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!
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} \( \text{sqrt\_newton}(2, 3) \leq 3 / 2 \)

Rule? (grind)
...

\text{sqrt\_newton} rewrites \text{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)*((1/2)*((1/2)* (1/2))
+ 3*((1/2)*(1/2)*((1/2)) + (1/2)*((1/2)*((1/2)*
+ (1/2)*(2/3))) + (1/2)*((1/2)*((1/2)*((1/2)*
+ (1/2)*(2/3))) + (1/2)*(1/2)*((1/2)*((1/2)*

Trying repeated skolemization, instantiation, and
if-lifting,
Q.E.D.
Grind is an Inefficient Calculator!

\[ \{1\} \quad 2 < \sqrt{\text{sqrt\_newton}(2, 10) \cdot \text{sqrt\_newton}(2, 10)} \]

Rule? (grind)

... 
\text{sqrt\_newton} rewrites \text{sqrt\_newton}(2, 4) 
\to (1/2) \cdot (2 / ((1/2) \cdot (2 / (3 \cdot ((1/2) \cdot (1/2)))) 
+ (1/2) \cdot (2/(3 \cdot (1/2) + (1/2) \cdot (2/3)))) 
+ (1/2) \cdot (1/2) \cdot (2/3)) 
+ 3 \cdot ((1/2) \cdot (1/2) \cdot (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 . . .

\[\text{\texttt{<GndEval> "sqrt_newton(2,3) \leq 3/2"}}\]
\[\Rightarrow\]
\[\text{TRUE}\]

\[\text{\texttt{<GndEval> "2 < sqrt_newton(2,10) \times sqrt_newton(2,10)"}}\]
\[\Rightarrow\]
\[\text{TRUE}\]
PVSio: A Rapid Prototyping Tool for PVS

... with a Poor Interface

<GndEval> "sqrt_newton(2,10)"

==>
10685404112580054249577309962027702517530617008867600050509277558408603486631630762456759957
127309052055361964809576183286318805390738103277561823284281325003132706371396517146582357
5298674176159059086658790668539856665540281158705113265823003418661673043593439606033431706
5848811644099834768644199817008307948102537698368653875912603870813970044395397342728487
283626639303583613156999614503895003382899371027557723330463738359457597728824912553479002
6095102838876667217608828542941439658998944011413943276015695324890773234847928444853126350
628661998571065399284273825907413878202296844504371623790338598978694908324202720875492996
848471731623224703430657/75557217077239794496483926252726485516493176616642293489057117342
84588432155131834629907352217519115613298261778041814987400645506328996879155313468554933
4807307896812356752391457302355532256850349032704878494636528771206171730831540313524856910
3002103080206182315299185952314636972146502157415893978546131789429918177417275181480917118
90003275114713610823966689939198281075025767026223908191547897904326798877528246547516113
4045013900744684813101236320467341695124520420927392325726649079522806244071949139398735419
741500862974954979891510383319369242860580372959405221411095017991925804059522093130803467
583186506109442752416009090650437487476983661568
Outline

PVSio

Semantic Attachments

Animation of Specifications

PVSio and PVS
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
Input Operations

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

10

sqrt(10)=3.1622777
```

sqrt(10)=3.1622777
Floating Points and a Random Surprise

\[ <\text{PVSio}> \text{SQRT}(2); \]
\[ \Rightarrow \]
\[ 1.4142135 \]

\[ <\text{PVSio}> \text{RANDOM} = \text{RANDOM}; \]
\[ \Rightarrow \]
\[ \text{FALSE} \]

\[ <\text{PVSio}> \text{let } r = \text{RANDOM} \text{ in } r = r; \]
\[ \Rightarrow \]
\[ \text{TRUE} \]
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 PVS Common Lisp machine.
- A user-friendly mechanism for extending the ground evaluator.
- Lisp functions attached to uninterpreted PVS functions.
User-defined Attachments

- **How:**
  ```lisp
  (defattach theory.name
    doc-string
    body)
  ```

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

- **Example:**
  ```lisp
  ;; 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

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
$ 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.
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:

