

SHARING PROOFS BETWEEN COQ AND PVS

Florent Kirchner

Logical Project

Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École
Polytechnique, INRIA, Université Paris-Sud

June 2006

COQ AND PVS, SITTING IN A TREE...

COQ Logical team at INRIA - Futurs

- natural deduction
- intuitionistic CIC
- light automation
- mathematics

PVS SRI's Computer Science Laboratory

- sequent calculus
- classical HOL with subtypes
- heavily automated
- engineering

COQ AND PVS, SITTING IN A TREE...

COQ Logical team at INRIA - Futurs

- natural deduction
- intuitionistic CIC
- light automation
- mathematics

PVS SRI's Computer Science Laboratory

- sequent calculus
- classical HOL with subtypes
- heavily automated
- engineering

SHARE core logics (HOL), tactic languages (LCF), libraries (\mathbb{R} ...)

... K-I-S-S-I-N-G

```
Variable A:Set
Variable R:A->A->Prop      assymetric, transitive.
Variable Eq:A->A->Prop.
Variable f:A->A            increasing.
Variable M:A              least upper bound.
```

Theorem Tarski : (Eq M (f M)).

Proof.

```
cut (R M (f M)).
```

```
repeat intros ; apply.
```

Qed.

STATING THE PROBLEM

How do we deal with...

- similar (universal?) formal developments

STATING THE PROBLEM

How do we deal with. . .

- similar (universal?) formal developments
- very different levels of automation

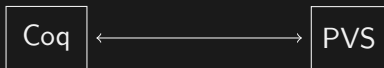
STATING THE PROBLEM

How do we deal with. . .

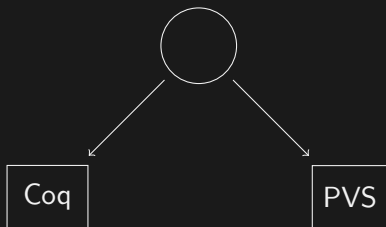
- similar (universal?) formal developments
- very different levels of automation
- multiple proof languages with disparate semantics

?

DRAFTING A SOLUTION



DRAFTING A SOLUTION



DRAFTING A SOLUTION

- An independent system of propositions and proofs
- Translation into both provers
- A proof manager to write and test proofs

DRAFTING A SOLUTION

- An independent system of propositions and proofs
- Translation into both provers
- A proof manager to write and test proofs

- Nicolaas de Bruijn's postulates:

Provers and their formalisms should stay simple.

People will never agree on the logics: we should to give them the choice.

FIRST ORDER SEQUENT CALCULUS

$$\text{cut}_{\mathcal{L}} \frac{x : A, \Gamma \vdash * : B; \Delta \quad x : A, \Gamma; * : B \vdash \Delta}{\Gamma; * : A \vdash \Delta}$$

$$\text{cut}_{\mathcal{R}} \frac{\Gamma \vdash * : B; x : A, \Delta \quad \Gamma; * : B \vdash x : A, \Delta}{\Gamma \vdash * : A; \Delta}$$

$$\text{imply}_{\mathcal{L}} \frac{\Gamma \vdash * : A; \Delta \quad \Gamma; * : B \vdash \Delta}{\Gamma; * : A \Rightarrow B \vdash \Delta}$$

$$\text{imply}_{\mathcal{R}} \frac{\Gamma, x : A \vdash * : B, \Delta}{\Gamma \vdash * : A \Rightarrow B; \Delta}$$

$$\text{forall}_{\mathcal{L}} \frac{\Gamma; * : B[x \leftarrow t] \vdash \Delta}{\Gamma; * : \forall(x : A)B \vdash \Delta}$$

$$\text{forall}_{\mathcal{R}} \frac{\Gamma \vdash * : B; \Delta}{\Gamma \vdash * : \forall(x : A)B; \Delta}$$

FIRST ORDER SEQUENT CALCULUS + $\bar{\lambda}\mu\tilde{\mu}$

$$\text{cut}_{\mathcal{L}} \frac{x : A, \Gamma \vdash v : B; \Delta \quad x : A, \Gamma; e : B \vdash \Delta}{\Gamma; \tilde{\mu}x.\langle v|e \rangle : A \vdash \Delta}$$

$$\text{cut}_{\mathcal{R}} \frac{\Gamma \vdash v : B; \alpha : A, \Delta \quad \Gamma; e : B \vdash \alpha : A, \Delta}{\Gamma \vdash \mu\alpha.\langle v|e \rangle : A; \Delta}$$

$$\text{imply}_{\mathcal{L}} \frac{\Gamma \vdash v : A; \Delta \quad \Gamma; e : B \vdash \Delta}{\Gamma; v \cdot e : A \Rightarrow B \vdash \Delta}$$

$$\text{imply}_{\mathcal{R}} \frac{\Gamma, x : A \vdash v : B; \Delta}{\Gamma \vdash \lambda x.v : A \Rightarrow B; \Delta}$$

$$\text{forall}_{\mathcal{L}} \frac{\Gamma; e : B[x \leftarrow t] \vdash \Delta}{\Gamma; t \cdot e : \forall(x : A)B \vdash \Delta} \quad \text{forall}_{\mathcal{R}} \frac{\Gamma \vdash v : B; \Delta}{\Gamma \vdash \lambda x.v : \forall(x : A)B; \Delta}$$

