Coq summary + Short introduction to PVS

Proving with computer assistance

Adam Koprowski

February 2007

http://www.win.tue.nl/~akoprows/teaching/Coq
A.Koprowski@tue.nl
HG 6.78
technische universiteit eindhoven

Some more advanced features of Coq:

- Records.

```
Record Rat : Set := mkRat {
  sign:        bool;
  top:         nat;
  bottom:      nat;
  bottom_cond: 0 <> bottom
}.
```

- Coinductive definitions.

```
CoInductive Stream : Set :=
| Seq: nat -> Stream -> Stream.
```

- User-defined equalities (Setoid).

```
Definition mset := list A.
Definition list_permut (l l': list A) := ...
Definition meq (M N: mset) := list_permut M N.
Lemma meq_eq: Setoid_Theory mset meq.
Add Setoid mset meq meq_eq as meq_sid.
```

- Implicit arguments.

```
 Set Implicit Arguments.
Definition perm (A:Set)(l l':list A) := ...
Goal perm nat (3::nil) (4::nil).
```

- User-defined notations.

```
Definition list2multiset (l: list A) : mset :=
Notation "M =m= N" := (meq M N) (at level 70).
Notation "{{ M }}" := (list2multiset M).
Goal forall M N, {{ M }} =m= {{ N }}.
```

- Sections and module system.
  Sections are used for abstraction.
  Modules to modularize development and to make parameterized theories. They are based on *ocaml* module system.
- `ring` and `omega` solvers.
  Powerful tactics to deal with ring and field equations and Presburger arithmetic, respectively.
- `auto` hint database.
  `auto` is capable of “learning” by means of user providing a database of lemmas to be used.
- Extraction.
  Mechanism allowing to obtain certified programs out of constructive proofs.

Prime Number Theorem

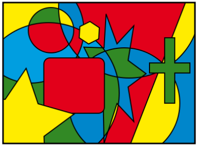
- Theorem:

$$\lim_{x \to \infty} \frac{\pi(x) \ln x}{x} = 1 \qquad (\pi(x) \sim \frac{x}{\ln x})$$

$$\pi(x) = |\{n : \mathbb{N} \mid n \leq x \wedge \mathrm{prime}(x)\}|$$

- Formalized by:
  - Jeremy Avigad (Carnegie Mellon University, Pittsburgh, PA, USA),
  - with help of: Kevin Donnely, David Grey and Paul Raff
- System: Isabelle
- Size: 0.97MB, 29.753 lines

Four Color Theorem



- Formalized by:
  - Georges Gonthier (Microsoft Research, Cambridge, UK)
  - Benjamin Werner (Ecole Polytechnique, Paris, FR)
- System: Coq
- Size: 2.50MB, 60.103 lines (about one third generated automatically)
- First major theorem proven with help of computers!... A paraphrased comment of the time:

> *A good mathematical proof is like a poem — this is a telephone directory!*

Four Color Theorem

```
Variable R : real_model.
Inductive point : Type := Point : (x, y: R) point.
Definition region : Type := point -> Prop.
Definition map : Type := point -> region.
Record proper_map [m: map] : Prop := ProperMap {
  map_sym : (z1, z2: point) m z1 z2 -> m z2 z1;
  map_trans : (z1, z2: point) m z1 z2 -> subregion (m z2) (m z1)
}.
Record simple_map [m: map] : Prop := SimpleMap {
  simple_map_proper :> proper_map m;
  map_open : (z: point) open (m z);
  map_connected : (z: point) connected (m z)
}.
Record coloring [m, k : map] : Prop := Coloring {
  coloring_proper :> proper_map k;
  coloring_inmap : subregion (inmap k) (inmap m);
  coloring_covers : covers m k;
  coloring_adj : (z1, z2 : point) k z1 z2 -> adjacent m z1 z2 -> m z1 z2
}.
Definition map_colorable [nc: nat; m: map] : Prop :=
   (EXT k | coloring m k & size_at_most nc k).

Theorem four_color : forall m: (map R),
  simple_map m -> map_colorable 4 m.
```

Jordan Curve Theorem

- Formalized by:
  - Tom Hales (University of Pittsburgh, PA, USA)
- System: HOL Light
- Size: 2.15MB, 75.242 lines

Flyspeck project

- Goal: verifying Kepler Conjecture.
- Estimated 20 man-years to complete.
- More people involved.
- Use of other theorem provers (Coq, Isabelle).

