ℝeal Number Proving in PVS

César A. Muñoz

NASA Langley Research Center
Cesar.A.Munoz@nasa.gov

PVS Class 2012

Outline

- Real Numbers in PVS
- Real Number Proving Tools
  - Basic Manipulations
  - Manip
  - Field
- Non-linear Arithmetic

Why Real Number Proving in PVS ?

- In *real* life , there are more numbers than integers (despite what model-checkers are telling you).
- Conceptually, it is easier to reason on a continuous framework than on a discrete one.
- A lot of classical results in calculus, trigonometry, and continuous mathematics.
- Sometimes you cannot avoid them: hybrid systems, engineering applications, etc.

I Use a CAS, Why Should I Bother with PVS?

Computer Algebra Systems (CAS):

- Mathematica, Maple, Matlab, Scilab,. . . offer very powerful symbolic and numerical engines.
- CAS do not aim *soundness*. Singularities and exceptions are well-known problems of CAS.
- CAS do not support specification languages but programming languages.

CAS and Theorem Provers

- Real analysis is not a traditional strength of theorem provers.
- Theorem provers and CAS can be integrated in useful ways:
  - Computer algebra systems can be used to perform mechanical simplifications and find potential solutions.
  - Theorem prover are then used to verify the correctness of a particular solution.

Real Numbers in PVS

- Reals are defined as an uninterpreted subtype of `number` in the prelude library:

  `real: TYPE+ FROM number`

- All numeric constants are `real`:
  - naturals: `0,1,...`
  - integers: `...,-1,0,1,...`
  - rationals: `...,-1/10,...,3/2,...`
- Decimal notation is supported: The decimal number `3.141516` is syntactic sugar for the rational number `31416/10000`.

PVS's `real` numbers are ℝeal

- All the standard properties: unbounded, connected, infinite, $\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}, \ldots$.
- ℝeal arithmetic: `1/3 + 1/3 + 1/3 = 1`.
- The type `real` is unbounded:

```
googol     : real = 10^100
googolplex : real = 10^googol

googol_prop : LEMMA
  googolplex > googol * googol
```

- ...but *machine physical limitations do apply*, e.g., don't try to prove `googol_prop` with `(grind)`.

ℝeal Number Proving in PVS
└Real Numbers in PVS

Rational Arithmetic is Built-in*

```
  |-------
{1} -(0.78 * 1.05504 * (0.92 - 0.78) * s) -
          0.78 * 1.08016 * (0.9 - 0.78) * s
          - 1.256 * (0.9 - 0.78) * s * u
          - 0.92944 * (0.92 - 0.78) * s * u
        + ...
         + 1.05504 * (0.92 - 0.78) * s * u
          + 1.08016 * (0.9 - 0.78) * s * u >= 0

Rule? (assert)
 |-------
{1} 0.0052256+-(0.115210368*s)+0.00844032*u+0.154213*s
    - 0.00568*(s*u) >= 0
```

*Decimal simplification available in PVS 6.0

Subtypes of `real`

```
nzreal  : TYPE+ = {r:real| r /= 0} % Nonzero reals
nnreal  : TYPE+ = {r:real| r >= 0} % Nonnegative reals
npreal  : TYPE+ = {r:real| r <= 0} % Nonpositive reals
negreal : TYPE+ = {r:real| r < 0} % Negative reals
posreal : TYPE+ = {r:real| r > 0} % Positive reals

rat     : TYPE+ FROM real
int     : TYPE+ FROM rat
nat     : TYPE+ FROM int
```

The uninterpreted type `number` is the only `real`'s supertype predefined in PVS: no complex numbers, no hyper-reals, no $\mathbb{R}^\infty$*, ...*

Real Numbers Properties

Real numbers in PVS are axiomatically defined in the PVS prelude:

- Theory `real_axioms`:
  Commutativity, associativity, identity, etc. These properties are known to the decision procedures, so they rarely need to be used in a proof.
- Theory `real_props`:
  Order and cancellation laws. These lemmas are not used automatically by the standard decision procedures.

