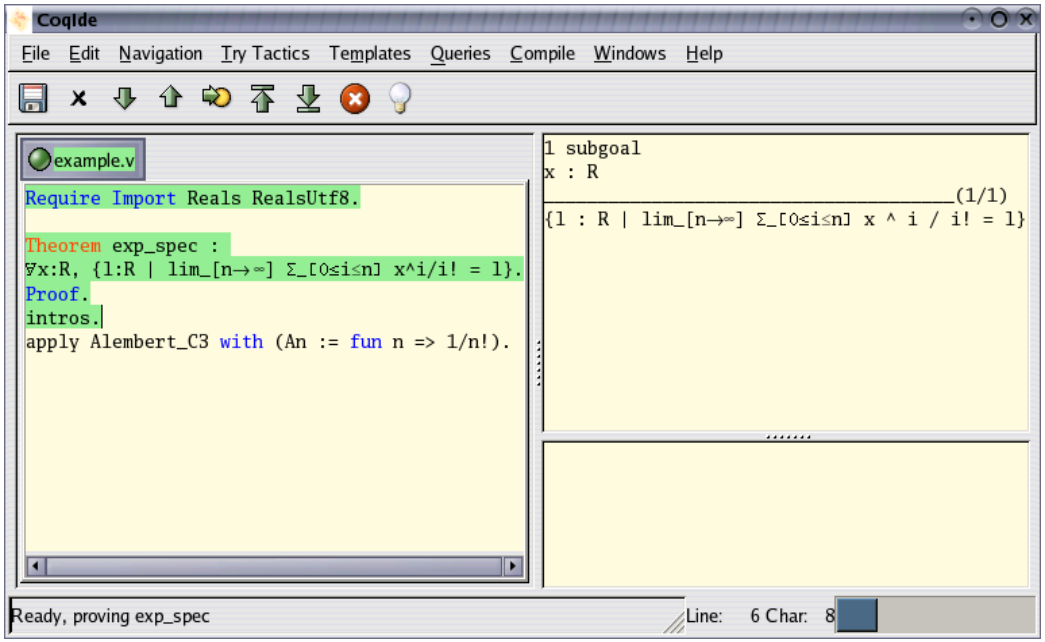
Validating Decision Procedures for Coq in Coq



Hugo Herbelin

Coq in a nutshell

- A logical formalism that is also a typed programming language: the *Calculus of Inductive Constructions* [Coquand-Paulin 1988]

- A verifier that checks that proofs are correct and programs are well-typed

On top of this:

- A concrete language of functions and specifications, e.g.:

```
Inductive formula :=
| Lit (b:bool) (s:string)
| And (f1 f2:formula)
| Or (f1 f2:formula).

Fixpoint negate f := match f with
  | Lit b s => Lit (negb b)
  | And f1 f2 => Or (negate f1) (negate f2)
  | Or f1 f2 => And (negate f1) (negate f2)
  end. e
```

- A language of semi-interactive tactics to solve specifications and theorems

```
Theorem negate_involutive : forall f, negate (negate f).
Proof. induction f as [ [] | | ]; simpl; f_equal; auto. Qed.
```

Top applications

- Certification of a C compiler (X. Leroy and the CompCert project)
- Full formalization of the 4-color theorem proof (G. Gonthier + B. Werner)
- Extensive constructive mathematics (Nijmegen's constructive repository of mathematics)
- Certification of a commercial JavaCard applet (GemAlto)
- Certification of large numbers primality (INRIA)
- see http://coq.inria.fr, link contributions ...

The Calculus of Inductive Constructions verifier

- Proofs have a concrete representation in Natural Deduction using the notations of λ-calculus:

Example:

```
fun (A B : Prop) (H : A /\ B) =>
       match H with
       | conj _ HB => or_introl HB
       end
```

is a proof of

```
forall A B : Prop, A /\ B -> B \/ A
```

The Calculus of Inductive Constructions verifier

- Proofs have a concrete representation (*proof-terms*):

Example:

```
fun (A B : Prop) (H : A /\ B) =>
       match H with
       | conj HA _ => or_intror HA
       end
```

is another proof of

```
forall A B : Prop, A /\ B -> B \/ A
```

The Calculus of Inductive Constructions verifier

- Proofs have a concrete representation (*proof-terms*):

Example:

```
fun (A B : Prop) (H : A /\ B) =>
       match H with
       | conj HA _ => or_intror HA
       end
```

is another proof of

```
forall A B : Prop, A /\ B -> B \/ A
```

- Coq provides a stand-alone verifier that can recheck, and re-certify proofs produced by another Coq session.

