Computer Algebra in Coq using reflection

Loïc Pottier

INRIA, team Marelle and Éducation Nationale

26-07-2012

Automatization in proof assistants

From Computer Algebra Systems to Proof Assistants : certified computations.

The case of polynomials.

Problem 1

$$x^2 - y^2 = (x + y)(x - y)$$

Computer Algebra System : compute canonical forms of both sides and test equality.

Proof Assistant : produces a formal proof of equality by rewriting or (better) using certified computations (reflection).
In Coq : tactic `ring` (Boutin, Grégoire, Mahboubi).

Proof by rewriting

Compute canonical form : $(x+y)(x-y) \to x^2 - y^2$

Rewriting from traces of computations :

$$
\begin{aligned}
&(x+y)(x-y) \to_{defsubtraction} \\
&(x+y)(x+(-y) \to_{rightdistributivity} \\
&x(x+(-y)) + y(x+(-y)) \to_{leftdistributivity} \\
&(x*x+x*(-y)) + y(x+(-y)) \to_{leftdistributivity} \\
&(x*x+x*(-y)) + (y*x+y*(-y)) \to_{associativity} \\
&x*x+((x*(-y)+y*x)+y*(-y)) \to_{multopp} \\
&x*x+((-x*y+y*x)+y*(-y)) \to_{multopp} \\
&x*x+((-x*y+y*x)+(-y*y)) \to_{commutativity} \\
&x*x+((-x*y+x*y)+(-y*y)) \to_{opposite} \\
&x*x+(0+(-y*y)) \to_{leftidentity} x*x+(-y*y)
\end{aligned}
$$

Non trivial expressions : huge proof ! Long time to verify it...

Reflection

Principle : replace space by time !

space = proof

time = computation

Space and time in the Coq system

Curry-Howard isomorphism :

- formula = type

$$A \to A$$

$$\forall A : Type, \forall x : A, x = x$$

- proof = term

$$x \mapsto x$$

$$A \mapsto x \mapsto refl\_equal(A, x)$$

Proof = (λ)terms :
(recursive) functions,
products ($\forall x : A, x = x$),
elements of inductive types (trees).

Verifying proofs = type checking

$$\lambda x.x : A \to A$$

$$A \to A \ : Prop$$

Computation = normal form computation

$$(\lambda xy.x)ab \to_\beta (\lambda y.a)b \to_\beta a$$

Kernel of Coq

Only two critical, small, efficient, and verified (Coquand, Paulin, Werner, Barras, Grégoire) algorithms :

Type checking : decide $t : T$

Conversion : decide $t \equiv_{\beta\iota\delta} t'$

Reflection in Coq

Let $P : A \to Prop$ a predicate on a type A. Let $c : A \to bool$ a semi-decision procedure on A :

$$\mathsf{c\_spec} : \forall a, c\ a = true \to P\ a$$

Now, to prove $P\ b$,

- apply c_spec, it remains $c\ b = true$ to prove
- if $c\ b$ reduces effectively to $true$ by conversion, then it suffices to prove $c\ b = true$ by reflexivity of equality.

Then c_spec b(refl_equal true) is a proof of $P\ b$.
Complexity ? Time to compute $c\ b$

Polynomials : tactic `ring`

Free algebra $\mathcal{A}$: polynomial expressions ($\simeq$ Maple)
= inductive type : variables V, sums, products, unit and zero, with equivalence $\sim$.

Polynomials in Horner normal form $\mathcal{P}$: efficient operations.

