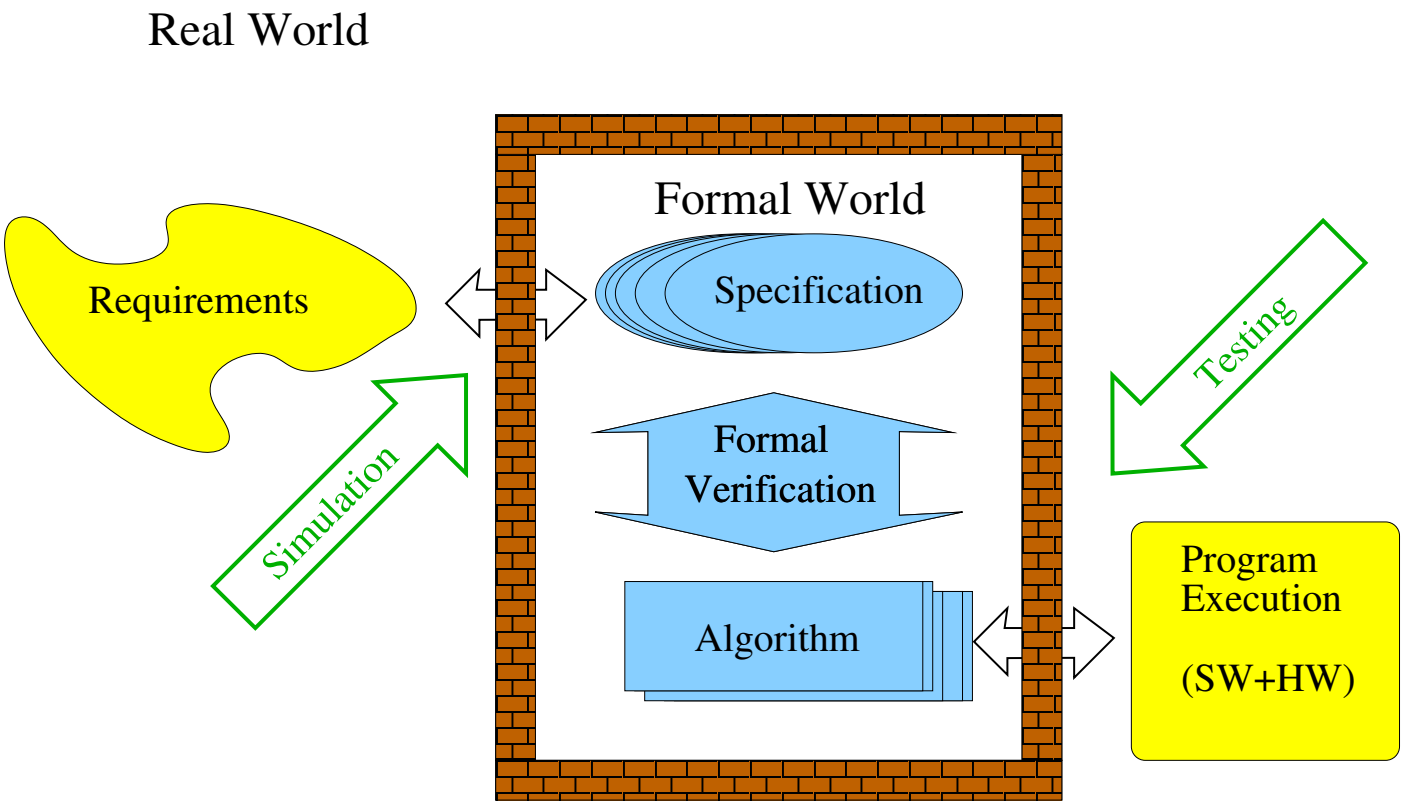
PVS for the Impatient

César A. Muñoz

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

The World According to PVS



All Models are Wrong

> *All models are wrong; the practical question is how wrong do they have to be to not be useful.*

G. Box and N. Draper, Empirical Model Building and Response Surfaces, 1987.

Prototype Verification System

- PVS (`http://pvs.csl.sri.com`) is developed by SRI International (`http://www.sri.com`).
- Strongly typed specification language based on classical higher-order logic.
- Powerful theorem prover.

