

Animation of Functional Specifications with PVSio

César A. Muñoz

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

PVS Class 2012



1

PVS as a Functional Programming Language

- Most specifications in PVS are functional, e.g.,

```
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
```

- What is the value of `sqrt_newton(2,10)`?

2

Animation

- ▶ Animation is the process of executing a specification to validate its intended semantics.
- ▶ Why: It is cheaper, faster, and more fun to test a specification than to prove it.
- ▶ How: **PVSio**.

3

PVSio is ...

- ▶ an *read-eval-loop* interface to the PVS Ground Evaluator;
- ▶ an efficient and sound mechanism to compute within the theorem prover;
- ▶ part of the PVS distribution;
- ▶ available as the standalone Unix command **pvsio** or through the Emacs command **M-x pvsio**.

4

```

+---
| PVSio-5.0 (10/11/12)
|
| Enter a PVS ground expression followed by a symbol ';'
| at the <PVSio> prompt.
| Enter a Lisp expression followed by a symbol '!'
| at the <PVSio> prompt.
|
| Enter help! for a list of commands and quit! to exit
| the evaluator.
|
| *CAVEAT*: evaluation of expressions which depend on
| unproven TCCs may be unsound, and result in the
| evaluator crashing into lisp, running out of stack,
| or worse. If you crash into lisp, type (restore) to
| resume.
|
+---

```

<PVSio>

5

```

<PVSio> sqrt_newton(2,10);
==>
1068540411258005424957730996202770251753061700886760050509
2775584086034866316307624567599571273090520553619648095761
8323863188053907381032775618232842813250031327063713965171
4658235752986741761590590866587906685398566655402811587051
1326582300341866167304359343960603343170658488116440998347
6684441998170083079481020253769836865387591260387081397004
4395397342728487283626639303583613156999614503895003382899
3710275557723330463738359457597728824912553479002609510283
8876667217608828542941439658998944011413943276015695324890
7732348479284448531263506286619985710653992842738259074138
7820229684450437162379033859897869490832422027208754929968
48471731623224703430657/7555721707723979449648392625272648
5551649317661664229348905711734284588432155133188346299073
5221751911561329826177804181498740064550632899687915531346
8554933480730789681235675239145730235553225685034903270487
8494636528771206171730831540313524856910300210308020618231
5299185952314636972146502157415893978546131789429918177417
2751814809171189000327511471361082396689939198281075027256
7020622390881915478979043267988775282465475161134045013900
7446848131012363204673416951245204209273923257266490795228
0624407194913939873541974150086297495497989151038331936924
2486058037295940522141109501799192580405952209313080346758
3186506109442752416009090650437487476983661568

```

6

```

<PVSio> print("sqrt_newton of 2: " + sqrt_newton(2,10));
sqrt_newton of 2: 1.4142135

<PVSio> printf("sqrt_newton of 2: ~f",sqrt_newton(2,10));
sqrt_newton of 2: 1.4142135*

<PVSio> print(sq(sqrt_newton(2,10)));
2.0

<PVSio> sq(sqrt_newton(2,10)) = 2.0;
==>
FALSE

<PVSio> sq(sqrt_newton(2,10));
==>
11417786104914273667156938719275144887173049593916168
1541156469937779509690251478413369...

```

*printf is available in PVS 6.0.

7

PVSio Capabilities

1. A predefined set of PVS functions for input/output operations, side-effects, unbounded-loops, exceptions, string manipulations, and floating point arithmetic
2. A high level interface for extending PVS programming language features.
3. A tool for rapid prototyping.
4. An efficient strategy for evaluating ground expressions.

PVSio Library: Input/Output Operations

```
<PVSio> let x = read_real in  
    print("sqrt("+x+") = "+sqrt_newton(x,10));  
10  
sqrt(10) = 3.1622777
```

9

PVSio Library: Floating Point Numbers 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 type-checking.

11

Extending PVSio Programming Features

- ▶ PVSio provides a “user-friendly” mechanism for extending the ground evaluator.
- ▶ Semantic attachments: **Lisp functions** attached to **uninterpreted PVS functions**.

12

User-defined Attachments

```
cubic_root : THEORY
BEGIN
  ...
  cubic(x:real) : real
  ...
END cubic_root
```

Create the file pvs-attachments in context directory:

```
;; File: pvs-attachments
(defattach cubic_root.cubic (x)
  "Cubic root of x"
  (expt x (/ 1 3))) ;;; <==== THIS IS LISP
```

In PVSio:

```
<PVSio> cubic(10);
==>
2.1544347
```

13

Rapid Prototyping

```
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
```

14

Add PVSio Code to Any PVS Theory

```
maxl_io : THEORY
BEGIN

  IMPORTING maxl_th

  test : void =
    println("Testing the function maxl") &
    LET s = query_line("Enter a list of real numbers: "),
        l = str2pvs[list[real]](s),
        m = maxl(l) IN
      println("The max of "+s+" is "+m)

END maxl_io
```

15

Animate It

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

Create PVSio Applications

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

17

PVSio and the PVS Theorem Prover

- ▶ 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.

18

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.

19

Fast and Furious

Well, as Sound as the PVS 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)

20

Ground Evaluation With (grind)

```
|-----  
{1} 2 < sqrt_newton(2, 10) * sqrt_newton(2, 10)  
  
Rule? (grind)  
sqrt_newton rewrites sqrt_newton(2, 10)  
  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))))  
  + ...
```

21

References

- ▶ Website:
<http://shemesh.larc.nasa.gov/people/cam/PVSio>.
- ▶ *Rapid prototyping in PVS*, C. Muñoz, NASA Contract Report.
- ▶ *Efficiently Executing PVS*, N. Shankar, SRI Technical Report.
- ▶ *Evaluating, Testing, and Animating PVS Specifications*, J. Crow, S. Owre, J. Rushby, N. Shankar, and D. Stringer-Calvert, SRI Technical Report.