Normalisation $n : \mathcal{A} \to \mathcal{P}$: morphism defined by structural recursion, verifying
n_spec : $\forall a, b : \mathcal{A}, n(a) = n(b) \to a \sim b$

Evaluation : $\forall R$ a ring, $\rho : V \to R$, recursive functions $e_\rho : \mathcal{A} \to R$, such that
e_spec : $\forall a, b : A, a \sim b \to e_\rho(a) = e_\rho(b)$

$$\begin{tikzcd} \mathcal{A} \arrow[r, "n"] \arrow[d, "e_\rho"'] & \mathcal{P} \\ R & \end{tikzcd}$$

Let R a ring, $x, y \in R$.
To prove $x = y$,

- **Reification :** find $\rho : V \to R$, $a, b \in A$ such that $e_\rho(a)$ reduces to x and $e_\rho(b)$ reduces to y.
- then e_spec a b(n_spec a b(refl_equal $n(a)$)) is a proof of $x = y$.

$$\begin{tikzcd} n(a) = n(b) \arrow[r, "\text{n\_spec}"] & a \sim b \arrow[d, "\text{e\_spec}"'] \\ & x \equiv e_\rho(a) = e_\rho(b) \equiv y \end{tikzcd}$$

Conclusion : huge proof (with long type checking) replaced by efficient certified computation.

Problem 2

$$x^2 + 1 = 0 \wedge xy - 1 = 0 \Rightarrow x^3 - y = 0$$

Nullstellensatz : $x^3 - y$ belongs to the radical of the ideal generated by $x^2 + 1$ and $xy - 1$:

$$x^3 - y = (x - y)(x^2 + 1) + x(xy - 1)$$

Which can be proven by the tactic `ring`.

Problem : find this identity !

Solution : Buchberger algorithm.

General problem

Prove

$$\begin{array}{l}\forall X_1, \ldots, X_n \in A, \\ P_1(X_1, \ldots, X_n) = Q_1(X_1, \ldots, X_n), \\ \ldots, \\ P_s(X_1, \ldots, X_n) = Q_s(X_1, \ldots, X_n) \\ \rightarrow P(X_1, \ldots, X_n) = Q(X_1, \ldots, X_n)\end{array}$$

where P, Q, P_1, Q_1, $\ldots$, P_s, Q_s are polynomials and A is an integral domain, i.e. a commutative ring with no zero divisor. For example, A can be $\mathbb{R}$, $\mathbb{Z}$, of $\mathbb{Q}$. Note that the equality $=$ used in these goals can be any equivalence relation, not only Leibnitz equality.

Nullstellensatz

Hilbert's Nullstellensatz reduces proofs of equalities on polynomials expressions to algebraic computations :
if P in $A[X_1, \ldots, X_n]$ verifies

$$cP^r = \sum_{i=1}^{s} S_i P_i$$

with $c \in A$, $c \neq 0$, $r \in \mathbb{N}^*$, and $S_i \in A[X_1, \ldots, X_n]$,
then

$$P_1 = 0, ..., P_s = 0 \rightarrow P = 0$$

(the converse is also true when A is an algebraic closed field : the method is complete).

Gröbner bases and Buchberger algorithm

To find the S_i (suppose $r = 1$ and $c = 1$ for simplicity) :

Compute a partiel Gröbner basis $\mathcal{G}$ of $(P_1, ..., P_s)$ with Buchberger algorithm

Each time a new polynomial is added to $\mathcal{G}$, divide P with $\mathcal{G}$: if it reduces to 0, then, remembering all the steps of divisions gives

$$P = \sum_{i=1}^{s} S_i P_i$$

else, replace P with the remainder.

Problem : quickly, the S_i are too big !

Straightline programs as certificates

Principle : during Gröbner bases computation and division, express new polynomials with olders, and don't expand them.

Complexity can be reduced by an exponential factor.

Example

Prove

$x^2 + 1 = 0 \wedge xy - 1 = 0 \Rightarrow x^3 - y = 0$

Let $P_1 = x^2 + 1$, $P_2 = xy - 1$ and $P = x^3 - y$
Gröbner basis computation : initial family $\{P_1, P_2\}$

- division of P by $\{P_1, P_2\}$ gives $R_1 = -x - y = P - xP_1$
- S-polynomial of P_1 and P_2 gives $P_3 = x + y = yP_1 - xP_2$, irreducible. Then the family becomes $\{P_1, P_2, P_3\}$.
- division of R_1 gives $0 = R_1 + P_3$ so we stop the completion