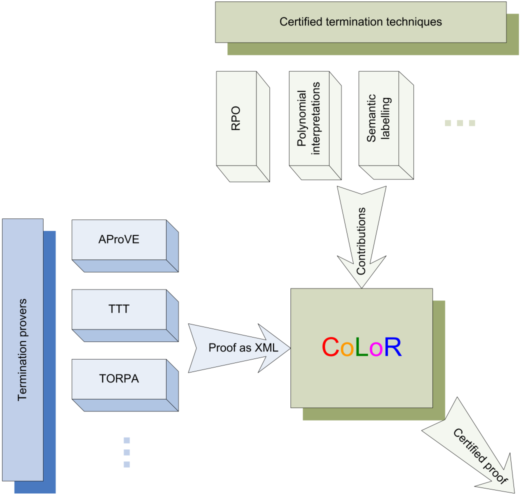
Certification for proving termination of term rewriting

- Termination is an important topic in term rewriting.
- There are tools for proving termination automatically.

CoLoR

CoLoR: a Coq library on rewriting and termination
`http://color.loria.fr`

- Objective: formalization of theory of term rewriting in the theorem prover Coq.
- Ultimate objective: certification of termination proofs produced by tools for proving termination of rewriting.


Certified termination techniques
RPO
Polynomial interpretations
Semantic labelling
...
Contributions
Termination provers
AProVE
TTT
TORPA
Proof as XML
CoLoR
Certified proof

CoLoR

- Contributors:
  - Frédéric Blanqui (INRIA, France),
  - Solange Coupet-Grimal, William Delobel (Université de Provence Aix-Marseille I, France)
  - Sébastien Hinderer (LORIA, France)
  - Adam Koprowski (Eindhoven University of Technology, Netherlands)
  - Stéphane Le Roux (ENS Lyon, France)
- System: Coq
- Size: 1.1 MB, 38.145 lines
- All contributions are welcome!

Freek Wiedijk
The Seventeen Provers of the World.
Springer, Lecture Notes in Artificial Intelligence, vol 3600, 2006

PVS

PVS

PVS: Prototype Verification System

`http://pvs.csl.sri.com`

Specification and verification system consisting of:

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

PVS: practical aspects

- Downloadable from `http://pvs.csl.sri.com` (Windows not supported!).
- Last version: 4.0.
- Available on `svstud` (via Exceed).
- Emacs interface.

PVS: the system & its logic

PVS: the system

- Implemented in LISP (more than 50.000 lines).
- Theories written and edited in text files (`*.pvs`).
- Proofs created interactively and saved as LISP data-structure (`*.prf`).

PVS: the logic

- Based on extensions to typed λ-calculus
- and *classical*, typed higher-order logic.
- Extensions allow for subset types.

Unlike Coq, PVS does not have small kernel (de Bruijn principle) and indeed in previous versions $0 = 1$ has been proven.

PVS types

- Type variables: `T : Type`, `T : Type+`.
- Base types: `bool`, `nat`, `real`.
- Abstract data-types: `Stack`, `List`, `Tree`.
- Function types: `[int, nat -> bool]`.
- Enumeration types: `{red, green, blue}`.
- Tuple types: `[A, B]`.
- Dependent record types: `[# a : A, b : B(b) #]`.
- Subset types: `{i : nat | i > 1}`.