Theory real_props

```
real_props: THEORY
 BEGIN
  both_sides_plus_le1: LEMMA x + z <= y + z IFF x <= y
  both_sides_plus_le2: LEMMA z + x <= z + y IFF x <= y
  both_sides_minus_le1: LEMMA x - z <= y - z IFF x <= y
  both_sides_minus_le2: LEMMA z - x <= z - y IFF y <= x
  both_sides_div_pos_le1: LEMMA x/pz <= y/pz IFF x <= y
  both_sides_div_neg_le1: LEMMA x/nz <= y/nz IFF y <= x
  ...
  abs_mult: LEMMA abs(x * y) = abs(x) * abs(y)
  abs_div: LEMMA abs(x / n0y) = abs(x) / abs(n0y)
  abs_abs: LEMMA abs(abs(x)) = abs(x)
  abs_square: LEMMA abs(x * x) = x * x
  abs_limits: LEMMA -(abs(x) + abs(y)) <= x + y AND
                    x + y <= abs(x) + abs(y)
END real_props
```


Predefined Operations

```
+, -, *: [real, real -> real]
/: [real, nzreal -> real]
-: [real -> real]

sgn(x:real)  : int = IF x >= 0 THEN 1 ELSE -1 ENDIF
abs(x:real)  : {nny: nnreal | nny >= x} = ...
max(x,y:real): {z: real | z >= x AND z >= y} = ...
min(x,y:real): {z: real | z <= x AND z <= y} = ...
^(x: real,i:{i:int | x /= 0 OR i >= 0}): real = ...
```

. . . and what about $\sqrt{}, \int, \log, \exp, \sin, \cos, \tan, \pi, \lim, \ldots$ *?*

NASA PVS Libraries

http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html

- `reals`: Square, square root, quadratic formula, polynomials.
- `analysis`: Real analysis, limits, continuity, derivatives, integrals.
- `vectors` and `vect_analysis`: Vector calculus and analysis.
- `series`: Power series, Taylor's theorem.
- `lnexp` and `lnexp_fnd`: Axiomatic and foundational logarithm, exponential, and hyperbolic functions.
- `trig` and `trig_fnd`: Axiomatic and foundational trigonometry.
- `complex`: Complex numbers.
- `float`: Floating point numbers.
- ...

To Be Or Not To Be (Foundational) ?

- Axiomatic theories `trig` and `lnexp` type-check faster.
- Foundational theories `trig_fnd` and `lnexp_fnd` have no axioms.
- Be careful what you wish for:

```
  |-------
{1}   sin(pi / 2) > 1 / 2

Rule? (grind)
Integral rewrites Integral[real](0, 1, atan_deriv_fn)
  to integral(0, 1, atan_deriv_fn)
atan_value rewrites atan_value(1)
  to integral(0, 1, atan_deriv_fn)
atan rewrites atan(1)
...
```

Real Number Proving Tools

- Basic manipulations.
- Manip.
- Field.

Basic Manipulations

PVS offers some proof commands for simple algebraic manipulations:

```
toy :

  |-------
{1}   x - x * x <= 1

Rule? (both-sides "-" "1/4")
toy :

  |-------
{1}   x - x * x - 1 / 4 <= 1 - 1 / 4
```

Note: Use `both-sides` only to add/subtract expressions.

Use case to Prove What You Need

```
  |-------
{1}   x - x * x - 1 / 4 <= 1 - 1 / 4

Rule? (case "x - x * x - 1 / 4 <= 0")
this yields  2 subgoals:
toy.1 :
{-1}  x - x * x - 1 / 4 <= 0
  |-------
[1]   x - x * x - 1 / 4 <= 1 - 1 / 4

Rule? (assert)
This completes the proof of toy.1.
```


Use hide to Focus on Relevant Formulas

```
toy.2 :
  |-------
{1}   x - x * x - 1 / 4 <= 0
[2]   x - x * x - 1 / 4 <= 1 - 1 / 4

Rule? (hide 2)
toy.2 :
  |-------
[1]   x - x * x - 1 / 4 <= 0
```