Then we obtain a straigthline program :

$$
\begin{aligned}
&P_1 := x^2 + 1;\\
&P_2 := xy - 1;\\
&P_3 := yP_1 + (-x)P_2;\\
&P := -((-x)P_1 + 0P_2 + 1P_3);
\end{aligned}
$$

Proving $P_1 = 0 \wedge P_2 = 0 \Rightarrow P = 0$ reduces to prove each line of the certificate with the ring tactic, and compose proofs with rewriting :)

Certificate

In general, a certificate CR for $P_1 = 0, ..., P_s = 0 \rightarrow P = 0$ is

$$\begin{aligned} CR = & \quad [c_1, \ldots, c_{s+p}] \\ C = & \quad [[a_{1\ s+1}, \ldots, a_{s\ s+1}], \\ & \quad \ldots \\ & \quad [a_{1\ s+p}, \ldots, a_{s\ s+p}, \ldots, a_{s+p-1\ s+p}]] \end{aligned}$$

where

$$\forall i \in [1; p], P_{s+i} = a_{1\ s+i} P_1 + \ldots + a_{s+i-1\ s+i} P_{s+i-1}$$

and

$$P = -(c_1 P_1 + \ldots + c_{s+p} P_{s+p})$$

Reification

Remember :

$$\begin{tikzcd} \mathcal{A} \arrow[r, "n"] \arrow[d, "e_\rho"'] & \mathcal{P} \\ R & \end{tikzcd}$$

Prove

$P_1 = 0, ..., P_s = 0 \rightarrow P = 0$ where $P_1, ..., P_s, P \in R$.

Reification = find $\rho : V \rightarrow R$, and $E_1, ..., E_s, E$ such that

$$e_\rho(E_1) \equiv P_1, ..., e_\rho(E_s) \equiv P_s, e_\rho(E) \equiv P$$

Find variables in $P_1, ..., P_s, P$, and build $E_1, ..., E_s, E$ by translating variables inyo elements of V (variables in $\mathcal{A}$), additions of R into additions in $\mathcal{A}$, etc

Several solutions in Coq system :

- program in ocaml (vintage).
- program in LTac, the tactic meta-langage of Coq (classic).
- use inference of Type Classes (fashion victim).

Type Classes in Coq

A dependent Prolog engine built in type inference, where

Predicates are types of records

Horn clauses are functions that build records

Unification as method to fill holes in terms.

```
...
Class reify (R:Type)`{Rr:Ring (T:=R)} (e:A)
            (lvar:list R) (t:R).

Instance  reify_zero (R:Type)  lvar op
 `{Ring (T:=R)(ring0:=op)}
 : reify (ring0:=op)(A0 0%Z) lvar op.
...
Instance  reify_add (R:Type)
  e1 lvar t1 e2 t2 op
 `{Ring (T:=R)(add:=op)}
 {_:reify (add:=op) e1 lvar t1}
 {_:reify (add:=op) e2 lvar t2}
 : reify (add:=op) (Aadd e1 e2) lvar (op t1 t2).
```

```
Instance  reify_var (R:Type) t lvar i
 `{nth R t lvar i}
 `{Rr: Ring (T:=R)}
 : reify (Rr:= Rr) (Avar i) lvar t
 | 100.
```

To reify a list of terms `lt` of a ring R :

```
Definition list_reifyl (R:Type) lexpr lvar lterm
 `{Rr: Ring (T:=R)}
 {_:reifylist (Rr:= Rr) lexpr lvar lterm}
 `{closed (T:=R) lvar}  := (lvar,lexpr).
```

gives the set of variables and the reified expressions in $\mathcal{A}$, by inference of implicit arguments in `(@list_reifyl _ _ _ (lterm:=lt))`

Tactic nsatz

Let R an integral domain, $P_1, ..., P_s, P \in R$,

to prove

$$P_1 = 0, ..., P_s = 0 \rightarrow P = 0$$

Reify $P_1, ..., P_s, P$ in $E_1, ..., E_s, E$.
Let $p_1 = n(E_1), ..., p = n(E)$ be the normal forms of $E_1, ..., E_s, E$.

