

A short introduction to PVS

Proving with computer assistance

Francien Dechesne

February 2006

<http://www.win.tue.nl/~fdechesn/2IF48>

f.dechesne@tue.nl

HG 7.17



Introduction

How does it work?

Demo

Final assignment



What is PVS?



(Typus arithmeticae, 1535)

PVS: Prototype Verification System

Developed by organization *SRI International*

- ▶ Formal specification language
- ▶ Model checker
- ▶ Theorem prover
- ▶ Documentation tools, etc.



Applications of PVS

Both academic and industrial:

- ▶ Verification of Javacard applets (LOOPtool: Nijmegen)
- ▶ Hardware verification (microprocessors)
- ▶ Protocol verification
- ▶ Protocol specification
- ▶ Formal Mathematics (coalgebraic datatypes, Bart Jacobs)
- ▶ Safety-critical systems (NASA)
- ▶ ... (see <http://pvs.csl.sri.com/users.shtml>)

In PVS more logic and notions are predefined than in Coq. This can be both an advantage and a disadvantage.



Example: proving correctness of JavaCard applets

Former project at Radboud University Nijmegen. This involved:

- ▶ Formalizing semantics of JavaCard in PVS (Hoare triples)
- ▶ Translating JavaCard programs into PVS
- ▶ Translating and prove properties in PVS
- ▶ Defining strategies for automatic proving.

Web Link: <http://www.cs.ru.nl/VerifiCard/>

Practical aspects:

- ▶ Free download from SRI: <http://pvs.cs1.sri.com/>.
- ▶ Latest version (available on svstud): 3.2
- ▶ Distributions for Solaris/Linux (from a Windows computer: access svstud using Exceed)
- ▶ Emacs interface



The system

- ▶ Text files (*.pvs) with theories: definitions, axioms, statements; edit in Emacs
- ▶ Interactively created proofs (*.prf) (possibly more proofs for one lemma; saved as LISP-datastructure.)
- ▶ Implementation language: LISP (more than 50.000 lines!) (in previous version, $1=0$ has been proved. . .)



The specification language and logic of PVS

- ▶ Some extension of typed λ -calculus (allowing for the definition of *subtypes*)
- ▶ *classical* higher-order logic



Types

"Only" three levels:

objects: types: Type

(Type is as \star in PVS, not as \square like in Coq)

No abstraction over Type:

Type \rightarrow Type

is not allowed (no type constructors, no fourth level \square)



Recursive functions

```
sum(n:nat): RECURSIVE nat =
  If n=0
  Then 0
  Else n+sum(n-1)
endif
MEASURE n
```



Typechecking

- ▶ To check that `sum` has type `nat \rightarrow nat`, we must check that

if `n/=0` then `n-1>=0`

if `n/=0` then `n-1<n`

- ▶ For complicated functions it is not always possible to check this automatically.
- ▶ Also, declarations of the form `A:Type+` imply inhabitation.

Typechecking in PVS is not decidable!



TCCs

- ▶ TCC=Type Correctness Condition
- ▶ Statements that must hold in order to prove that a term has a given type.
- ▶ Generated when using:
 - Recursive definitions.
 - Subtypes
- ▶ Most of them can be automatically proven



Final assignment: procedure

Detailed instructions available at

<http://www.win.tue.nl/~fdechesn/2IF48/>

- ▶ Form a group of 2 or 3 students.
- ▶ Choose one exercise from the list of choices (A-E).
- ▶ **Before March 1st:** register your group and choice.
- ▶ Execute the exercise and write a report
- ▶ Deadline: **May 12, 2006**.
- ▶ Examination/evaluation: discussion in presence of the whole group, of the code and the report (by either Adam Koprowski or Francien Dechesne).
- ▶ Final grade for 2IF40: average of grades for theoretical part and final assignment.



Final assignment: recommendations

- ▶ Try to use the existing Coq modules (like `Arith`, `List`)
- ▶ Make active use of the online documentation, try out tactics we have not discussed yet.
- ▶ Be careful with expanding definitions (easily makes things unreadable)
- ▶ Divide the proofs into small steps: formulate sublemmas.
- ▶ Try to keep understanding what the hypotheses and goals *mean*
- ▶ For the requirements on the report: see webpage.
- ▶ Don't postpone!