Arrange Expressions With `case-replace`

```
toy.2 :
  |-------
[1]   x - x * x - 1 / 4 <= 0
Rule? (case-replace
       "x - x * x - 1 / 4 = -(x-1/2)*(x-1/2)"
       :hide? t)
this yields  2 subgoals:
toy.2.1 :
  |-------
{1}   -(x - 1 / 2) * (x - 1 / 2) <= 0
```


Introduce New Names With `name-replace`

```
toy.2.1 :
  |-------
{1}   -(x - 1 / 2) * (x - 1 / 2) <= 0

Rule? (name-replace "X" "(x-1/2)")
toy.2.1 :

  |-------
{1}   -X * X <= 0

Rule? (assert)
This completes the proof of toy.2.1.
```

Tip: Don't Reinvent the Wheel

Look into the NASA libraries first!

Theory `reals@quadratic`:

```
quadratic_le_0 : LEMMA
  a*sq(x) + b*x + c <= 0 IFF
  ((discr(a,b,c) >= 0 AND
    ((a > 0 AND x2(a,b,c) <= x AND x <= x1(a,b,c)) OR
     (a < 0 AND (x <= x1(a,b,c) OR x2(a,b,c) <= x)))) OR
   (discr(a,b,c) < 0 AND c <= 0))
```


A Simpler Proof

```
  |-------
{1}   x * (1 - x) <= 1

Rule? (lemma "quadratic_le_0"
               ("a" "-1" "b" "1" "c" "-1" "x" "x"))
      (grind)
Trying repeated skolemization, instantiation, and
if-lifting,
Q.E.D.
```

Manip

- Manip is a PVS package for algebraic manipulations of real-valued expressions.
- `http://shemesh.larc.nasa.gov/people/bld/manip.html`.
- The package consists of:
  - Strategies.
  - Extended notations for formulas and expressions.
  - Emacs extensions.
  - Support functions for strategy developers.
- Manip is pre-installed in PVS 5.0.

Manip Strategies: Basic Manipulations

Strategy	**Description**
(`swap-rel` *fnums*)	Swap sides and reverse relations
(`swap!` *exprloc*)	$x \circ y \Rightarrow y \circ x$
(`group!` *exprloc* `LR`)	$(x \circ y) \circ z \Rightarrow x \circ (y \circ z)$
(`flip-ineq` *fnums*)	Negate and move inequalities
(`split-ineq` *fnum*)	Split $\leq$ $(\geq)$ into $<$ $(>)$ and $=$

Extended Formula Notation

- Standard
  - `*`: All formulas.
  - `-`: All formulas in the antecedent.
  - `+`: All formulas in the consequent.
- Extended (Manip strategies only)
  - `(^` $n_1 \ldots n_k$`)`: All formulas but $n_1, \ldots, n_k$
  - `(-^` $n_1 \ldots n_k$`)`: All antecedent formulas but $n_1, \ldots, n_k$
  - `(+^` $n_1 \ldots n_k$`)`: All consequent formulas but $n_1, \ldots, n_k$

(Basic) Extended Expression Notation

- Term indexes:
  - `L`,`R`: Left- or right-hand side of a formula.
  - n: n-th term from left to right in a formula.
  - $-n$: n-th term from right to left in a formula.
  - `*`: All terms in a formula.
  - `(^` n_1 $\ldots n_k$`)`: All terms in a formula but $n_1, \ldots, n_k$.
- Location references:
  - `(!` *fnum* `LR` $i_1 \ldots i_n$`)`: Term in formula *fnum*, Left- or Right-hand side, at recursive path location $i_1 \ldots i_k$.

