. This may have surprising effects, for instance $\mathbf{x} - \mathbf{x}$ is, in general, different from the interval 0, e.g., $[0, 1] - [0, 1] = [-1, 1]$. This effect is also called *decorrelation* and it is due to the fact that interval identity is lost in interval arithmetic.

Consider the function $f(x) = x \times (1 - x)$. A simple analysis reveals that f reaches its maximum at $x = \frac{1}{2}$ with $f(\frac{1}{2}) = \frac{1}{4}$. If $x \in [0, 1]$, the minimums are reached at the bounds with $f(0) = f(1) = 0$. Hence,

$$\forall x \in [0, 1] :\ x \times (1 - x) \in \left[0, \frac{1}{4}\right].$$

On the other hand, the interval expression $\mathbf{x} \times (1 - \mathbf{x})$, where $\mathbf{x} = [0, 1]$, evaluates to $[0, 1]$. The inclusion property guarantees that $[0, 1]$ is a correct approximation of $f(x)$, for $x \in \mathbf{x}$. However, as this example shows, it may not be the best one:

$$\left[0, \frac{1}{4}\right] \quad \subsetneq \quad [0, 1].$$

*M. Daumas (Marc.Daumas@ens-lyon.fr) and G. Melquiond (Guillaume.Melquiond@ens-lyon.fr) are with the LIP Computer Science Laboratory, UMR 5668 CNRS–ENS Lyon–INRIA, France. César Muñoz (munoz@nianet.org) is with the National Institute of Aerospace, 144 Research Drive, Hampton, VA, USA. This work was partially supported by the National Aeronautics and Space Administration under NASA Cooperative Agreement NCC-1-02043, by the French National Center for Scientific Research under CNRS PICS grant 2533, and by the Department of the Development and the Department of the Industrial Relations (DirDRI) of the INRIA.

[1] In the case of division, it is assumed that $0 \notin \mathbf{y}$.

Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'05)
1063-6889/05 $20.00 © 2005 IEEE

There are a few techniques to reduce the effect of decorrelation. They usually require arithmetic manipulations of the interval expressions and a fine analysis of the interval variables. For example, an expert will probably note that $f(x)$ is equivalent to $\frac{1}{4} - \left(x - \frac{1}{2}\right)^2$, and that the corresponding interval expression $\frac{1}{4} - \left(\mathbf{x} - \frac{1}{2}\right)^2$ does not suffer from decorrelation. The proof that a particular numerical expression satisfies some given bounds is not necessarily difficult, but it can be cumbersome, tedious, and, definitively, error prone.

This paper presents a set of tools that support mechanical proof checking of numerical bounds using interval arithmetic. The tools implement two techniques to reduce decorrelation. The first technique is based on interval splitting. The second technique is based on Taylor's series expansions. The tools are designed for the verification system PVS,[2] which is developed by SRI International [16]. However, minimal PVS expertise is required to use our tools as most of the technical burden of proving properties in a proof assistant system is hidden from the user. A C++ library generates proof obligations and proof scripts, in the form of PVS files, for a given numerical problem. The files are processed by PVS in batch mode and a summary of the status of the proofs is printed. User interaction with the theorem prover is minimized. This approach is usually referred as *invisible formal methods* [21].

The rest of this paper is organized as follows. Section 2 presents an overview of the PVS interval arithmetic library. The usage of the library is illustrated with the toy example $x \times (1 - x) \in [0, 1]$ for $x \in [0, 1]$. This example is reused in Section 3 to show how the precision can be improved by a simple interval splitting technique. Section 4 motivates a Taylor's series expansion technique with an example taken from a critical aeronautical application. That technique guarantees that an implemented polynomial approximation is close to about one unit in the last place (ulp) of the exact transcendental function. The relative error is exactly bounded by 1.36×10^{-6}. Section 5 presents the C++ library that generates the proof obligations and proof scripts of the examples in sections 3 and 4.

2 A PVS Library for Interval Arithmetic

The Prototype Verification System (PVS) [16] is a mechanical proof checker that provides a strongly typed specification language and a theorem prover for higher-order logic. PVS developments are organized in theories. A theory is a collection of mathematical and logical objects such as function definitions, variable declarations, axioms, and lemmas.

[2]PVS is available from `http://pvs.csl.sri.com`.

The library Interval is a set of PVS theories defining rational interval arithmetic. The library provides a set of proof strategies that automate interval reasoning, specially with respect to decorrelation effects.

