# Specification, Verification, and Interactive Proof

Sam Owre

SRI International

May 23, 2016

# PVS

- PVS - Prototype Verification System
- PVS is a verification system combining language expressiveness with automated tools.
- It features an interactive theorem prover with powerful commands and user-definable strategies
- PVS has been available since 1993
- It has hundreds of users
- It is open source (GitHub)

# PVS Language

- The PVS language is based on higher-order logic (type theory)
- Many other systems use higher-order logic including Coq, HOL, Isabelle/HOL, Nuprl, Agda, Lean
- PVS uses classical (non-constructive) logic
- It has a set-theoretic semantics

# PVS Theories

- *Theories* contain declarations, importings
- Theories may be parameterized with types, constants, and other theories
- Theories and theory instances may be *imported*
- Theory interpretations may be given, using *mappings* to interpret uninterpreted types, constants, and theories
- Theories may have *assumptions* on the parameters
- Theories may state what is visible, through *exportings*

# Declarations

PVS supports a number of kinds of declarations
Declarations may themselves have formal parameters (types)

- Types
- Constants, definitions, macros
- Recursive definitions
- Inductive and coinductive definitions
- Formulas and axioms
- Assumptions on formal parameters
- Judgements, including recursive judgements
- Conversions
- Auto-rewrites

# Types

PVS has a rich type system

## Basic types:

`number`, `boolean`, etc. New basic types may be introduced

## Enumeration types:

`{red, green, blue}`

## Function, record, tuple, and cotuple types:

```
[number -> number]
[\# flag:\ boolean, value:\ number \#]
[boolean, number]
[boolean + number]
```

# Recursive Types

## Datatype

```
list[T: TYPE]: DATATYPE BEGIN
 null: null?
 cons(car: T, cdr: list): cons?
END DATATYPE
```

## Codatatype

```
colist[T: TYPE]: CODATATYPE BEGIN
 cnull: cnull?
 ccons(car: T, cdr: list): ccons?
END CODATATYPE
```

# Subtypes

PVS has two notions of subtype:

**Predicate Subtypes**

```
{x: real | x /= 0}
{f: [real -> real] | injective?(f)}
```

- The type `{x: T | P(x)}` may be abbreviated as `(P)`.

**Structural Subtypes**

```
[# x, y: real, c: color #] <: [# x, y: real #]
```

- Class hierarchy may be captured with this
- Update is structural subtype polymorphic:
  - `{r WITH [`x := 0]}`

# Dependent types

Function, tuple, record, and (co)datatypes may be dependent:

## Dependent Types

```
[n: nat -> {m: nat | m <= n}]
[n: nat, {m: nat | m <= n}]
[# n: nat, m: {k: nat | k <= n} #]

dt: DATATYPE BEGIN
 b: b?
 c(n: nat, d: {d: dt | d /= b}): c?
END DATATYPE
```

# Expressions

- Logic: `TRUE`, `FALSE`, `AND`, `OR`, `NOT`, `IMPLIES`, `FORALL`, `EXISTS`, `=`
- Arithmetic: `+`, `-`, `*`, `/`, `<`, `<=`, `>`, `>=`, `0`, `1`, `2`, ...
- Function application, abstraction, and update
- Binder macro: `the! (x: nat) p(x)`
- Coercions
- Record construction, selection, and update
- Tuple construction, projection, and update
- `IF-THEN-ELSE`, `COND`
- `CASES`: Pattern matching on (co)datatypes
- Tables

# Names

Names may be heavily overloaded

All names have an identifier; in addition, they may have:

- a theory identifier
- actual parameters
- a library identifier
- a mapping giving a theory interpretation

For example, a reference to "a" may internally be equivalent to the form

**Expanded name**

```
lib@th[int, 0][nat]{{T := real, c := 1}}.a
```

# Prover

- The PVS prover is interactive, but with powerful automation
- It supports exploration, design, implementation, and maintenance of proofs
- The prover was designed to preserve correspondence with an informal argument
- Support for user defined strategies and rules
- Based on the sequent calculus

# PVS Libraries

- Any directory containing PVS files (`.pvscontext`) may be used as a library
- Libraries may contain `pvs-strategies`, `pvs-lib.lisp`, and `pvs-lib.el` files - automatically loaded
- `PVS_LIBRARY_PATH` is a colon-separated set of paths to search
- `M-x load-prelude-library` has the effect of extending the prelude so that the library theories do not need to be explicitly imported

# NASA Libraries

NASA has a large and growing set of libraries at (google for `nasalib`)

About 140,000 lines of PVS code, over 23,000 theorems (including TCCs)

| | |
|---|---|
| affine_arith | a model for self-validated numerical analysis |
| algebra | groups, monoids, rings, etc |
| analysis | real analysis, limits, continuity, derivatives, integrals |
| analysis_ax | fundamental_theorem, integrals, taylor expansions, chain_rule |
| Bernstein | solver for polynomial inequalities |
| calculus | axiomatic version of calculus |
| complex | complex numbers |
| co_structures | sequences of countable length defined as coalgebra datatypes |

# NASA Libraries

| | |
|---|---|
| digraphs | directed graphs: circuits, maximal subtrees, paths, dags |
| fault_tolerance | consensus protocols, clock synchronization |
| float | floating point numbers and arithmetic |
| graphs | graph theory: connectedness, walks, trees, Menger's Theorem |
| ints | integer division, gcd, mod, prime factorization, min, max |
| interval_arith | interval bounds and numerical approximations |
| lnexp | logarithm, exponential and hyperbolic functions |
| lnexp_fnd | foundational definitions of logarithm, exponential, and hyperbolic functions |
| matrices | inverse, determinants, upper-triangular,diagonal |

# NASA Libraries

| | |
|---|---|
| MetiTarski | PVS with MetiTarski |
| numbers | primes, sqrt(2) irr, chinese remainder, Fermats little theorem |
| orders | abstract orders, lattices, fixedpoints |
| power | roots, powers and generalized logs |
| reals | summations, sup, inf, sqrt over the reals, abs lemmas |
| scott | Theories for reasoning about compiler correctness |
| series | power series, comparison test, ratio test, Taylor's theorem |
| sets_aux | powersets, orders, cardinality over infinite sets |
| sigma_set | summations over countably infinite sets |
| structures | bounded arrays, finite sequences and bags |
| Sturm | Sturm sequences |
| Tarski | generalizes Sturm |

# NASA Libraries

| | |
|---|---|
| topology | continuity, homeomorphisms, connected and compact spaces, Borel sets/functions |
| trig | trigonometry: definitions, identities, approximations |
| trig_fnd | foundational development of trigonometry: proofs of trig axioms |
| vectors | basic properties of vectors |
| vect_analysis | vector analysis |
| while | semantics for the programming language "while" |