Compute a certificate of $c\ p^r = \sum_i S_i p_i$ with adapted Buchberger algorithm (in ocaml)

Prove lines $p_{s+k} = \sum_i a_{ik} p_{s+i}$ of the certificate with `ring`

Combine these proofs by rewriting to obtain the final proof of $P = 0$.

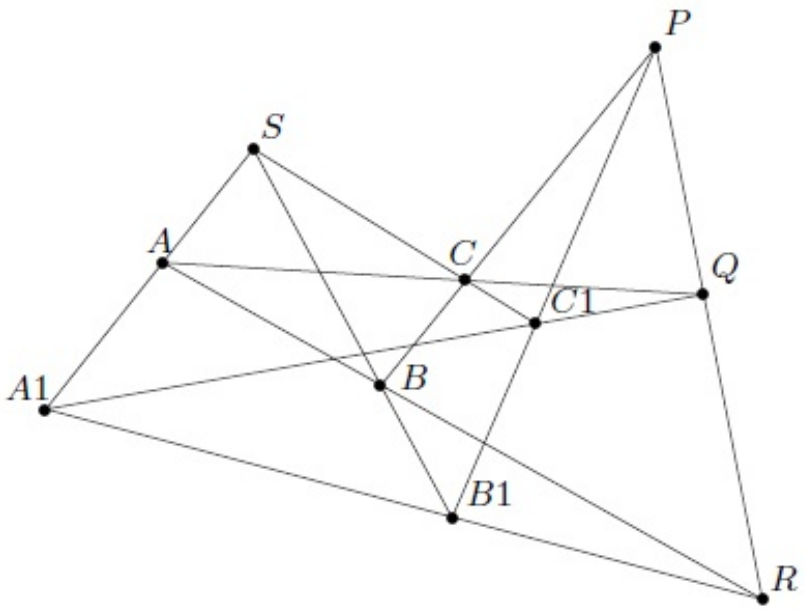
Examples : geometry

Points = coordinates

Geometric predicates = polynomials

```
Definition collinear (A B C:point):=
(X A - X B) * (Y C - Y B)
- (Y A - Y B) * (X C - X B) = 0.
```

Desargues' theorem




```
Lemma Desargues: forall A B C A1 B1 C1 P Q R S:point,
collinear A S A1 -> collinear B S B1 -> collinear C S C1 -
collinear B1 C1 P -> collinear B C P -> collinear A1 C1 Q
collinear A C Q -> collinear A1 B1 R -> collinear A B R
-> collinear P Q R.
```

False...

Don't forget degenerated cases !
A correct statement :

```
Lemma Desargues: forall A B C A1 B1 C1 P Q R S:point,
X S = 0 -> Y S = 0 -> Y A = 0 -> (* to speed up *)
collinear A S A1 -> collinear B S B1 -> collinear C S C1 -
collinear B1 C1 P -> collinear B C P -> collinear A1 C1 Q
collinear A C Q -> collinear A1 B1 R -> collinear A B R
-> collinear P Q R
\/ X A = X B \/ X A = X C \/ X B = X C  \/ X A = 0
\/ Y B = 0 \/ Y C = 0
\/ collinear S B C \/ parallel A C A1 C1
\/ parallel A B A1 B1.
```

Proof


```
Proof.
geo_begin.
let lv := rev (X A :: X B :: Y B :: X C :: Y C :: Y A1
 :: X A1 :: Y B1 :: Y C1 :: X R :: Y R :: X Q :: Y Q
 :: X P :: Y P :: X C1 :: X B1 :: nil) in
nsatz with radicalmax :=1%N strategy:=0%Z
  parameters:=(X A::X B::Y B::X C::Y C::X A1::Y B1::Y C1
                ::nil)
  variables:= lv.
Qed. (* 8s *)
```