FIRST ORDER SEQUENT CALCULUS + $\bar{\lambda}\mu\tilde{\mu}$

- There is always an active formula
- As many multiplicative rules as possible
- 4 primitive tactics: axiom, cut, elim, weaken

FIRST ORDER SEQUENT CALCULUS + $\bar{\lambda}\mu\tilde{\mu}$

- There is always an active formula
- As many multiplicative rules as possible
- 4 primitive tactics: axiom, cut, elim, weaken

- Multiple options: intuitionistic / classical, minimal / full
- Proof terms

ASSUMING FOL

- Not that expressive, but powerful enough
- Adhoc implementation, via the theory of classes
- More on this on Tuesday

GIVING PEOPLE THE CHOICE... OF PROVERS

- Cannot rely on tactic translation

GIVING PEOPLE THE CHOICE... OF PROVERS

- Cannot rely on tactic translation
- Proposition translation, proof of logical rule correctness

GIVING PEOPLE THE CHOICE... OF PROVERS

- Cannot rely on tactic translation
- Proposition translation, proof of logical rule correctness
- Checkout the proofs into Coq or PVS

GIVING PEOPLE THE CHOICE... OF PROVERS

- Cannot rely on tactic translation
- Proposition translation, proof of logical rule correctness
- Checkout the proofs into Coq or PVS
- Export to natural language (C.S.C.)

GENERATING PROOF SCRIPTS

- Goals and specifications are translated

GENERATING PROOF SCRIPTS

- Goals and specifications are translated
- Sequents are flattened:

$$A_1 \dots A_n \vdash B_1 \dots B_n \rightsquigarrow A_1 \wedge \dots \wedge A_n \Rightarrow B_1 \vee \dots \vee B_n$$

GENERATING PROOF SCRIPTS

- Goals and specifications are translated
- Sequents are flattened:

$$A_1 \dots A_n \vdash B_1 \dots B_n \rightsquigarrow A'_1 \wedge \dots \wedge A'_n \Rightarrow B'_1 \vee \dots \vee B'_n$$

GENERATING PROOF SCRIPTS

- Goals and specifications are translated
- Sequents are flattened:

$$A_1 \dots A_n \vdash B_1 \dots B_n \rightsquigarrow A'_1 \wedge \dots \wedge A'_n \Rightarrow B'_1 \vee \dots \vee B'_n$$

- Each logical rule application is proven:

$$\frac{S_2}{S_1} \rightsquigarrow \frac{(S'_2 \Rightarrow S'_1) \wedge S'_2}{S'_1}$$

WORKING WITH PROOF TERMS

- Plain λ -term generation.
- Extract proof scripts for $\bar{\lambda}\mu\tilde{\mu}$ -terms.

$\llbracket x \rrbracket = \text{exact } x$

$\llbracket \alpha \rrbracket^{H:T} = \text{exact } H$

$\llbracket \top \rrbracket = \text{constructor}$

$\llbracket \perp \rrbracket^{H:T} = \text{contradiction}$

$$\llbracket \mu\alpha. \langle v : T | e \rangle \rrbracket = \text{cut } T ; [\text{intro } H ; \llbracket e \rrbracket^{H:T} \mid \llbracket v \rrbracket]$$

$$\llbracket v \cdot e \rrbracket^{H:T' \rightarrow T''} = \text{cut } T'' ; [\text{intro } H' ; \llbracket e \rrbracket^{H':T''} \mid \text{apply } H ; \llbracket v \rrbracket]$$

$$\llbracket \tilde{\mu}x. \langle v : T | e \rangle \rrbracket^{H:T'} = \text{rename } H \ x ; \text{cut } T ; [\text{intro } H' ; \llbracket e \rrbracket^{H':T'} \mid \llbracket v \rrbracket]$$

$$\llbracket [e, e'] \rrbracket^{H:T' \vee T''} = \text{destruct } H \text{ as } [H' | H''] ; [\llbracket e \rrbracket^{H':T'} \mid \llbracket e \rrbracket^{H'':T''}]$$

