PVSio: A Rapid Prototyping Tool for PVS

César A. Muñoz

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

A Trivia Question

- In addition to prove mathematical theories, what else can you do in PVS ?
- To animate them!

A Trivia Question

- In addition to prove mathematical theories, what else can you do in PVS ?
- To animate them!

Animation

- What: Animation is the process of executing a specification to validate its intended semantics.
- Why: It is cheaper, faster, more fun to test a specification than to prove it.
- How: The PVS ground evaluator.

PVS is a Functional Programming Language

Most specifications in PVS are functional:

```
sqrt_newton(a:nnreal,n:nat): recursive posreal =
  if n=0 then a+1
  else let r=sqrt_newton(a,n-1) in
    (1/2)*(r+a/r)
  endif
  measure n+1
```

Grind as a Calculator

```
|-------
{1}   sqrt_newton(2, 3) <= 3 / 2

Rule? (grind)
...
sqrt_newton rewrites sqrt_newton(2, 3)
  to (1/2)*(2/(3*((1/2)*(1/2)) + (1/2)*(2/(3*(1/2)
  + (1/2)*(2/3))) + (1/2)*(1/2)*(2/3)))
  + 3*((1/2)*(1/2)*(1/2)) + (1/2)*(1/2)*(2/(3*(1/2)
  + (1/2)*(2/3))) + (1/2)*(1/2)*(1/2)*(2/3)
Trying repeated skolemization, instantiation, and
if-lifting,
Q.E.D.
```

Grind as a Calculator

```
|-------
{1}   sqrt_newton(2, 3) <= 3 / 2

Rule? (grind)
...
sqrt_newton rewrites sqrt_newton(2, 3)
  to (1/2)*(2/(3*((1/2)*(1/2)) + (1/2)*(2/(3*(1/2)
  + (1/2)*(2/3))) + (1/2)*(1/2)*(2/3)))
  + 3*((1/2)*(1/2)*(1/2)) + (1/2)*(1/2)*(2/(3*(1/2)
  + (1/2)*(2/3))) + (1/2)*(1/2)*(1/2)*(2/3)
Trying repeated skolemization, instantiation, and
if-lifting,
Q.E.D.
```

Grind is an Inefficient Calculator!

```
 |-------
{1}   2 < sqrt_newton(2, 10) * sqrt_newton(2, 10)

Rule? (grind)

...
sqrt_newton rewrites sqrt_newton(2, 4)
  to (1/2) * (2 / ((1/2) * (2 / (3 * ((1/2) * (1/2))
  + (1/2) * (2/(3 * (1/2) + (1/2) * (2/3)))
  + (1/2) * (1/2) * (2/3)))
  + 3 * ((1/2) * (1/2) * (1/2))))
  + ...
```

Grind is an Inefficient Calculator!

```
 |-------
{1}   2 < sqrt_newton(2, 10) * sqrt_newton(2, 10)

Rule? (grind)

...
sqrt_newton rewrites sqrt_newton(2, 4)
  to (1/2) * (2 / ((1/2) * (2 / (3 * ((1/2) * (1/2))
  + (1/2) * (2/(3 * (1/2) + (1/2) * (2/3)))
  + (1/2) * (1/2) * (2/3)))
  + 3 * ((1/2) * (1/2) * (1/2))))
  + ...
```

The PVS Ground Evaluator

- An experimental feature of PVS 3.x.
- An efficient Lisp code generator for PVS functional specifications.
- A read-eval-loop interface available with the Emacs command `M-x pvs-ground-evaluator`.
- Remark: The ground evaluator is not integrated into the theorem prover.

An Efficient Calculator . . .

```
<GndEval> "sqrt_newton(2,3) <= 3/2"
==>
TRUE

<GndEval> "2 < sqrt_newton(2,10) * sqrt_newton(2,10)"

==>
TRUE
```

An Efficient Calculator . . .

```
<GndEval> "sqrt_newton(2,3) <= 3/2"
==>
TRUE

<GndEval> "2 < sqrt_newton(2,10) * sqrt_newton(2,10)"

==>
TRUE
```

An Efficient Calculator . . .

```
<GndEval> "sqrt_newton(2,3) <= 3/2"
==>
TRUE

<GndEval> "2 < sqrt_newton(2,10) * sqrt_newton(2,10)"

==>
TRUE
```

An Efficient Calculator . . .

```
<GndEval> "sqrt_newton(2,3) <= 3/2"
==>
TRUE

<GndEval> "2 < sqrt_newton(2,10) * sqrt_newton(2,10)"

==>
TRUE
```

An Efficient Calculator . . .

```
<GndEval> "sqrt_newton(2,3) <= 3/2"
==>
TRUE

<GndEval> "2 < sqrt_newton(2,10) * sqrt_newton(2,10)"

==>
TRUE
```

