Nonlinear Arithmetic in PVS

Anthony Narkawicz

NASA Langley Research Center
anthony.narkawicz@nasa.gov

PVS Class 2012

Nonlinear Arithmetic

Interval can solve problems like

```
ex_ba : LEMMA
   x ## [|-1/2,0|] IMPLIES
   abs(ln(1+x) - x) - epsilon <= 2*sq(x)
```

Bernstein can solve problems like:

```
p1 : LEMMA
  FORALL (x,y:real): -0.5 <= x AND x <= 1 AND
                     -2 <= y AND y <= 1 IMPLIES
    4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 > -3.4

p2 : LEMMA
  EXISTS (x,y:real): -0.5 <= x AND x <= 1 AND
                     -2 <= y AND y <= 1 AND
    4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 < -3.39
```

These lemmas are proved by executing a single command!

Interval

http://shemesh.larc.nasa.gov/people/cam/Interval

- Interval is a PVS package for interval analysis.
- The package consists of:
  - The library `interval_arith`, which presents a formalization of interval analysis for real-valued functions including: trigonometric functions, logarithm and exponential functions, square root, absolute value, etc.
  - The strategy `numerical`, which implements a provably correct branch-and-bound interval analysis algorithm.
- Interval is part of the NASA PVS Libraries.

A Simple Problem

Prove that the turn rate of an aircraft with a bank angle of 35^o is greater than 3^o per second.

A Simple Problem

Prove that the turn rate of an aircraft with a bank angle of 35^o is greater than 3^o per second.

```
IMPORTING interval_arith@strategies

g:posreal=9.8        %[m/s^2]

v:posreal=250*0.514  %[m/s]

tr(phi:(Tan?)): MACRO real = g*tan(phi)/v

tr_35 : LEMMA
  3*pi/180 <= tr(35*pi/180)
```

numerical

```
 tr_35 :

  |-------
{1}   3 * pi / 180 <= g * tan(35 * pi / 180) / v

Rule? (numerical)
Evaluating formula using numerical approximations,
Q.E.D.
```

Note that `pi` is the mathematical irrational number π and `tan` is the trigonometric function tan.

numerical

```
 tr_35 :

  |-------
{1}   3 * pi / 180 <= g * tan(35 * pi / 180) / v

Rule? (numerical)
Evaluating formula using numerical approximations,
Q.E.D.
```

Note that `pi` is the mathematical irrational number π and `tan` is the trigonometric function tan.

A Simple Property of Logarithms

```
G(x:real|x < 1): MACRO real = 3*x/2 - ln(1-x)

A_and_S : LEMMA
  let x = 0.5828 in
    G(x) > 0
```

A Simple Property of Logarithms

```
 A_and_S :

  |-------
{1}   LET x = 0.5828 IN 3 * x / 2 - ln(1 - x) > 0

Rule? (numerical)
Evaluating formula using numerical approximations,
Q.E.D.
```

Note that `ln` is natural logarithm function.

A Simple Property of Logarithms

```
 A_and_S :

   |-------
{1}   LET x = 0.5828 IN 3 * x / 2 - ln(1 - x) > 0

Rule? (numerical)
Evaluating formula using numerical approximations,
Q.E.D.
```

Note that `ln` is natural logarithm function.

Interval Arithmetic

```
{-1}  x ## [| 0, 2 |]
  |-------
{1}   sqrt(x) + sqrt(3) < pi + 0.1

Rule? (numerical :vars "x")
Evaluating formula using numerical approximations,
Q.E.D.
```

Interval Arithmetic

```
{-1}  x ## [| 0, 2 |]
  |-------
{1}   sqrt(x) + sqrt(3) < pi + 0.1

Rule? (numerical :vars "x")
Evaluating formula using numerical approximations,
Q.E.D.
```

Interval Analysis

Prove that for all $x \in [-\frac{1}{2}, 0]$,

$$|\ln(1+x) - x| - \epsilon \leq 2x^2,$$

where $\epsilon = 0.15$:[1]

```
ex_ba : LEMMA
   x ## [|-1/2,0|] IMPLIES
   abs(ln(1+x) - x) - epsilon <= 2*sq(x)
```

[1] Thanks to Behzad Akbarpour.

instint

```
ex_ba :
 |-------
{1} FORALL (x: real):
      x ## [|-1/2,0|] IMPLIES abs(ln(1+x)-x)-0.15 <= 2*sq(x)

Rule? (skeep)
ex_ba :
{-1}  x ## [| -1 / 2, 0 |]
  |-------
{1}   abs(ln(1 + x) - x) - 0.15 <= 2 * sq(x)

Rule? (numerical :vars (("x" 10)))
Evaluating formula using numerical approximations,
Q.E.D.
```

instint

```
ex_ba :
 |-------
{1} FORALL (x: real):
      x ## [|-1/2,0|] IMPLIES abs(ln(1+x)-x)-0.15 <= 2*sq(x)

Rule? (skeep)
ex_ba :
{-1}  x ## [| -1 / 2, 0 |]
  |-------
{1}   abs(ln(1 + x) - x) - 0.15 <= 2 * sq(x)

Rule? (numerical :vars (("x" 10)))
Evaluating formula using numerical approximations,
Q.E.D.
```

instint

```
ex_ba :
 |-------
{1} FORALL (x: real):
      x ## [|-1/2,0|] IMPLIES abs(ln(1+x)-x)-0.15 <= 2*sq(x)

Rule? (skeep)
ex_ba :
{-1}  x ## [| -1 / 2, 0 |]
  |-------
{1}   abs(ln(1 + x) - x) - 0.15 <= 2 * sq(x)

Rule? (numerical :vars (("x" 10)))
Evaluating formula using numerical approximations,
Q.E.D.
```