Examples

```
{-1}  x * r + y * r + 1 >= r - 1
  |-------
{1}   r = y * 2 * x + 1

Rule? (swap-rel -1)

{-1}  r - 1 <= x * r + y * r + 1
  |-------
[1]   r = y * 2 * x + 1

Rule?  (swap! (! -1 R 1))

{-1}  r - 1 <= r * x + y * r + 1
  |-------
[1]   r = y * 2 * x + 1
```



```
{-1}  r - 1 <= r * x + y * r + 1
  |-------
[1]   r = y * 2 * x + 1

Rule? (group! (! 1 R 1) R)

[-1]  r - 1 <= r * x + y * r + 1
  |-------
{1}   r = y * (2 * x) + 1

Rule? (flip-ineq -1)

  |-------
{1}   r - 1 > r * x + y * r + 1
[2]   r = y * (2 * x) + 1
```

```
{-1}   r - 1 <= r * x + y * r + 1
  |-------
{1}    r = y * (2 * x) + 1

Rule? (split-ineq -1)

{-1}  r - 1 [=] r * x + y * r + 1
{-2}  r - 1 <= r * x + y * r + 1
  |-------
{1}   r = y * (2 * x) + 1

Rule? (postpone)

{-1}  r - 1 <= r * x + y * r + 1
  |-------
{1}   r - 1 [=] r * x + y * r + 1
{2}   r = y * (2 * x) + 1
```


More Strategies

Strategy	**Description**
(`mult-by` *fnums term*)	Multiply formula by term
(`div-by` *fnums term*)	Divide formula by term
(`move-terms` *fnum* `L|R` *tnums*)	Move additive terms left and right
(`isolate` *fnum* `L|R` *tnum*)	Isolate additive terms
(`cross-mult` *fnums*)	Perform cross-multiplications
(`factor` *fnums*)	Factorize formulas
(`factor!` *exprloc*)	Factorize terms
(`mult-eq` *fnum fnum*)	Multiply equalities
(`mult-ineq` *fnum fnum*)	Multiply inequalities

More Examples

```
{-1}  (x * r + y) / pa > (r - 1) / pb
  |-------
{1}   r - y * 2 * x = 1

Rule? (cross-mult -1)

{-1}  pb * r * x + pb * y > pa * r - pa
  |-------
[1]   r - y * 2 * x = 1

Rule? (isolate 1 L 1)

[-1]  pb * r * x + pb * y > pa * r - pa
  |-------
{1}   r = 1 + y * 2 * x
```



```
{-1}  x * y - pa + na < x * na * pa
{-2}  r - y * 2 * x = 1
  |-------
{1}   2 * pa = 2 * x + 2 * y

Rule? (move-terms -1 L (2 3))

{-1}  x * y < x * na * pa + pa - na
[-2]  r - y * 2 * x = 1
  |-------
[1]   2 * pa = 2 * x + 2 * y

Rule? (factor 1)

[-1]  x * y < x * na * pa + pa - na
[-2]  r - y * 2 * x = 1
  |-------
{1}   2 * pa = 2 * (x + y)
```

```
[-1]  x * y < x * na * pa + pa - na
[-2]  r - y * 2 * x = 1
  |-------
{1}   2 * pa = 2 * (x + y)

Rule? (mult-eq -1 -2)

{-1}  (x * y)*(r - y * 2 * x) < (x * na * pa + pa - na)*1
[-2]  x * y < x * na * pa + pa - na
[-3]  r - y * 2 * x = 1
  |-------
[1]   2 * pa = 2 * (x + y)

Rule? (mult-ineq -1 -2 (+ +))

{-1}  ((x*y)*(r-y*2*x))*(x*y) < ((x*na*pa+pa-na)*1)*(x*na*pa+pa-na)
...
  |-------
[1]   2 * pa = 2 * (x + y)
```



```
...
  |-------
[1]   2 * pa = 2 * (x + y)

Rule? (div-by 1 "2")

...
  |-------
{1}   pa = (x + y)

Rule? (mult-by 1 "100")

...
  |-------
{1}   100*pa = 100*(x + y)
```

Field

- Field is a PVS package for simplifications in the closed field of real numbers.
- `http://shemesh.larc.nasa.gov/people/cam/Field`.
- The package consists of:
  - The strategy `field`.
  - Several *extra-tegies*.
  - Field is pre-installed in PVS 5.0.