The "mathematical" proof language (C-zar)

```
Theorem negate_involutive : forall f, negate (negate f) = f.
proof.
  assume f:formula.
  per induction on f.
    suppose it is (Lit true s).
    thus thesis.
    suppose it is (Lit false s).
    thus thesis.
    suppose it is (And f1 f2) and IH1: thesis for f1 and IH2: thesis for f2.
    reconsider thesis as
      (And (negate (negate f1)) (negate (negate f2)) = And f1 f2).
    thus thesis by IH1, IH2.
    suppose it is (Or f1 f2) and IH1: thesis for f1 and IH2: thesis for f2.
    reconsider thesis as
      (Or (negate (negate f1)) (negate (negate f2)) = Or f1 f2).
    thus thesis by IH1, IH2.
  end induction.
end proof.
Qed.
```

Main approaches to automation

- full reflexion using Coq as a programming language

  $\hookrightarrow$ now using the built-in ocaml-style bytecode (strong) interpreter [B. Grégoire] and the mapping of integers to machine words [A. Spiwack]

- trace-based reflexion: the decision procedure returns a compact proof in its own language and a Coq function maps the proof language of the decision procedure to a Coq proof-term

- shallow embedding of the decision procedure proof steps as Coq proof steps

  – in ocaml, statically linked to Coq

  – in ocaml, dynamically linked to Coq (using ocaml 3.11 plugin mechanism)

  – in $\mathcal{L}_{tac}$, the user-level tactic language of Coq

  – external communication with Coq (XML syntax)

Main automation procedures in Coq

- Decision of intuitionistic first-order logic [C. Muñoz, D. Delahaye, P. Corbineau]
- Decision of closed equations by congruence closure [P. Corbineau]
- Decision of Presburger arithmetic (Pugh, 1992, quantifier-free, w/o dark shadow, "deep embedding" + experimental trace-based reflexion) [P. Crégut]
- Decision of equality of polynomial expressions (full reflexion) [S. Boutin, P. Loiseleur, A. Mahboubi, B. Grégoire, B. Barras]
- Decision of equality of polynomial fractions (full reflexion) [D. Delahaye, M. Mayero, L. Théry, B. Grégoire]
- Fourier-Motzin on $\mathbb{Q}$ and $\mathbb{R}$ (deep embedding [L. Pottier] + reflexion [F. Besson])
- Sums-of-squares based approximated decision of closed real fields (`csdp` try to find certificates for the Positivstellensatz Theorem, verification by polynomial simplification)
- Nullstellensatz Theorem using Gröbner bases [L. Pottier, independent release]
- Support for Maple simplification tools [D. Delahaye, M. Mayero, independent release]
- Resolution [C. Paulin-Mohring]

- Various dedicated procedures: theory of finite sets [P. Letouzey], inequalities between real numbers [R. O'Connor, G. Melquiond, R. Zumkeller] with application to the proof of Kepler's conjecture; preprocessing of advanced arithmetical operations (min, max, ...), ...
- SMT solving [S. Conchon, E. Contejean, S. Lécuyer, work in progress]
- + uncertified support for `CVC3`, `Z3`, `Simplify`, `Zenon`, `haRVey` for use in the program certification platforms `Why`, `Krakatoa` and `Caduceus` [J.-C. Filliâtre, C. Marché, C. Paulin-Mohring].
- ...

A certified SMT solver

S. Conchon, E. Contejean, 2006: `Alt Ergo`, a purely functional implemetation of a modular classical proposition solver modulo theories, including Fourier-Motzin and congruence closure (... lines of Objective Caml).

S. Lécuyer, in progress: implementation of the underlying SAT solver in Coq (reflexion) experimentation