Listing 1 shows a few definitions from the PVS theory `Interval`. Dots are used to simplify the presentation and hide some technical parts. Comments start with the symbol `%` and extend to the end of the line. The theory defines the type `Interval` and mathematical variables `x,y` of type `real` (real numbers), `X,Y` of type `Interval`, and `n` of type `nat` (natural numbers).

Intervals are stored as pairs of real numbers `[|x,y|]`. For instance, the PVS object `[|0,1|]` represents the interval $[0, 1]$. If `X` is a PVS interval, `lb(X)` is the lower bound and `ub(X)` is the upper bound of `X`. The propositions $x \in \mathbf{x}$ and $\mathbf{x} \subseteq \mathbf{y}$ are written `x ## X` and `X << Y`, respectively. Furthermore, the proposition `X > x` states that all values in `X` are strictly greater than `x`; similarly for `X ≥ x`, `X < x`, and `X ≤ x`.

The four basic interval operations are defined as in [9]:

$$
\begin{aligned}
\mathbf{x} + \mathbf{y} &= [\underline{\mathbf{x}} + \underline{\mathbf{y}}, \overline{\mathbf{x}} + \overline{\mathbf{y}}], \\
\mathbf{x} - \mathbf{y} &= [\underline{\mathbf{x}} - \overline{\mathbf{y}}, \overline{\mathbf{x}} - \underline{\mathbf{y}}], \\
\mathbf{x} \times \mathbf{y} &= [\min\{\underline{\mathbf{x}}\underline{\mathbf{y}}, \underline{\mathbf{x}}\overline{\mathbf{y}}, \overline{\mathbf{x}}\underline{\mathbf{y}}, \overline{\mathbf{x}}\overline{\mathbf{y}}\}, \\
&\qquad \max\{\underline{\mathbf{x}}\underline{\mathbf{y}}, \underline{\mathbf{x}}\overline{\mathbf{y}}, \overline{\mathbf{x}}\underline{\mathbf{y}}, \overline{\mathbf{x}}\overline{\mathbf{y}}\}], \\
\mathbf{x} \div \mathbf{y} &= \mathbf{x} \times \left[\frac{1}{\overline{\mathbf{y}}}, \frac{1}{\underline{\mathbf{y}}}\right], \quad \text{if } \underline{\mathbf{y}}\overline{\mathbf{y}} > 0.
\end{aligned}
$$

We also define negative, absolute value, square, and power functions as follows:

$$
\begin{aligned}
-\mathbf{x} &= [-\overline{\mathbf{x}}, -\underline{\mathbf{x}}], \\
|\mathbf{x}| &= [\min\{|\underline{\mathbf{x}}|, |\overline{\mathbf{x}}|\}, \max\{|\underline{\mathbf{x}}|, |\overline{\mathbf{x}}|\}], \quad \text{if } \underline{\mathbf{x}}\overline{\mathbf{x}} \geq 0. \\
|\mathbf{x}| &= [0, \max\{|\underline{\mathbf{x}}|, |\overline{\mathbf{x}}|\}], \quad \text{if } \underline{\mathbf{x}}\overline{\mathbf{x}} < 0. \\
\mathbf{x}^n &= \begin{cases} [1] & \text{if } n = 0, \\ [\underline{\mathbf{x}}^n, \overline{\mathbf{x}}^n] & \text{if } \underline{\mathbf{x}} \geq 0 \text{ or odd?}(n), \\ [\overline{\mathbf{x}}^n, \underline{\mathbf{x}}^n] & \text{if } \overline{\mathbf{x}} \leq 0 \text{ and even?}(n), \\ [0, \max\{\underline{\mathbf{x}}^n, \overline{\mathbf{x}}^n\}] & \text{otherwise.} \end{cases}
\end{aligned}
$$

Interval union $\mathbf{x} \cup \mathbf{y}$, written in PVS `X U Y`, is defined as the smallest rational interval that contains both $\mathbf{x}$ and $\mathbf{y}$.

All these operations are defined such that they satisfy the inclusion property. Indeed, the lemmas shown in Listing 2 are, among many others, provided by the library and formally verified. Free variables are implicitly quantified universally. As we will see, these properties are the basis of the automated support for interval reasoning.

Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'05)
1063-6889/05 $20.00 © 2005 **IEEE**

PVS Listing 1 Basic definitions

```
% interval.pvs
Interval : THEORY
BEGIN

  Interval : TYPE = ...

  x,y : VAR real
  X,Y : VAR Interval
  n   : VAR nat

  +(X,Y): Interval = [|lb(X)+lb(Y),
                       ub(X)+ub(Y)|]
  -(X,Y): Interval = [|lb(X)-ub(Y),
                       ub(X)-lb(Y)|]
  -(X)  : Interval = [|-ub(X),
                       -lb(X))|]
  *(X,Y): Interval = ...
  /(X,Y): Interval = X * [|1/ub(Y),
                           1/lb(Y)|]
  Abs(X): Interval = ...
  Sq(X) : Interval = ...
  ^(X,n): Interval = ...

  U(X,Y) : Interval = [|min(lb(X),lb(Y)),
                        max(ub(X),ub(Y))|]

  ...
```

PVS Listing 2 Inclusion properties (I)

```
  ...

  Add_sharp : LEMMA
    x ## X AND y ## Y ⟹ x+y ## X+Y

  Sub_sharp : LEMMA
    x ## X AND y ## Y ⟹ x-y ## X-Y

  Neg_sharp : LEMMA
    x ## X ⟹ -x ## -X

  Mult_sharp : LEMMA
    x ## X AND y ## Y ⟹ x*y ## X*Y

  Zeroless?(X): bool = X > 0 OR X < 0

  Div_sharp : LEMMA
    Zeroless?(Y) AND
    x ## X AND y ## Y ⟹ x/y ## X/Y

  Abs_sharp : LEMMA
    x ## X ⟹ abs(x) ## abs(X)

  Sq_sharp : LEMMA
    x ## X ⟹ sq(x) ## sq(X)

  Pow_sharp : LEMMA
    x ## X ⟹ x^n ## X^n

END Interval
```


1063-6889/05 $20.00 © 2005 **IEEE**

The square root and the trigonometric functions are implemented by approximation series. A PVS library of approximations was originally developed by one of the authors for the verification of an algorithm for aircraft conflict detection [14]. It was completed and extended with logarithm, exponential and arc tangent functions by David Lester. The approximation library is part of the NASA Langley PVS libraries[3].

The basic idea is to provide for each real function $f : \mathbb{R} \to \mathbb{R}$, parametric algebraic functions $\underline{f} : (\mathbb{R}, \mathbb{N}) \to \mathbb{R}$ and $\overline{f} : (\mathbb{R}, \mathbb{N}) \to \mathbb{R}$, such that for all x, n

$$\underline{f}(x,n) \leq f(x) \leq \overline{f}(x,n), \quad (1)$$

$$\underline{f}(x,n) \leq \underline{f}(x,n+1), \quad (2)$$

$$\overline{f}(x,n+1) \leq \overline{f}(x,n), \quad (3)$$

$$\lim_{n\to\infty} \underline{f}(x,n) = f(x) = \lim_{n\to\infty} \overline{f}(x,n). \quad (4)$$

Formula (1) states that $\underline{f}$ and $\overline{f}$ are, respectively, lower and upper bounds of f, and formulas (2), (3), and (4) state that these bounds can be improved, as much as needed, by increasing the approximation parameter n. Furthermore, we require that $\underline{f}$ and $\overline{f}$ are closed for rational numbers.

For each $\overline{f}$, the corresponding parametric interval function $\mathbf{f}$ is defined as follows:

$$\mathbf{f}(\mathbf{x},n) = [\underline{f}(\underline{\mathbf{x}},n), \overline{f}(\overline{\mathbf{x}},n)], \text{ if } f \text{ is increasing,}$$

$$\mathbf{f}(\mathbf{x},n) = [\underline{f}(\overline{\mathbf{x}},n), \overline{f}(\underline{\mathbf{x}},n)], \text{ if } f \text{ is decreasing.}$$

If f is neither increasing nor decreasing, as in the case of trigonometric functions, $\mathbf{f}$ is defined by case analysis on subintervals that are increasing or decreasing. The parameter n sets the accuracy of the approximations. In a real numerical problem, this parameter is set in advance by an external program that explores in an efficient way a vast number of values before deciding for the best one. This is one of the functionalities of the C++ library presented in Section 5.

The definitions of the square root and trigonometric interval operations satisfy the inclusion properties in Listing 3. Appropriate preconditions such as `X ≥ 0` and `Tan?(X)` guarantee that the operations `sqrt` and `tan` are well-defined.