Bernstein

http://shemesh.larc.nasa.gov/people/cam/Bernstein

- Bernstein is a PVS package for solving multivariate polynomial global optimization problems using Bernstein polynomials.
- The package consists of:
  - The library `Bernstein`, which presents a formalization of an efficient representation of multivariate polynomials.
  - The strategy `bernstein`, which discharges simply quantified multivariate polynomial inequalities on closed/open ranges.
  - `Grizzly`, which is a prototype client-server tool for solving global optimization problems.
- Bernstein is part of the NASA PVS Libraries.

Solving Polynomial Inequalities

```
IMPORTING Bernstein@strategy

p1 : LEMMA
  FORALL (x,y:real): -0.5 <= x AND x <= 1 AND
                     -2 <= y AND y <= 1 IMPLIES
    4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 > -3.4

p2 : LEMMA
  EXISTS (x,y:real): -0.5 <= x AND x <= 1 AND
                     -2 <= y AND y <= 1 AND
    4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 < -3.39
```

```
 |-------
{1} FORALL (x, y: real):
     -0.5 <= x AND x <= 1 AND -2 <= y AND y <= 1 IMPLIES
      4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 > -3.4

Rule? (bernstein)
Proving polynomial inequality using Bernstein'basis,
Q.E.D.
```

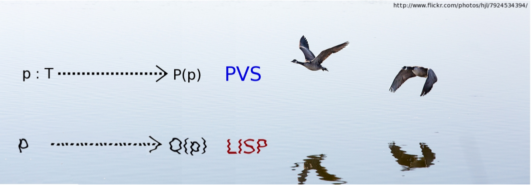
```
 |-------
{1} FORALL (x, y: real):
     -0.5 <= x AND x <= 1 AND -2 <= y AND y <= 1 IMPLIES
      4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 > -3.4

Rule? (bernstein)
Proving polynomial inequality using Bernstein'basis,
Q.E.D.
```

```
|-------
{1} EXISTS (x, y: real):
          -0.5 <= x AND x <= 1 AND -2 <= y AND y <= 1 AND
      4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 < -3.39

Rule? (bernstein)
Proving polynomial inequality using Bernstein's basis,
Q.E.D.
```

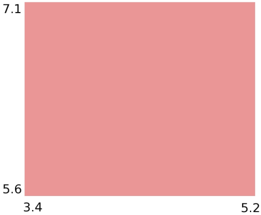
```
|-------
{1} EXISTS (x, y: real):
         -0.5 <= x AND x <= 1 AND -2 <= y AND y <= 1 AND
      4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 < -3.39

Rule? (bernstein)
Proving polynomial inequality using Bernstein's basis,
Q.E.D.
```

Reflection

Both Interval and Bernstein use computation reflection

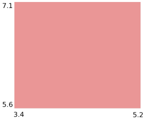


Both try to prove the result on a large box:

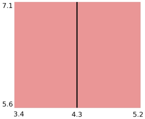


Reflection

Interval and Bernstein each have a function that can (sometimes) tell whether the result holds on a particular box.

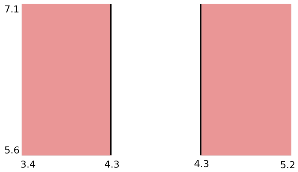


If that function returns *unknown*, then the box is split in two:

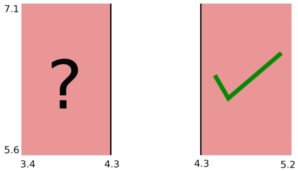


Reflection

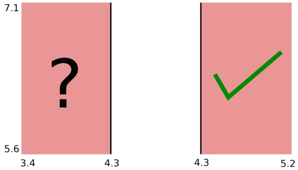
The two halves of the big box are now considered separately



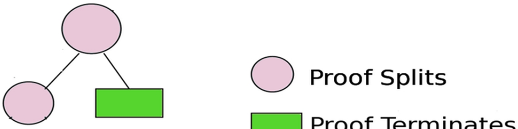
Perhaps we can prove it on the right but not the left sub-box:



Reflection

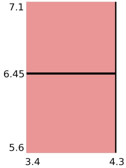


This turns the proof tree into

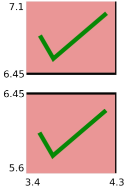


Reflection

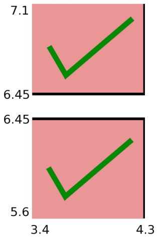
Now we split the left hand box into two smaller pieces:



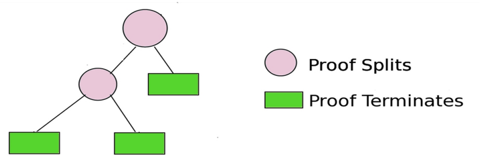
Perhaps the result can be be proved on each of these boxes:



Reflection

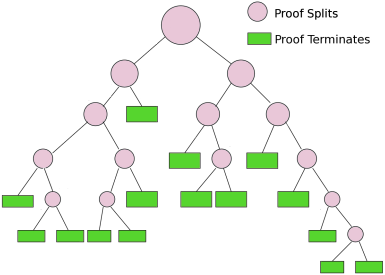


This turns the proof tree into



Reflection

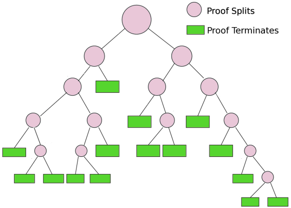
Sometimes the proof tree can get very large:



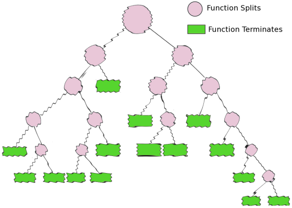
With 100s of splits, the proof is infeasible in the PVS prover language

Reflection

Instead of having PVS develop a proof of that looks like



There is a recursive reflection function in PVS whose execution looks like

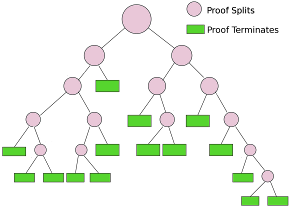


Reflection

- The proof tree happens entirely in LISP
- All of the proofs have the same length in PVS, for Interval and Bernstein
- Complicated problems could not be solved in PVS without using computational reflection in this way.

Reflection

There is a generic function in the *structures* library that defines a recursive splitting algorithm for arbitrary types.



It can be used to solve almost any binary branching problem

Reflection

```
branch_and_bound(simplify,evaluate,branch,subdivide,denorm,combine,prune,le,ge,select,accumulate,maxdepth)
                (obj,dom,acc,(dirvars|length(dirvars) <= maxdepth)) :
  RECURSIVE Output =
  LET  nobj    = simplify(obj),
       thisans = evaluate(dirvars,dom,nobj),
       newacc1 = IF none?(acc) THEN thisans ELSE accumulate(TRUE,some(acc),thisans) ENDIF,
       thisout = mk_out(thisans,ge(dirvars,newacc1,thisans),length(dirvars),0)
   IN
    IF length(dirvars)=maxdepth OR le(thisans) OR thisout`exit OR prune(dirvars,newacc1,thisans) THEN
       thisout
    ELSE
       LET
         (dir,v)      = select(dirvars,newacc1,dom,nobj),
         funsplit     = branch(v,nobj),
         domsplit     = subdivide(v,dom),
         (sp1,sp2)    = IF dir THEN (funsplit`1,funsplit`2) ELSE (funsplit`2,funsplit`1) ENDIF,
         (dom1,dom2)  = IF dir THEN (domsplit`1,domsplit`2) ELSE (domsplit`2,domsplit`1) ENDIF,
         firstout     = branch_and_bound(simplify,evaluate,branch,subdivide,denorm,combine,
                                  prune,le,ge,select,accumulate,maxdepth)
                                 (sp1,dom1,Some(newacc1),pushDirVar((dir,v),dirvars))
       IN
         IF firstout`exit THEN
           mk_out(combine(v,denorm((dir,v),firstout`ans),thisans),
                  TRUE,firstout`depth,firstout`splits+1)
         ELSE
           LET
             newacc2     = accumulate(FALSE,newacc1,firstout`ans),
             secondout   = branch_and_bound(simplify,evaluate,branch,subdivide,denorm,combine,
                                   prune,le,ge,select,accumulate,maxdepth)
                                  (sp2,dom2,Some(newacc2),pushDirVar((NOT dir,v),dirvars)),
             (real1,real2) = IF dir THEN (firstout,secondout) ELSE (secondout,firstout) ENDIF
           IN
             mk_out(combine(v,denorm(left(v),real1`ans),denorm(right(v),real2`ans)),
                    secondout`exit,
                    max(firstout`depth,secondout`depth),
                    firstout`splits+secondout`splits+1)
         ENDIF
    ENDIF
  MEASURE maxdepth-length(dirvars)
```

- This algorithm can be evaluated by (eval-formula)
- ... and therefore, it can be used for computational reflection
- ... as long as everything it has to compute is a ground term

Reflection

Yogi Berra: *"It aint over 'til it's over"*

Interval and Bernstein are not perfect

This algorithm may not terminate, even with Interval and Bernstein

There are some inequalites that are true that will not prove in a reasonable amount of time

THE END

THE END