Specification Language

- Classical logic: "To be or not to be" holds!
- Higher-order logic: Quantification over sets and functions.
- Strongly typed language: All declarations have to be explicitly typed.
  - Predicate subtyping.
  - Dependent records.
  - Abstract Data Types.
  - Co-inductive types.
- Undecidable type checking: Type of natural numbers n greater than 2, such that there exist a, b, c positive natural numbers, where $a^n + b^n = c^n$.

Mechanical Theorem Prover

- Interactive theorem prover.
- Decision procedures for several theories: propositional logic, linear arithmetic, finite state machines, equality with uninterpreted functions, etc.

Example: National Airspace System (NAS) Simplified Model

- Database that keeps information of every aircraft in the airspace.
- Operations:
  - Add an aircraft.
  - Delete an aircraft.
  - Find an aircraft.

Theories

A PVS *theory* is a collection of declarations and definitions of mathematical objects.

```
% National Airspace System
NAS : THEORY
BEGIN

   ...

END NAS
```

Parametric Theory

Theories in PVS model a parametric *family* of problems.

```
% National Airspace System
NAS[Identifier: TYPE+, Aircraft:TYPE+] : THEORY
BEGIN

    ...

END NAS
```

Definitions

PVS is a functional language (it doesn't have a notion of *state* or *memory*).

```
% National Airspace System
NAS[Identifier:TYPE+,Aircraft:TYPE+] : THEORY
BEGIN

  %% Type
  NAS : TYPE = ARRAY[Identifier->Aircraft]

  %% Variables
  nas : VAR NAS
  ac  : VAR Aircraft
  id  : VAR Identifier

END NAS
```

Specification of Operations

```
% Functional specification
find(nas,id) : Aircraft = nas(id)

% Axiomatic specification
add    : [[NAS,Identifier,Aircraft]->NAS]

add_ax : AXIOM
  find(add(nas,id,ac),id) = ac
```

Functions in PVS are total

```
NAC:Aircraft

emptynas : NAS

emptynas_ax : AXIOM
   find(emptynas,id) = NAC

remove : [[NAS,Identifier]->NAS]

remove_ax : AXIOM
  find(remove(nas,id),id) = ... ?
```

A Few Axioms

```
ex : AXIOM
  EXISTS (ac): ac /= NAC

remove_add : AXIOM
  remove(add(nas,id,ac),id) = nas

% These axioms make the theory inconsistent
myfalse : THEOREM
  1=0
```

Is This an Useful Model?

- Never use axioms!
  - Unless you really need them.
- How many aircraft are in the NAS?

```
length(nas) : nat = ...
```

Simple NAS Model

- Records and arrays.

```
NAS : TYPE = [# n : nat,
                ar : ARRAY[Identifier->Aircraft]
             #]
```

- Arrays are functions.

```
NAS : TYPE = [# n : nat,
                ar : [Identifier->Aircraft]
             #]
```

Record and Function Values

```
%% Record and function values
emptynas : NAS = (# n  := 0,
                    ar := LAMBDA(id): NAC
                 #)

%% Record and function access
find(nas,id) : Aircraft = nas`ar(id)
```

Record and Function Overriding

```
add(nas,id,ac) : NAS =
  nas WITH [ `n       := nas`n+1,
             `ar(id) := ac ]
```

Some Lemmas

```
emptynas_lem :LEMMA
   find(emptynas,id) = NAC

 add_lem : LEMMA
   find(add(nas,id,ac),id) = ac
```

Proof Assistant

```
emptynas_lem :

  |-------
{1}   FORALL (id: Identifier): find(emptynas, id) = NAC
Rule? (grind)
Q.E.D.

add_lem :

  |-------
{1}   FORALL (ac: Aircraft, id: Identifier, nas: NAS):
        find(add(nas, id, ac), id) = ac

Rule? (grind)
Q.E.D
```

Type Correctness Conditions

- In NAS.pvs

```
remove(nas,id) : NAS =
  nas WITH [`n := nas`n - 1,
            `ar(id) := NAC]
```