Three basic strategies are provided by Interval: `sharp`, `instint`, and `joint`.

[3]Available from `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html`.

PVS Listing 3 Inclusion properties (II)

```
Sqrt_sharp : LEMMA
  X ≥ 0 AND
  x ## X ⟹  sqrt(x) ## Sqrt(X,n)

Sin_sharp : LEMMA
   x ## X ⟹  sin(x) ## Sin(X,n)

Cos_sharp : LEMMA
   x ## X ⟹  cos(x) ## Cos(X,n)

Tan_sharp : LEMMA
  Tan?(X) AND
  x ## X  ⟹  tan(x) ## Tan(X,n)
```

Let $e(x_1, \ldots, x_n)$ be a real expression with variables $x_1, \ldots, x_n$, and $\mathbf{e}(\mathbf{x_1}, \ldots, \mathbf{x_n})$ be the interval expression corresponding to e (for a pre-determined approximation parameter), where $x_i \in \mathbf{x_i}$, for $1 \leq i \leq n$.

- The proof rule `sharp` automatically discharges goals of the form

$$\frac{x_1 \in \mathbf{x_1}, \ldots, x_n \in \mathbf{x_n}}{e(x_1, \ldots, x_n) \in \mathbf{e}(\mathbf{x_1}, \ldots, \mathbf{x_n})},$$

  using the inclusion lemmas in Listing 2 and Listing 3.

- Let $\mathbf{y}$ be a rational interval. The proof rule `instint` automatically discharges goals of the form

$$\frac{x_1 \in \mathbf{x_1}, \ldots, x_n \in \mathbf{x_n}}{e(x_1, \ldots, x_n) \in \mathbf{y}},$$

  by showing that $e(x_1, \ldots, x_n) \in \mathbf{e}(\mathbf{x_1}, \ldots, \mathbf{x_n})$ using `sharp`, and then evaluating the numerical interval expression

$$\mathbf{e}(\mathbf{x_1}, \ldots, \mathbf{x_n}) \subseteq \mathbf{y}.$$

- The proof rule `joint` automatically discharges goals of the form

$$\frac{x_1 \in \mathbf{x_1}, \ldots, x_n \in \mathbf{x_n}}{e(x_1, \ldots, x_n) \in \mathbf{y}},$$

  by showing that

$$\frac{x_1 \in \mathbf{x_1}, \ldots, x_i \in \mathbf{x_i}', \ldots, x_n \in \mathbf{x_n}}{e(x_1, \ldots, x_n) \in \mathbf{y}},$$

  and

$$\frac{x_1 \in \mathbf{x_1}, \ldots, x_i \in \mathbf{x_i}'', \ldots x_n \in \mathbf{x_n}}{e(x_1, \ldots, x_n) \in \mathbf{y}},$$

  provided that (1) $\mathbf{x_i} = \mathbf{x_i}' \cup \mathbf{x_i}''$ and (2) $\mathbf{x_i}'$ and $\mathbf{x_i}''$ overlap. Appropriate parameters tell the strategy how to select x_i, $\mathbf{x_i}'$, and $\mathbf{x_i}''$.

1063-6889/05 $20.00 © 2005 **IEEE**

The file `fair.pvs`, in Listing 4, includes the lemma `fair_approx` that states

$$\forall x \in [0,1] : \; x \times (1-x) \in [0,1].$$

PVS Listing 4 Toy example

```
% fair.pvs
fair : THEORY
BEGIN

  fair_approx : LEMMA
    FORALL (x:real):
      x ## [|0,1|] IMPLIES
      x*(1-x) ## [|0,1|]

  %|- fair_approx : PROOF (instint) QED

END fair
```

The PVS theorem prover is generally used in interactive mode. However, batch proving is supported in PVS by the package ProofLite.[4] Proof scripts are written as comments using the special comment symbol `%|-`. In this case, the interval proof strategy `instint` is associated to the lemma `fair_approx`. The file `fair.pvs` is proof checked in batch mode with the command `proveit`:

```
$ proveit -package Interval fair.pvs
```

After a few seconds, the following message is displayed:

```
Theory totals:
1 formulas, 1 attempted, 1 succeeded
```

3 Sharper Bounds by Interval Splitting

Lemma `fair_approx` of Section 2.4 is very inaccurate as it bounds $x \times (1-x)$, with $x \in [0,1]$, by $[0,1]$ instead of $\left[0, \frac{1}{4}\right]$. This is due to decorrelation on x. In many cases, the easiest way to reduce decorrelation is to divide the input interval in many subintervals and to evaluate the expression on these subintervals separately.