$$\llbracket \text{inj}_l v \rrbracket = \text{left} ; \llbracket v \rrbracket$$

$$\llbracket \text{inj}_r v \rrbracket = \text{right} ; \llbracket v \rrbracket$$

$$\llbracket \text{proj}[x, y, \langle v | e \rangle] \rrbracket = \text{destruct } H \text{ as } [x \ y] ; \llbracket \mu\alpha. \langle v | e \rangle \rrbracket$$

$$\llbracket (v, v') \rrbracket = \text{split} ; [\llbracket v \rrbracket \mid \llbracket v' \rrbracket]$$

$$\llbracket \lambda x. v \rrbracket = \text{intro } x ; \llbracket v \rrbracket$$

$$\llbracket (t, v) \rrbracket = \text{exists } t ; \llbracket v \rrbracket$$

$$\llbracket t \cdot e \rrbracket^{H=\forall x:T'.T''} = \text{cut } T'' [x \leftarrow t] ; [\text{intro } H' \mid \text{apply } H] ; \llbracket e \rrbracket^{H':T''} [x \leftarrow t]$$

$$\llbracket \text{proj}[x, y, \langle x | e \rangle] \rrbracket^{H:\exists x:T'.T''} = \text{destruct } H \text{ as } [x \ y] ; \llbracket e \rrbracket^{y:T''}$$

THE PROOF MONAD

- π is the representation of a proof tree
- `tactic` :: $\pi \rightarrow \mathcal{M}\pi$
- Integer n represents the number of subgoals
- String s stores the description of the exception

```
type  $\mathcal{M}\Pi$  = success  $\pi$   
           | subgoals  $n \pi$   
           | exception  $s \pi$ 
```

THE PROOF MONAD

- $unit :: \Pi \rightarrow \mathcal{M}\Pi$
 $unit \pi = subgoals\ 0\ \pi$

THE PROOF MONAD

- $unit :: \Pi \rightarrow \mathcal{M}\Pi$
 $unit \pi = subgoals\ 0 \ \pi$
- $(\star) :: \mathcal{M}\Pi \rightarrow (\Pi \rightarrow \mathcal{M}\Pi) \rightarrow \Pi$
 $m \star t = \text{case } m \text{ of}$
 - | $success \ \pi \Rightarrow success \ \pi$
 - | $subgoals \ i \ \pi \Rightarrow f \ i \ (t \ \pi)$
 - | $exception \ s \ \pi \Rightarrow exception \ s \ \pi$

THE PROOF MONAD

- $unit :: \Pi \rightarrow \mathcal{M}\Pi$
 $unit \pi = subgoals\ 0 \pi$
- $(\star) :: \mathcal{M}\Pi \rightarrow (\Pi \rightarrow \mathcal{M}\Pi) \rightarrow \Pi$
 $m \star t = \text{case } m \text{ of}$
 - | $success \pi \Rightarrow success \pi$
 - | $subgoals\ i \pi \Rightarrow f\ i\ (t \pi)$
 - | $exception\ s \pi \Rightarrow exception\ s \pi$
- $(idtac)(\pi) \equiv unit \pi$
 $(step1 ; step2)(\pi) \equiv step1(\pi) \star step2$

THE PROOF MONAD

- Solid foundation for proof languages
 - Monadic operators
 - Monad destructors
- Used to wrap up the sequent calculus
- Basis for PVS# (C.M.)

PVS#: THE CASE OF TRY

- $|(SKIP)| = skip$
- $|(FAIL)| = failure$
- $|(TRY A B C)| =$

{	$ C $	if $ A \in \{skip, backtracking\}$
	$ A $	if $ A \in \{failure, success\}$
	$backtracking$	if $ A = subgoals, B \in \{failure, backtracking\}$
	$subgoals$	if $ A = subgoals, B \in \{skip, subgoals\}$
	$success$	if $ A = subgoals, B = success$

THE LANGUAGE

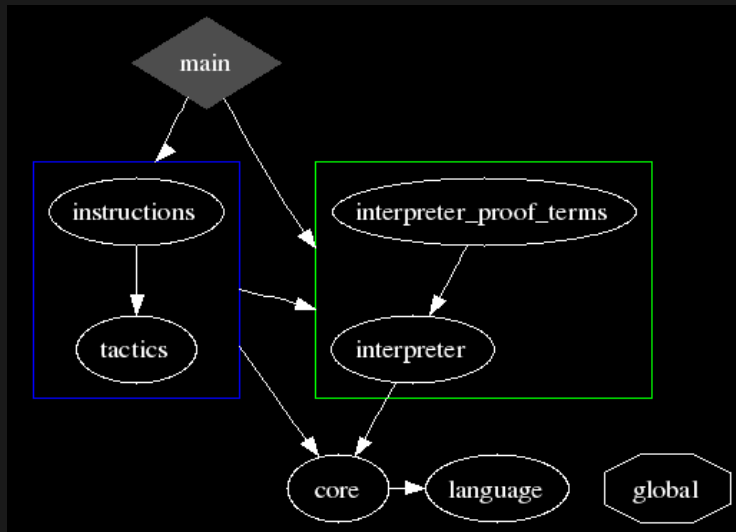
TACTICS *axiom, cut, elim, weaken*, contraction, focus, apply, ';' , ';[|]' , in.