PVS expressions

- Basic expressions:
  `TRUE : bool`   `0, 23 + 5, 17 * 10 : int`
- Function abstraction and application:
  `(LAMBDA (i, j : nat) : i + j) : [nat, nat -> nat]`
  `f(i, j)`
- Logic:
  `AND, OR, NOT, IMPLIES, IFF, =, /=, FORALL, EXISTS`
- Conditionals:
  `IF c THEN` e_1 `ELSE` e_2 `ENDIF`
- Records:
  `point WITH [`x := 24]`
- Subtypes:
  `Interval(m, n : int) : TYPE = {i : int | m <= i <= n}`
  `/ : [int, {n : int | n /= 0} -> int]`

PVS recursive definitions

$$\mathit{sum}(n : \mathit{nat}) : \mathit{RECURSIVE}\ \mathit{nat} =$$
$$(\mathit{IF}\ n = 0\ \mathit{THEN}\ 0\ \mathit{ELSE}\ n + \mathit{sum}(n - 1)\ \mathit{ENDIF})$$
$$\mathit{MEASURE}\ n$$

This recursive definition generates two type checking conditions:

- for type-consistency: $\mathtt{IF\ n /= 0\ THEN\ n - 1 >= 0}$
- for termination: $\mathtt{IF\ n /= 0\ THEN\ n - 1 < n}$

Such conditions are called TCCs (Type Checking Conditions). They:

- are generated for recursive definitions and subtypes and
- most of them can be automatically discarded by PVS.

Type-checking in PVS is not decidable!

PVS theories

- PVS developments are organized into theories.
- Theories consist of definitions, declarations, axioms and lemmas.
- Theories can be parameterized.
- Prelude contains a (large) number of predefined theories.

$$
\begin{array}{l}
\mathit{wf\_induction}\,[T : \mathit{TYPE},\ <: (\mathrm{well\_founded?}[T])] : \mathit{THEORY} \\
\quad \mathit{BEGIN} \\
\qquad \mathit{wf\_induction} : \mathit{LEMMA} \ldots \\
\quad \mathit{END}\ \mathit{wf\_induction}
\end{array}
$$

PVS sequents

PVS proof obligations are presented as sequents:

```
 {-1}   A
 {-2}   B
 {-3}   C
|-----------
 {1}    S
 {2}    T
```

The sequent stands for: $A \wedge B \wedge C \implies S \vee T$ with:

- negatively numbered ancedents/assumptions above the line,
- positively numbered consequents/goals below the line.
- PVS maintains a proof tree of such sequents.

Comparison of tactics

Coq	PVS
intro, intros	(flatten), (skolem)
apply	(lemma), (use)
unfold	(expand)
simpl	(beta), (simplify)
induction	(induct), (induction-and-simplify)
auto, tauto	(grind), (prop), (assert)
rewrite	(rewrite), (replace)
Undo	(undo)

PVS demo

PVS demo

Acknowledgments to Erik Poll.
This PVS introduction (and the demo file) are based on his one-day introduction course given in Eindhoven during IPA days.

`http://www.cs.ru.nl/~erikpoll/Teaching/PVS/index.html`

Final assignment: procedure

- Form a group of 2-3 students.
- Choose an exercise from the list of choices (A-E).
- Before **March 15** register your group and your choice.
- Solve the exercise and write a report.
- You may ask for assistance but this will have an influence on your grade.
- Submit your solution before **May 11**.
- Final evaluation: short meeting and discussion of the code and the report.
- Final grade for 2IF40: average of grades for theoretical part and final assignment.

Final assignment: hints

Final assignment:

- Try to use existing Coq modules (like `Arith`, `List` etc.)
- Explore the documentation; try things not discussed during the course.
- Split proof into smaller parts – careful choice of lemmas is often half of the success!
- Understand what you are doing.
- Do not wait till the very last moment with solving the exercise!

Final assignment: hints

Report:

- Keep report short (max. 15 pages).
- Do not repeat the code or write obvious remarks.
- Rather write about:
  - the problems you encountered,
  - solutions to them (along with alternative solutions and reasons for choosing the one used),
  - your experience with proving in Coq.
- See course web page for more details.