For example, the interval $[0,1]$ could be evenly divided into 16 intervals $\left[\frac{i}{16}, \frac{i+1}{16}\right]$. Each arithmetic evaluation is a subset of $\left[0, \frac{9}{32}\right]$ that is a little larger than the optimal $\left[0, \frac{1}{4}\right]$.

The C++ library presented in Section 5 finds the same degree of accuracy with only 8 intervals:

$$\left[0, \frac{1}{4}\right], \left[\frac{1}{4}, \frac{3}{8}\right], \left[\frac{6}{16}, \frac{7}{16}\right], \left[\frac{7}{16}, \frac{8}{16}\right],$$

$$\left[\frac{8}{16}, \frac{9}{16}\right], \left[\frac{9}{16}, \frac{10}{16}\right], \left[\frac{5}{8}, \frac{3}{4}\right], \left[\frac{3}{4}, 1\right].$$

Indeed, the PVS file `toy.pvs` in Listing 5 is automatically generated from the original problem and some extra parameters. In this case, 16 lemmas are necessary to guarantee the required accuracy $\left[0, \frac{9}{32}\right]$. Lemmas `ToyI0` and `ToyI1` prove the case of the subintervals $\left[0, \frac{1}{4}\right]$ and $\left[\frac{1}{4}, \frac{3}{8}\right]$, respectively, using the proof rule `instint`. Lemma `ToyC1` proves the case of the subinterval $\left[0, \frac{3}{8}\right]$ by using the proof rule `joint`. The strategy shows that $\left[0, \frac{3}{8}\right] = \left[0, \frac{1}{4}\right] \cup \left[\frac{1}{4}, \frac{3}{8}\right]$, and then applies lemmas `ToyI0` and `ToyI1`. The final lemma `ToyC7` proves the result on the whole interval $[0,1]$.

PVS Listing 5 Toy example (revisited)

```
% toy.pvs
toy : THEORY
BEGIN

  x :VAR  real

  ToyI0 : LEMMA
    x ## [|0,4/16|] IMPLIES
    x * (1 - x) ## [|0,9/32|]
  %|- ToyI0 : PROOF (instint) QED

  ToyI1 : LEMMA
    x ## [|4/16,6/16|] IMPLIES
    x * (1 - x) ## [|0,9/32|]
  %|- ToyI1 : PROOF (instint) QED

  ToyC1 : LEMMA
    x ## [|0,6/16|] IMPLIES
    x * (1 - x) ## [|0,9/32|]
  %|- ToyC1 : PROOF
  %|-   (joint "ToyI0" "ToyI1")
  %|- QED

  ...

  ToyC7 : LEMMA
    x ## [|0,16/16|] IMPLIES
    x * (1 - x) ## [|0,9/32|]
  %|- ToyC7 : PROOF
  %|-   (joint "ToyC6" "ToyI7")
  %|- QED

END toy
```

Proof checking the file `toy.pvs` in batch mode reports:

```
$ proveit -package Interval toy.pvs
Theory totals:
16 formulas, 16 attempted, 16 succeeded
```

[4] Available from `http://research.nianet.org/~munoz/ProofLite`.

1063-6889/05 $20.00 © 2005 IEEE

4 Taylor's Series Expansions

Taylor's Theorem yields the following rule on interval arithmetic. Let $x, a \in \mathbf{x}$ and $\mathbf{x}_0, \ldots, \mathbf{x}_n$ be a list of intervals,

$$\frac{\frac{d^i f}{dx^i}(a) \in \mathbf{x}_i, \text{for } 0 \leq i < n, \quad \text{and} \quad \forall y \in \mathbf{x} : \frac{d^n f}{dx^n}(y) \in \mathbf{x}_n}{f(x) \in \Sigma_{i=0}^{n}(\mathbf{x}_i \times (\mathbf{x} - a)^i)/i!}$$

This rule is implemented by the strategy `taylor` in the library Interval. In this section, we show how this rule may be used to reduce decorrelation in a real example.