. . . with a Poor Interface

```
<GndEval> "sqrt_newton(2,10)"

==>
106854041125800542495773099620277025175306170088676005050927755840860348663163076245675995712730905205536196480957618323863188053907381032775618232842813250031327063713965171465823575298674176159059086658790668539856665540281158705113265823003418661673043593439606033431706584881164409983476684441998170083079481020253769836865387591260387081397004439539734272848728362663930358361315699961450389500338289937102755577233304637383594575977288249125534790026095102838876667217608828542941439658998944011413943276015695324890773234847928444853126350628661998571065399284273825907413878202296844504371623790338598978694908324220272087549299684847173162322470343065775557217077239794496483926252726485551649317661664229348905711734284588432155133188346299073522175191156132982617780418149874006455063289968791553134685549334807307896812356752391457302355322568503490327048784946365287712061717308315403135248569103002103080206182315299185952314636972146502157415893978546131789429918177417275181480917118900032751147136108239668993919828107502725670206223908819154789790432679887752824654751611340450139007446848131012363204673416951245204209273923257266490795228062440719491393987354197415008629749549798915103833193692424860580372959405221411095017991925804059522093130803467583186506109442752416009090650437487476983661568
```

. . . with a Poor Interface

```
<GndEval> "sqrt_newton(2,10)"

==>
```

106854041125800542495773099620277025175306170088676005050927755840860348663163076245675995712730905205536196480957618323863188053907381032775618232842813250031327063713965171465823575298674176159059086658790668539856665540281158705113265823003418661673043593439606033431706584881164409983476684441998170083079481020253769836865387591260387081397004439539734272848728362663930358361315699961450389500338289937102755577233304637383594575977288249125534790026095102838876667217608828542941439658998944011413943276015695324890773234847928444853126350628661998571065399284273825907413878202296844504371623790338598978694908324220272087549299684847173162322470343065 7/75557217077239794496483926252726485551649317661664229348905711734284588432155133188346299073522175191156132982617780418149874006455063289968791553134685549334807307896812356752391457302355532256850349032704878494636528771206171730831540313524856910300210308020618231529918595231463697214650215741589397854613178942991817741727518148091711890003275114713610823966899391982810750272567020622390881915478979043267988775282465475161134045013900744684813101236320467341695124520420927392325726649079522806244071949139398735419741500862974954979891510383319369242486058037295940522141109501799192580405952209313080346758318650610944275241600909065043748747698366156 8
```
```

Outline

PVSio

- An alternative interface to the PVS Ground Evaluator.
- Implemented as a PVS package (prelude library extension).
- Safely integrated into the theorem prover.
- Pre-installed in PVS 5.0.

More than a Pretty Face

- A predefined set of PVS functions for input/output operations, side-effects, unbounded-loops, exceptions, string manipulations, and floating point arithmetic
- A high level interface for extending PVS programming language features.
- A tool for rapid prototyping.

M-x pvsio

```
+----
| PVSio-4.a
| ...
+----

<PVSio> println(sqrt_newton(2,10));

1.4142135
```

M-x pvsio

```
+----
| PVSio-4.a
| ...
+----

<PVSio> println(sqrt_newton(2,10));

1.4142135
```

M-x pvsio

```
+----
| PVSio-4.a
| ...
+----

<PVSio> println(sqrt_newton(2,10));

1.4142135
```

Input Operations

```
<PVSio> let x = read_real in
        println("sqrt("+x+")="+sqrt_newton(x,10));

10

sqrt(10)=3.1622777
```

Input Operations

```
<PVSio> let x = read_real in
        println("sqrt("+x+")="+sqrt_newton(x,10));

10

sqrt(10)=3.1622777
```

Input Operations

```
<PVSio> let x = read_real in
        println("sqrt("+x+")="+sqrt_newton(x,10));

10

sqrt(10)=3.1622777
```

Floating Points and a Random Surprise

```
<PVSio> SQRT(2);
==>
1.4142135

<PVSio> RANDOM = RANDOM;
==>
FALSE

<PVSio> let r=RANDOM in r = r;
==>
TRUE
```

Floating Points and a Random Surprise

```
<PVSio> SQRT(2);
==>
1.4142135

<PVSio> RANDOM = RANDOM;
==>
FALSE

<PVSio> let r=RANDOM in r = r;
==>
TRUE
```

Floating Points and a Random Surprise

```
<PVSio> SQRT(2);
==>
1.4142135

<PVSio> RANDOM = RANDOM;
==>
FALSE

<PVSio> let r=RANDOM in r = r;
==>
TRUE
```

Floating Points and a Random Surprise

```
<PVSio> SQRT(2);
==>
1.4142135

<PVSio> RANDOM = RANDOM;
==>
FALSE

<PVSio> let r=RANDOM in r = r;
==>
TRUE
```

Floating Points and a Random Surprise

```
<PVSio> SQRT(2);
==>
1.4142135

<PVSio> RANDOM = RANDOM;
==>
FALSE

<PVSio> let r=RANDOM in r = r;
==>
TRUE
```

Floating Points and a Random Surprise

```
<PVSio> SQRT(2);
==>
1.4142135

<PVSio> RANDOM = RANDOM;
==>
FALSE

<PVSio> let r=RANDOM in r = r;
==>
TRUE
```

Furthermore

- String manipulations.
- Streams and files.
- Unbounded loops.
- Exceptions.
- Local and global variables.
- Basic parsing and lexing.
- PVS parsing and typechecking.