INSTRUCTIONS declare, theorem, qed, discard, checkout, open, export natural language, min, full, lj, lk, next, prev, undo, quit.

ENTER FELLOWSHIP

- Objective Caml: leveraging objects and modules
- Small is beautiful: 4200+ code lines
- GPL-compatible CeCILL license
- Fantastic user base
- Ongoing work

ENTER FELLOWSHIP



TOPPING

- Compiler and toplevel options
- Toplevel help with terms, tactics and instructions
- Localized error printing
- UTF-8, formatted output
- Vi syntax file for `.fsp` extensions

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals
- Implement a toplevel-centric (\neq editor-centric) interface

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals
- Implement a toplevel-centric (\neq editor-centric) interface
- Implement a communication layer for graphical frontends (eg. PGIP)

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals
- Implement a toplevel-centric (\neq editor-centric) interface
- Implement a communication layer for graphical frontends (eg. PGIP)
- Generate proof scripts from $\bar{\lambda}\mu\tilde{\mu}$ -terms

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals
- Implement a toplevel-centric (\neq editor-centric) interface
- Implement a communication layer for graphical frontends (eg. PGIP)
- Generate proof scripts from $\bar{\lambda}\mu\tilde{\mu}$ -terms
- Extend to other provers (Isabelle, HOL, NuPrl, Mizar, etc.)

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals
- Implement a toplevel-centric (\neq editor-centric) interface
- Implement a communication layer for graphical frontends (eg. PGIP)
- Generate proof scripts from $\bar{\lambda}\mu\tilde{\mu}$ -terms
- Extend to other provers (Isabelle, HOL, NuPrl, Mizar, etc.)
- Integrate to provers

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals
- Implement a toplevel-centric (\neq editor-centric) interface
- Implement a communication layer for graphical frontends (eg. PGIP)
- Generate proof scripts from $\bar{\lambda}\mu\tilde{\mu}$ -terms
- Extend to other provers (Isabelle, HOL, NuPrl, Mizar, etc.)
- Integrate to provers
- Integrate concept to provers

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals
- Implement a toplevel-centric (\neq editor-centric) interface
- Implement a communication layer for graphical frontends (eg. PGIP)
- Generate proof scripts from $\bar{\lambda}\mu\tilde{\mu}$ -terms
- Extend to other provers (Isabelle, HOL, NuPrl, Mizar, etc.)
- Integrate to provers
- Integrate concept to provers
- Go modulo (P.B. and C.H.)

IN THE CROSSHAIRS

- Scale (\mathbb{R} is currently at 1000 lines)
- Develop the proof language: auto, tacticals
- Implement a toplevel-centric (\neq editor-centric) interface
- Implement a communication layer for graphical frontends (eg. PGIP)
- Generate proof scripts from $\bar{\lambda}\mu\tilde{\mu}$ -terms
- Extend to other provers (Isabelle, HOL, NuPrl, Mizar, etc.)
- Integrate to provers
- Integrate concept to provers
- Go modulo (P.B. and C.H.)
- Go higher-order (S.L. and A.M.)

THE NUTSHELL

- Fellowship is a Super Prover

THE NUTSHELL

- Fellowship is a Super Prover
- First order sequent calculus, correctness

THE NUTSHELL

- Fellowship is a **Super** Prover
- First order sequent calculus, correctness
- Generation of Coq and PVS proofs

THE NUTSHELL





- Fellowship is a Super Prover
- First order sequent calculus, correctness
- Generation of Coq and PVS proofs
- Play!

Google ‘Florent Kirchner’

I'm Feeling Lucky

- > Software
- > Fellowship

REFERENCES

-  B. Barras et al,
The Coq Proof Assistant Reference Manual – Version 8.0,
INRIA, 2005.
-  S. Owre et al,
PVS System Guide / Language Reference / Prover Guide,
SRI International, 2004.
-  G. Dowek et al,
HOL- $\lambda\sigma$: An intentional first-order expression of higher-order
logic
MSCS, 2001.
-  P.-L. Curien and H. Herbelin,
The Duality of Computation,
ICFP, 2000.