The function

$$r(\phi) = \frac{a}{1 + (1 - f)^2 \tan^2 \phi},$$

where a and f are constants defined by WGS84,[5] appears in the implementation of aircraft navigation algorithms. Indeed, $r(\phi)$ is used to translate aircraft geodesic coordinates, as calculated by global positioning systems, to Cartesian coordinates used, for example, by geometric conflict detection and resolution algorithms [3].

For efficiency reasons, one may want to approximate the function $r(\phi)$ by polynomial

$$\begin{aligned}\hat{r}(\phi) \quad = \quad & \frac{4439091}{4} + (\phi_m^2 - \phi^2)\times \\ & \left(\frac{9023647}{4} + (\phi_m^2 - \phi^2)\times \right. \\ & \left(\frac{13868737}{64} + (\phi_m^2 - \phi^2)\times \right. \\ & \left(\frac{13233647}{2048} + (\phi_m^2 - \phi^2)\times \right. \\ & \left(\frac{-1898597}{16384} + (\phi_m^2 - \phi^2)\times \right. \\ & \left.\left.\left.\left.\frac{-6661427}{131072}\right)\right)\right)\right),\end{aligned}$$

where $\phi_m = 715/512 \gtrapprox 80\pi/180$ and $\phi \in [0, \phi_m]$, as the latitude is assumed to be between 0 and 80°.

The coefficients of the polynomial approximation and ϕ_m^2 are stored exactly using IEEE single precision. Thus, the objective is to show that $\frac{e(\phi)}{r(\phi)}$, where

$$e(\phi) = r(\phi) - \hat{r}(\phi),$$

is bounded by 1.36×10^{-6}, i.e., about an ulp of the exact value.

Let $\mathbf{r}(\Phi)$, $\hat{\mathbf{r}}(\Phi)$, and $\mathbf{e}(\Phi)$ be the interval expressions corresponding to $r(\phi)$, $\hat{r}(\phi)$, and $e(\phi)$, respectively, and Φ be an interval such that $\phi \in \Phi \subseteq [0, \frac{715}{512}]$. Decorrelation on $\mathbf{e}(\Phi)$ yields that for any interval Φ, $\mathbf{e}(\Phi)$ is wider than the sum of the width of intervals $\mathbf{r}(\Phi)$ and $\hat{\mathbf{r}}(\Phi)$. Therefore, the splitting technique presented in the previous section would require more than ϕ_m/ulp subintervals to verify that $\mathbf{e}(\Phi)$ is no wider than one ulp. No proof assistant can guarantee such a large number of lemmas in a reasonable time.

To reduce decorrelation, we use Taylor's series expansion with $x = \phi$, $n = 1$, a equal to the midpoint of Φ, $\mathbf{x} = \Phi$, $\mathbf{x}_0 = \mathbf{e}([a])$, and $\mathbf{x}_1 = \mathbf{e}'(\Phi)$, where $\mathbf{e}'$ is the interval function corresponding to the first derivative of e. We get

$$e(\phi) \quad \in \quad \mathbf{e}(a) + (\Phi - a)\mathbf{e}'(\Phi), \tag{5}$$

as both (1) $e(a) \in \mathbf{e}([a])$ and (2) $\forall y \in \mathbf{\Phi} : e'(y) \in \mathbf{e}'(\Phi)$ trivially hold. Decorrelation on Formula (5) is reduced to first order with respect to the original $\mathbf{e}(\Phi)$.

Further reduction of decorrelation could be obtained by increasing the order of the Taylor series expansion, e.g., the second order expansion yields

$$e(\phi) \quad \in \quad \mathbf{e}(a) + (\Phi - a)\mathbf{e}'(a) + \frac{(\Phi - a)^2}{2}\mathbf{e}''(\Phi).$$

However, note that $\hat{r}(\phi)$ is a least square approximation of $r(\phi)$ on Chebyshev's polynomials [18] and $r(\phi)$ is a relatively smooth function. Therefore, the first order expansion, along with interval splitting, is sufficient in this case to show the required accuracy.

Listing 6 illustrates the PVS definitions used in this example. As a convention in PVS, real functions are written in lowercase, and interval functions are written in uppercase. In particular, `r`, `hat_r`, and `e` correspond to r, $\hat{r}$, and e, respectively, whereas `R`, `hat_R`, and `E` correspond to $\mathbf{r}$, $\hat{\mathbf{r}}$, and $\mathbf{e}$, respectively. PVS is a strongly typed language where every function has to be well-defined. The user-defined type `Phi` rules out values `phi` where `r(phi)` is undefined. In contrast to real operations, interval operations are defined everywhere. The empty interval acts as an exceptional value in cases where the real function is undefined.