- In NAS.tcc

```
% Subtype TCC generated for  nas`n - 1
   % expected type  nat
 % unfinished
remove_TCC1: OBLIGATION FORALL (nas: NAS):
  nas`n - 1 >= 0;
```

Predicate Subtyping

- In NAS.pvs

```
nonemptynas?(nas) : bool =
 nas`n /= 0

remove(nas:{x:NAS|nonemptynas?(x)}NAS|nonemptynas?(nas)(no
  nas WITH [`n       := nas`n - 1,
            `ar(id) := NAC]
```

- In NAS.tcc

```
remove_TCC1: OBLIGATION FORALL (nas: (nonemptynas?)):
  nas`n - 1 >= 0;
```

More Lemmas

```
nnas : VAR (nonemptynas?)

rem_lem : LEMMA
  find(remove(nnas,id),id) = NAC

remove_add : LEMMA
  remove(add(nas,id,ac),id) = nas
```

```
remove_lem :

  |-------
{1}    FORALL (id: Identifier, nnas: (nonemptynas?)):
        find(remove(nnas, id), id) = NAC

Rule? (grind)
Q.E.D.
```

```
remove_add :

  |-------
{1}   FORALL (ac: Aircraft, id: Identifier, nas: NAS):
        remove(add(nas, id, ac), id) = nas

Rule? (grind)

  |-------
{1}   nas!1 WITH [`n := nas!1`n, `ar(id!1) := NAC] = nas!1

Rule? (decompose-equality)

  |-------
{1}   nas!1`ar WITH [(id!1) := NAC] = nas!1`ar
```

Ahhh ?

This sequent is unprovable:

```
  |-------
{1}  nas!1‘ar WITH [(id!1) := NAC] = nas!1‘ar
```

This formula holds only when

```
nas!1‘ar(id!1) = NAC
```

Is This an Useful Model?

- Is NAC really necessary?
- How to iterate on all aircraft in the NAS?
- Aren't there better data structure in PVS to model this problem?

More PVS Data Structures

Infinite collections

- Unbounded arrays: `ARRAY[nat->T]`, which is the same as `[nat->T]`.
- Sets: `set[T]`, which is the same as `[T->bool]`.

More PVS Data Structures

Finite collections

- Lists: `list[T]` (Generic ADT with constructors `null` and `cons`).
- Finite sets: `finite_set[T]` (A sub-type of `set[T]`).
- Bounded arrays: `ARRAY[subrange(1,n)->T]`.
- Sequences:

```
% Finite sequences from prelude
finseq: TYPE = [# length: nat,
                  seq: [below[length]->T] #]
% Padded sequences from NASA Libraries
fseq: TYPE = [# length: nat,
                seq: barray(length) #]
```

Final NAS Model

```
NAS : TYPE = [# n    : nat,
                seq  : ARRAY[subrange(1,n)->Aircraft],
                hash: ARRAY[Identifier->upto(n)]
             #]

emptynas : NAS = (# n    := 0,
                    seq  := LAMBDA(k:subrange(1,0)):
                                  (epsilon! (ac): true),
                    hash := LAMBDA(id):0
                 #)
```

Definitions: `find` and `add`

```
find(nas,id) : upto(nas`n) = nas`hash(id)

add(nas,id,ac) : NAS =
 LET i = find(nas,id) IN
   IF i > 0 THEN
     nas WITH [`seq(i) := ac]
   ELSE
     nas WITH [`n             := nas`n+1,
               `seq(nas`n+1) := ac,
               `hash(id)     := nas`n+1]
   ENDIF
```

Definitions: remove

```
remove(nas,id) : NAS =
  LET i = find(nas,id) IN
  IF i = 0 THEN nas
  ELSE nas WITH [`n    := nas`n-1,
                 `seq  := LAMBDA(k:subrange(1,nas`n-1)):
                            IF k < i THEN nas`seq(k)
                            ELSE nas`seq(k+1) ENDIF,
                `hash := LAMBDA(id) :
                           IF nas`hash(id) < i THEN
                             nas`hash(id)
                           ELSIF nas`hash(id) = i THEN
                             0
                           ELSE nas`hash(id)-1
                           ENDIF]
  ENDIF
```