field

```
{-1}  vox > 0
{-2}  s * s - D*D > D
{-3}  s * vix * voy - s * viy * vox /= 0
{-4}  ((s * s - D*D) * voy - D * vox * sqrt(s*s - D*D))/
       (s * (vix * voy - vox * viy)) * s * vox /= 0
{-5}   voy * sqrt(s * s - D*D) - D * vox /= 0
  |-------
{1}   (viy * sqrt(s * s - D*D) - vix * D) /
       (voy * sqrt(s * s - D*D) - vox * D) =
       (D*D - s * s) / (((s * s - D*D) * voy - D * vox *
       sqrt(s * s - D*D)) /
       (s * (vix * voy - vox * viy)) * s * vox) +
       vix / vox

Rule? (field 1)
Q.E.D.
```

Some Extra-tegies

Strategy	Description
(grind-reals)	grind with real_props
(cancel-by *fnum* *term*)	Cancel a common term in a formula
(skoletin *fnum*)	Skolemize let-in expressions
(skeep *fnum*)	Skolemize with same variable names
(neg-formula *fnum*)	Negate a formula
(add-formula *fnum* *fnum*)	Add formulas
(sub-formula *fnum* *fnum*)	Subtract formulas

grind-reals

```
  |-------
{1}   (x - 1 / 2) * (x - 1 / 2) >= 0

Rule? (grind-reals :nodistrib 1)

Q.E.D.
```

cancel-by

```
{-1}  4 * (pa * pb) + (pa * 6) * pa = pa * ((c + 1) * 2)
   |-------
{1}   2 * pb + 3 * pa = c

Rule? (cancel-by -1 "2*pa")

{-1}  (3 * pa) + (2 * pb) = 1 + c
   |-------
{1}   2 * pa = 0
{2}   3 * pa + 2 * pb = c
```


PVS's Let-in Expressions

- Let-in expressions are used in PVS to introduce local definitions.
- They are automatically unfolded by the theorem prover.

```
   |-------
{1}   LET a = a * y + 2 IN
        LET b = a + x IN
          LET c = a + b IN -b + 4 * a * c / 2 = 0

Rule? (assert)

   |-------
{1}   (32 + 8 * (x*x * y*y) + 4 * (x*x*y) + 16 * (x*y) +
        16 * (x*y) + 8*x) / 2 + -(2 + x*y + x) = 0
```

Let-in Expressions Go Wild

```
  |-------
{1}   LET a = (x + 1) IN LET b = a * a IN
      LET c = b * b IN c * c >= a

Rule? (assert)

  |-------
{1} 1 + x + (x*x*x*x*x*x*x*x + x*x*x*x*x*x*x)
        + (x*x*x*x*x*x*x + x*x*x*x*x*x)
        + (x*x*x*x*x*x*x + x*x*x*x*x*x)
      ...
        + (x*x + x)
        + (x*x + x)
        + (x*x + x)
        >= 1 + x
```


skoletin

```
  |-------
{1}   LET a = (x + 1) IN LET b = a * a IN
      LET c = b * b IN c * c >= a

Rule? (skoletin 1)
{-1}  a = (x + 1)
  |-------
{1}   LET b = a * a IN LET c = b * b IN c * c >= a

Rule? (skoletin* 1)
{-1}  c = b * b
{-2}  b = a * a
[-3]  a = (x + 1)
  |-------
{1}   c * c >= a
```

More examples

```
  |-------
{1}   FORALL (nnx: nnreal, x: real):
        nnx > x - nnx*nnx AND x + 2 * nnx*nnx >= 4 * nnx
        IMPLIES nnx > 1

Rule? (skeep)
{-1}  nnx > x - nnx*nnx
{-2}  x + 2 * nnx*nnx >= 4 * nnx
  |-------
{1}   nnx > 1

Rule? (neg-formula -1)
{-1}  nnx*nnx - x > -nnx
[-2]  x + 2 * nnx*nnx >= 4 * nnx
  |-------
[1]   nnx > 1
```



```
{-1}  nnx*nnx - x > -nnx
[-2]  x + 2 * nnx*nnx >= 4 * nnx
  |-------
[1]   nnx > 1

Rule? (add-formulas -1 -2)
{-1}  3 * (nnx*nnx) > -nnx + 4 * nnx
  |-------
[1]   nnx > 1

Rule? (cancel-by -1 "nnx")

Q.E.D.
```

Non-linear Arithmetic

To be continued tomorrow . . .