Contrary to the approach described in [19], we do not have to generate a new Taylor approximation for each subrange. By using an interval-based Taylor expansion, the same expression can be reused for all the subranges. We do not suffer from the Taylor coefficients being irrational numbers, they are simply given by interval expressions involving rational functions. Relying on rational interval arithmetic leads to conceptually simpler proofs: one single global Taylor expansion has to be validated, and the proofs for all the subranges simply consist in an interval instantiation of this expansion.

5 A C++ Library for Real Applications

The splitting technique presented in Section 3 is implemented by an external library written in C++. Given a nu-

[5] Available from `http://www.wgs84.com`.

1063-6889/05 $20.00 © 2005 IEEE

PVS Listing 6 Functions R, $\hat{r}$, and E in PVS

```
a       : real = 6378137
f       : real = 1000000000/298257223563
umf2    : real = sq(1 - f)
sqmax   : real = 511225/262144

Phi : TYPE = {x: real |
              x ## [|0,715/512|]}

r(phi:Phi) : real =
  a / sqrt(1 + umf2 * sq(tan(phi)))

R(PHI:Interval) : Interval =
  a / Sqrt(1 + umf2 * Sq(Tan(PHI,4)),7)

hat_r(phi:Phi) : real =
  (4439091/4) + (sqmax - sq(phi)) * (
  (9023647/4) + (sqmax - sq(phi)) * (
  (13868737/64) + (sqmax - sq(phi)) * (
  (13233647/2048) + (sqmax - sq(phi)) * (
  (-1898597/16384) + (sqmax - sq(phi)) *
  (-6661427/131072)))))

hat_R(PHI:Interval) : Interval =
  (4439091/4) + (sqmax - Sq(PHI)) * (
  (9023647/4) + (sqmax - Sq(PHI)) * (
  (13868737/64) + (sqmax - Sq(PHI)) * (
  (13233647/2048) + (sqmax - Sq(PHI)) * (
  (-1898597/16384) + (sqmax - Sq(PHI)) *
  (-6661427/131072)))))

e(phi:Phi) : real =
  r(phi) - hat_r(phi)

E(PHI:Interval) : Interval =
  R(PHI) - hat_R(PHI)
```

merical problem, the library finds an appropriate subinterval division and generates proof obligations, in the form of lemmas, and proof guarantees, in the form of proof scripts, that yield a required accuracy. Lemmas and proofs are grouped in files such that they can be processed in parallel by PVS. The library also sets, as needed, the approximation parameters of the square root and trigonometric functions.

An external library has several advantages over a PVS proof strategy encoding interval splitting:

- The generation code does not need to understand the arcane of PVS internal structures.
- Porting this work to another proof assistant is possible as soon as a comparable interval library is available and batch proving is supported on the alternate proof assistant.
- A C++ library can efficiently explore many possibilities and generate the lemmas for a local optimal solution.
- Publicly available C++ libraries, such as Boost [1] and GMP's multiple precision rational arithmetic [5], can be used.

Although the C++ library checks that the reported intervals are sufficiently accurate compared to the one that are produced using exact rational arithmetic, the C++ library does not *formally* guarantee the result. The library provides an efficient mechanism to finely tune the input needed by PVS. The actual proof guarantee is provided only by the proof checker.

The utility `qmake` from Sun Grid Engine [20] was used to automatically target clusters of computers. The cluster used in this example consists of 48 processors 2.60 GHz Intel Xeon. A machine with 116 processors (1.80 GHz AMD Opteron) will soon be available. As this work is massively parallel, it will scale with no problem. The context file maintained by PVS is located on the local hard-drive of each node to enhance performances.