Certificate

50K :

$$CR := [u_1^4 - u_1^3 u_2 u_3 + u_1^2 - u_1 u_2 u_3 u_4^2 - 2u_1^3 + 2u_1^2 u_2 u_3 u_4 - u_1^4 - u_1^2 u_2^2 + 2u_1^3 u_2 + u_1^3 + u_1 u_2^2 - 2u_1^2 u_2 u_4 u_5 u_6^2 - u_1^5 + u_1^4 u_2 u_3 - u_1^3 + u_1^2 u_2 u_3 u_4^2 + 2u_1^4 - 2u_1^3 u_2 u_3 u_4 + u_1^5 + u_1^3 u_2^2 - 2u_1^4 u_2 - u_1^4 - u_1^2 u_2^2 + 2u_1^3 u_2 u_4 u_5 u_6; -u_1 + u_2 u_3^3 u_4^2 + u_1^2 - u_1 u_2 u_3^3 u_4 + u_2^3 - 2u_1 u_2^2 + u_1^2 u_2 u_3 u_5^2 + u_1 u_2^2 - u_1^2 u_2 u_3^2 - u_1^2 - 2u_2^2 + 3u_1 u_2 u_3^2 u_4 u_5 u_6 + u_1^2 - u_1 u_2 u_3^2 u_4^2 - u_1^3 + u_1^2 u_2 u_3^2 u_4 - u_1 u_2^3 + 2u_1^2 u_2^2 - u_1^3 u_2 u_5^2 - u_1^2 u_2^2 + u_1^3 u_2 u_3 + u_1^3 + 2u_1 u_2^2 - 3u_1^2 u_2 u_3 u_4 u_5 u_7 l + u_1 - u_2 u_3^3 u_4^3 - u_1^2 + u_1 u_2 u_3^3 u_4^2 - u_2^3 + 2u_1 u_2^2 - u_1^2 u_2 u_3 u_4 u_5^2 + u_1^2 + 2u_2^2 - 3u_1 u_2 u_3^2 u_4^2 - u_1 u_2^2 + u_1^2 u_2 u_3^2 u_4 u_5 u_6 + u_1 u_2^2 - u_1^2 u_2 u_3^2 u_4^2 - u_1^2 u_2^2 + u_1^3 u_2 u_3^2 u_4 + u_1 u_2^4 - 2u_1^2 u_2^3 + u_1^3 u_2^2 u_5^2 + u_1^2 u_2^3 - u_1^3 u_2^2 u_3 - 2u_1 u_2^3 + 3u_1^2 u_2^2 - u_1^3 u_2 u_3 u_4 u_5 - u_1 + u_2 u_3^2 u_4^3 + u_1^2 - u_2^2 u_3^2 u_4^2 + u_1 u_2^2 - u_1^2 u_2 u_3^2 u_4 - u_2^4 + 2u_1 u_2^3 - u_1^2 u_2^2 + u_2^3 - 2u_1 u_2^2 + u_1^2 u_2 u_4 u_5^2 - u_1 u_2^3 + u_1^2 u_2^2 u_3 - u_1^2 - 2u_2^2 + 3u_1 u_2 u_3 u_4^2 + 2u_2^3 - 2u_1 u_2^2 u_3 u_4 u_5 u_6 u_7; \ldots$$

Improvements

Fractions : to deal with degenerate cases, work in

$$A(X_1, \ldots X_n)[Y_1, \ldots, Y_m]$$

Ordering of variables : complexity of Buchberger algorithm depends heavily on ordering fo variables. Need to parametrize it.

Strategy : in Buchberger algorithm. "Sugar" by default.

Radical : bound r in $P^r = \sum_i S_i P_i$, trying $r = 1$, then $r = 2$, etc. Using extra variables to find r is expensive...

Examples

Many examples here (thanks to Laurent Théry) :
`http://www-sop.inria.fr/marelle/CertiGeo`

Conclusion

Type theory to formalize mathematics

Reflection as method to integrate decision prodedures in Proof Assistants (to integrate Maple inside Coq ?)

Unification / Type Classes to perform reification in reflection

Certificates to remember proofs

Certified reduction to perform computation