Animation of Functional Specifications with PVSio

César A. Muñoz

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

PVS Class 2012

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)`?

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)`?

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.

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`.

```
+---
| 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>
```

```
<PVSio> sqrt_newton(2,10);
==>
106854041125800542495773099620277025175306170088676005050927755840860348663163076245675995712730905205536196480957618323863188053907381032775618232842813250031327063713965171
465823575298674176159059086658790668539856665540281158705113265823003418661673043593439606033431706584881164409983476684441998170083079481020253769836865387591260387081397004439539734272848728362663930358361315699961450389500338289937102755577233304637383594575977288249125534790026095102838876667217608828542941439658998944011413943276015695324890773234847928444853126350628661998571065399284273825907413878202296844504371623790338598978694908324220272087549299684847173162322470343065 7/7555721707723979449648392625272648555164931766166422934890571173428458843215513318834629907352217519115613298261778041814987400645506328996879155313468554933480730789681235675239145730235553225685034903270487849463652877120617173083154031352485691030021030802061823152991859523146369721465021574158939785461317894299181774172751814809171189000327511471361082396689939198281075027256702062239088191547897904326798877528246547516113404501390074468481310123632046734169512452042092739232572664907952280624407194913939873541974150086297495497989151038331936924248605803729594052214110950179919258040595220931308034675831865061094427524160090906504374874769836615687
```

```
<PVSio> sqrt_newton(2,10);
==>
106854041125800542495773099620277025175306170088676005050927755840860348663163076245675995712730905205536196480957618323863188053907381032775618232842813250031327063713965171465823575298674176159059086658790668539856665540281158705113265823003418661673043593439606033431706584881164409983476684441998170083079481020253769836865387591260387081397004439539734272848728362663930358361315699961450389500338289937102755577233304637383594575977288249125534790026095102838876667217608828542941439658998944011413943276015695324890773234847928444853126350628661998571065399284273825907413878202296844504371623790338598978694908324220272087549299684847173162322470343065 7/755572170772397944964839262527264855516493176616642293489057117342845884321551331883462990735221751911561329826177804181498740064550632899687915531346855493348073078968123567523914573023555322568503490327048784946365287712061717308315403135248569103002103080206182315299185952314636972146502157415893978546131789429918177417275181480917118900032751147136108239668993919828107502725670206223908819154789790432679887752824654751611340450139007446848131012363204673416951245204209273923257266490795228062440719491393987354197415008629749549798915103833193692424860580372959405221411095017991925804059522093130803467583186506109442752416009090650437487476983661568
```

```
<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));
==>
114177861049142736671569387192751448871730495939161681541156469937779509690251478413369...
```

*printf is avaialable in PVS 6.0.

```
<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));
==>
1.414778610491427366715693871927514488717304959391616815411564699377795096902514784133 69...
```

[†] printf is avaialable in PVS 6.0.

```
<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));
==>
114177861049142736671569387192751448717304959391616815411564699377795096902514784133 69...
```

‡ printf is avaialable in PVS 6.0.

```
<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));
==>
114177861049142736671569387192751448871730495939161681
541156469937779509690251478413369...
```

[§] printf is avaialable in PVS 6.0.

```
<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));
==>
1141778610491427366715693871927514887173049593916168
154115646993777950969025147841336 9...
```

¶printf is avaialable in PVS 6.0.

```
<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));
==>
114177861049142736671569387192751448871730495939161681541156469937779509690251478413369...
```

‖ printf is avaialable in PVS 6.0.

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
```

PVSio Library: Input/Output Operations

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

PVSio Library: Input/Output Operations

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

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
```

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
```

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
```

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.

Extending PVSio Programming Features

- PVSio provides a "user-friendly" mechanism for extending the ground evaluator.
- Semantic attachements: Lisp functions attached to uninterpreted PVS functions.

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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.

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 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)
```

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)
```

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))))
  + ...
```

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))))
  + ...
```

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.