Lemmas

```
emptynas_lem : LEMMA find(emptynas,id) = 0

add_exists : LEMMA exists?(id)(add(nas,id,ac))

add_lem : LEMMA
  LET newnas = add(nas,id,ac) IN
  newnas`seq(find(newnas,id)) = ac

remove_exists : LEMMA
  notexists?(id)(remove(nas,id))

remove_add : LEMMA
  remove(add(nas,id,ac),id) = nas
```

PVS Batch Proving Via `proveit`

```
$ proveit NAS

Processing ./NAS.pvs. Writing output to file ./NAS.summary
 Proof summary for theory NAS
    remove_add.............................unfinished
    Theory totals: 16 formulas, 16 attempted, 14 succeeded
Grand Totals: 16 proofs, 16 attempted, 14 succeeded (0.83 s)
```

PVS Bath Proving Via `proveit`

```
$ more NAS.summary
*** NAS (14:33:9 7/28/2011)
Proof summary for theory NAS
    add_TCC1.............................proved - complete
    add_TCC2.............................proved - complete
    add_TCC3.............................proved - complete
    emptynas_lem.........................proved - complete
    add_exists...........................proved - complete
    add_lem_TCC1.........................proved - complete
    add_lem..............................proved - complete
    remove_TCC1..........................proved - complete
    remove_TCC2..........................proved - complete
    remove_TCC3..........................proved - complete
    remove_TCC4..........................proved - complete
    remove_TCC5..........................proved - complete
    remove_TCC6..........................proved - complete
    remove_exists........................proved - complete
    remove_add...........................unfinished
   Theory totals: 15 formulas, 15 attempted, 14 succeeded (0.83 s)
Grand Totals: 15 proofs, 15 attempted, 14 succeeded (0.83 s)
```

Exercise 1

- Is this lemma provable?

```
remove_add : LEMMA
  remove(add(nas,id,ac),id) = nas
```

- If not, fix it and prove it in PVS.

Hint: You may need the following strategies (not necessarily in that order): `(skosimp :preds? t)`, `(decompose-equality)`, `(grind)`, `(typepred "<expr>")`, `(expand "<name>")`, and `(assert)`.

A More Concrete Specification

```
MyNAS : THEORY
BEGIN

  Identifier : TYPE = string

  Aircraft : TYPE = [#
    id  : Identifier,
    lat : {x:real | -90 <= x AND x <= 90},
    lon : {x:real | abs(x) < 360},
    alt : nnreal,  % [m]
    gs  : posreal, % Ground speed [m/s]
    vs  : real,    % Vertical speed [m/s]
    trk : {x:real | abs(x) < 360} % Track [True North]
  #]

...
```

```
IMPORTING NAS[Identifier,Aircraft]

ac1 : Aircraft = (#
  id  := "AC1",
  lat := 0, lon := 92.2, alt := 10000,
  gs  := 308, vs := 0, trk := -90
 #)

ac2 : Aircraft = (#
  id  := "AC2",
  lat := 0, lon := 93.5, alt := 15000,
  gs  := 300, vs := -200, trk := 90
#)
```

```
addac(nas:NAS,ac:Aircraft) : NAS =
  add(nas,ac`id,ac)

mynas : NAS = addac(addac(emptynas,ac1),ac2)

printac(ac:Aircraft): void =
  println(pvs2str(ac))

printnas(nas:NAS) : void =
  FORALL(i:subrange(1,nas`n)):
   printac(nas`seq(i))
```

Animation of Executable Specifications (via PVSio)

```
$ pvsio MyNAS

<PVSio> printnas(mynas);
(# alt := 10000,
   gs  := 308,
   lat := 0, lon := 461/5,
   id  := "AC1",
   trk := -90, vs := 0 #)
(# alt := 15000,
   gs  := 300,
   lat := 0, lon := 187/2,
   id  := "AC2",
   trk := 90, vs := -200 #)
```

```
<PVSio> printnas(remove(mynas,"AC1"));
(# alt := 15000,
   gs  := 300,
   lat := 0,
   lon := 187/2,
   id  := "AC2",
   trk := 90,
   vs  := -200 #)
```

Lesson Learned

Don't be Impatient!