The splitting technique applied to Formula (5) starts with 100,000 subintervals to guarantee that the relative error is bounded by 1.36×10^{-6}. The trigonometric functions must be approximated to the 4th term and the square root to the 7th term. In total 9,935 intervals were considered. For each interval, 3 lemmas and their respective proof scripts were automatically generated by the C++ library. As expected, the final lemma in this development reads:

```
PHI : Interval = [|0,715/512|]

RI : LEMMA
  FORALL (phi:real) :
    phi ## PHI ==>
    e(phi) / r(phi) ##
      [|-136/1000000000,136/1000000000|]
```

6 Conclusion and Perspective

The examples presented in Sections 3 and 4 could have been handled in HOL-light[6] using one of the tools presented in [6]. According to Sturm's theorem [10, p. 434] that development is more efficient on these specific examples but it is limited to problems that can ultimately be approximated by polynomial functions. On the other hand, the techniques presented here can seamlessly guarantee rational approximations or even arbitrary programed approximations as long as they are piecewise continuously differentiable (for the developments of Section 4).

[6] Available from `http://www.cl.cam.ac.uk/users/jrh/hol-light`.

1063-6889/05 $20.00 © 2005 **IEEE**

In summary, the tools allow users to state and formally verify numerical properties in PVS with a minimal interaction with the theorem prover. No PVS expertise is required in most cases.

Research is conducted to study the feasibility of enhancing the prototypes with some of the following features:

- Floating point arithmetic rather than rational arithmetic as developed in [2].
- Use of high speed multiple precision techniques.
- Implementation of latest developments on Taylor's models [13], and mix Taylor's models and floating point arithmetic [17].

The tools are currently being used to check numerical properties of aircraft navigation algorithms developed at the National Institute of Aerospace (NIA).[7]

Acknowledgment

Proofs of Sections 4 and 5 were checked on high performance clusters set up and maintained by the Reso research project of the LIP computer science laboratory.

References

[1] H. Brönnimann, G. Melquiond, and S. Pion. The Boost interval arithmetic library. In *Real Numbers and Computers*, pages 65–80, Lyon, France, 2003.

[2] M. Daumas and G. Melquiond. Generating formally certified bounds on values and round-off errors. In *Real Numbers and Computers*, Dagstuhl, Germany, 2004.

[3] G. Dowek, A. Geser, and C. Muñoz. Tactical conflict detection and resolution in a 3-D airspace. In *Proceedings of the 4th USA/Europe Air Traffic Management R&DSeminar, ATM 2001*, Santa Fe, New Mexico, 2001. A long version appears as report NASA/CR-2001-210853 ICASE Report No. 2001-7.

[4] D. Gage and J. McCormick. We did nothing wrong. *Baseline*, 1(28):32–58, 2004.

[5] T. Granlund. *The GNU multiple precision arithmetic library*, 2004. Version 4.1.3.

[6] J. Harrison. Floating point verification in HOL light: the exponential function. Technical Report 428, University of Cambridge Computer Laboratory, 1997.

[7] Information Management and Technology Division. Patriot missile defense: software problem led to system failure at Dhahran, Saudi Arabia. Report B-247094, United States General Accounting Office, 1992.

[8] L. Jaulin, M. Kieffer, O. Didri, and E. Walter. *Applied interval analysis*. Springer, 2001.

[9] R. B. Kearfott. Interval computations: Introduction, uses, and resources. *Euromath Bulletin*, 2(1):95–112, 1996.

[10] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 1997. Third edition.

[11] L. Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, 1993.

[12] J. Lions et al. Ariane 5 flight 501 failure report by the inquiry board. Technical report, European Space Agency, Paris, France, 1996.

[13] K. Makino and M. Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003.

[14] C. Muñoz, V. Carreño, G. Dowek, and R. Butler. Formal verification of conflict detection algorithms. *International Journal on Software Tools for Technology Transfer*, 4(3):371–380, 2003.

[15] A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, 1990.

[16] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[17] N. Revol, K. Makino, and M. Berz. Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY. *Journal of Logic and Algebraic Programming*, 2005. To appear.

[18] T. Rivlin. *Chebychev polynomials*. John Wiley & Sons, 1990.

[19] J. Sawada. Formal verification of divide and square root algorithms using series calculation. In *3rd International Workshop on the ACL2 Theorem Prover and its Applications*, pages 31–49. University of Grenoble, 2002.

[20] Sun Microsystems. *Sun Grid Engine — Administration and User's guide*, 2002. Version 5.3.

[21] A. Tiwari, N. Shankar, and J. Rushby. Invisible formal methods for embedded control systems. *Proceedings of the IEEE*, 91(1):29–39, Jan. 2003.

[7] The PVS library Interval described in this paper is available from `http://research.nianet.org/~munoz/Interval`.

1063-6889/05 $20.00 © 2005 **IEEE**