Semantic Attachments

- A high-level interface to the the PVS Common Lisp machine.
- A user-friendly mechanism for extending the ground evaluator.
- Lisp functions attached to uninterpreted PVS functions.

User-defined Attachments

- How:

```
(defattach theory.name
   doc-string
   body)
```

- Where: `pvs-attachments` or `<user>/.pvs-attachments`.
- Example:

```
;; File: pvs-attachments
(defattach my_cosh.cosh (x)
   "Hyperbolic cosine of X"
   (cosh x))
```

Animation of Specifications

```
maxl_ax : THEORY
BEGIN
 IMPORTING list[real]

 maxl : [list->real]

 l : VAR list
 x : VAR real

 Maxl : AXIOM
   member(x,l) implies
     x <= maxl(l)
END maxl_ax
```

```
maxl_th : THEORY
BEGIN
 IMPORTING list[real]

 maxl(l:list) : RECURSIVE real =
   cases l of
     null : 0,
     cons(a,r) : max(a,maxl(r))
   endcases
   MEASURE l by <<
END maxl_th
```

PVSio Bells and Whistles

```
test : THEORY
BEGIN

  IMPORTING maxl_th,
            maxl_ax{{ maxl :=  maxl }}
END test
```

PVSio Bells and Whistles

```
test : THEORY
BEGIN

  IMPORTING maxl_th,
            maxl_ax{{ maxl :=  maxl }}

  main : void =
    println("Testing the function maxl") &
    let s = query_line("Enter a list of real numbers:") in
    let l = str2pvs[list[real]](s) in
    let m = maxl(l) in
      println("The max of "+s+" is "+m)
END test
```

Test It

```
<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, 5, 3, 2 :)
The max of (: -1, -2, 5, 3, 2 :) is 5

<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, -3, -4 :)
The max of (: -1, -2, -3, -4 :) is 0
```

Test It

```
<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, 5, 3, 2 :)
The max of (: -1, -2, 5, 3, 2 :) is 5

<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, -3, -4 :)
The max of (: -1, -2, -3, -4 :) is 0
```

Test It

```
<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, 5, 3, 2 :)
The max of (: -1, -2, 5, 3, 2 :) is 5

<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, -3, -4 :)
The max of (: -1, -2, -3, -4 :) is 0
```

Test It

```
<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, 5, 3, 2 :)
The max of (: -1, -2, 5, 3, 2 :) is 5

<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, -3, -4 :)
The max of (: -1, -2, -3, -4 :) is 0
```

Test It

```
<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, 5, 3, 2 :)
The max of (: -1, -2, 5, 3, 2 :) is 5

<PVSio> main;
Testing the function maxl
Enter a list of real numbers:
(: -1, -2, -3, -4 :)
The max of (: -1, -2, -3, -4 :) is 0
```

For the Emacs-Allergic: PVSio Applications

```
$ pvsio test:main
Testing the function maxl
Enter a list of real numbers:
(: 5, 4 ,3 ,2 :)
The max of (: 5, 4 ,3 ,2 :) is 5
```

PVSio and PVS

- PVSio safely enables the ground evaluator in the theorem prover.
- Ground expressions are translated into Lisp and evaluated in the PVS Lisp engine.
- The theorem prover only trusts the Lisp code automatically generated from PVS functional specifications.
- Semantic attachments are always considered harmful for the theorem prover.

The Strategy eval-formula

Evaluation of ground expressions via the ground evaluator:

```
  |-------
{1}   2 < sqrt_newton(2, 10) * sqrt_newton(2, 10)

Rule? (eval-formula 1)

Q.E.D.
```

The Strategy eval-formula

Evaluation of ground expressions via the ground evaluator:

```
  |-------
{1}   2 < sqrt_newton(2, 10) * sqrt_newton(2, 10)

Rule? (eval-formula 1)

Q.E.D.
```

The Strategy eval-formula

Evaluation of ground expressions via the ground evaluator:

```
  |-------
{1}   2 < sqrt_newton(2, 10) * sqrt_newton(2, 10)

Rule? (eval-formula 1)

Q.E.D.
```

Fast and Sound

Well, as Sound as the Lisp Engine

```
|-------
{1}   RANDOM /= RANDOM
Rule? (eval-formula 1)

Function stdmath.RANDOM is defined as a semantic attachment.
It cannot be evaluated in a formal proof.

No change on: (eval-formula 1)
```

References

- PVSio: `http://shemesh.larc.nasa.gov/people/cam/PVSio`.
- Rapid prototyping in PVS, Csar Muoz, NASA Contract Report, `http://hdl.handle.net/2060/20040046914`.
- Efficiently Executing PVS, N. Shankar, SRI Technical Report.
- Evaluating, Testing, and Animating PVS Specifications Judy Crow, Sam Owre, John Rushby, N. Shankar, and Dave Stringer-Calvert, SRI Technical Report, `http://www.csl.sri.com/users/rushby/abstracts/attachments`.