Constructive quantifier elimination for real numbers and complex numbers, in a proof assistant

Joint Work with Cyril Cohen

Assia Mahboubi

INRIA Microsoft Research Joint Centre (France)
INRIA Saclay – Île-de-France
École Polytechnique, Palaiseau

July 7th 2011

This work has been partially funded by the FORMATH project, nr. 243847, of the FET program within the 7th Framework program of the European Commission.

Motivations

- Formalization in a type-theory based proof assistant (Coq)
- Of quantifier elimination procedures
- Motivated by the application to the theory of real closed and algebraically closed fields

The language of rings and fields

Terms are:

- Variables : $x, y, \ldots$
- Constants 0 and 1
- Opposites: $-t$
- Sums: $t_1 + t_2$
- Differences: $t_1 - t_2$
- Products: $t_1 * t_2$
- Divisions: $t_1 \ t_2$

Terms are polynomial expressions in the variables.

Terms are rational fractions in the variables.

First order formulas in the language of ordered rings

Atoms are:

- Equalities: $t_1 = t_2$
- Inequalities: $t_1 \geq t_2$, $t_1 > t_2$, $t_1 \leq t_2$, $t_1 < t_2$

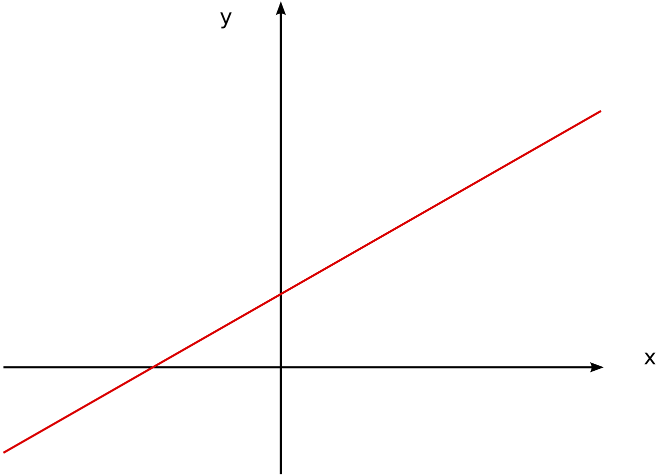
Formulas are:

- Atoms
- Conjunctions: $F_1 \wedge F_2$
- Disjunctions: $F_1 \vee F_2$
- Negations: $\neg F$
- Implications: $F_1 \Rightarrow F_2$
- Quantifications: $\exists x, F$, $\forall x, F$

Formulas are quantified systems of polynomial constraints.

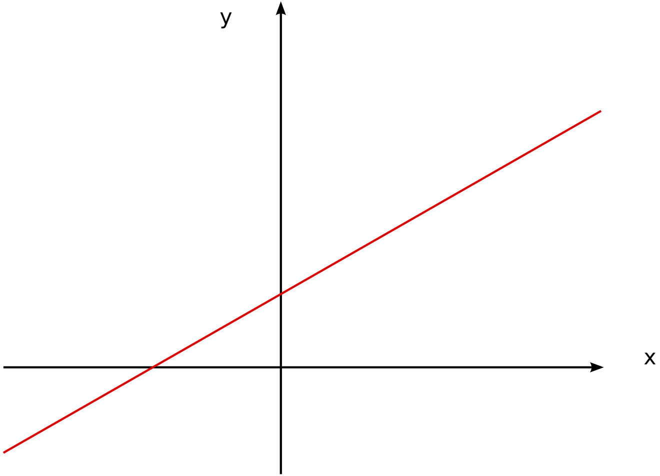
A taste of the first order language of ordered rings

"Any polynomial of degree one has a real root."



A taste of the first order language of ordered rings

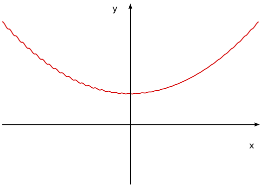
"Any polynomial of degree one has a real root."

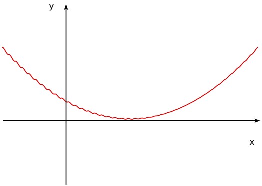


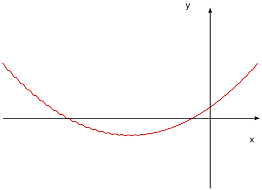
$$\forall a \forall b, \exists x, a * x + b = 0$$

A taste of the first order language of ordered rings

"Any polynomial of degree two has at most two real roots."

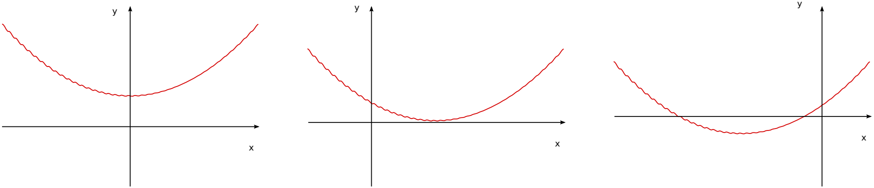






A taste of the first order language of ordered rings

"Any polynomial of degree two has at most two real roots."



$$\forall a \forall b, \forall c \forall x \forall y \forall z,$$
$$(ax^2 + bx + c = 0 \wedge ay^2 + by + c = 0 \wedge az^2 + bz + c = 0)$$
$$\Rightarrow (x = y \vee x = z \vee y = z)$$

A taste of the first order language of ordered rings

- "Any polynomial has less roots than its degree."

A taste of the first order language of ordered rings

- "Any polynomial has less roots than its degree."

  This demands higher order.

A taste of the first order language of ordered rings

- "Any polynomial has less roots than its degree."

  This demands higher order.

- "Any number is either rational or non rational."

A taste of the first order language of ordered rings

- "Any polynomial has less roots than its degree."

  This demands higher order.
- "Any number is either rational or non rational."

  The language is not precise enough.

First order theory of discrete ordered rings

- The theory of ordered rings (resp. fields) is:
  - The theory of rings (resp. fields)
  - A total order $\leq$
  - Compatibility of the order with ring (resp. field) operations

First order theory of discrete ordered rings

- The theory of ordered rings (resp. fields) is:
  - The theory of rings (resp. fields)
  - A total order $\leq$
  - Compatibility of the order with ring (resp. field) operations
- The theory of discrete real closed field is the theory of real closed field plus decidability of the order relation.

Real closed fields, Algebraically closed fields

- The theory of real closed fields is:
  - The theory of ordered fields
  - The axiom scheme: for all $n \in \mathbb{N}$, any polynomial of degree $2n + 1$ has a root.

Real closed fields, Algebraically closed fields

- The theory of real closed fields is:
  - The theory of ordered fields
  - The axiom scheme: for all $n \in \mathbb{N}$, any polynomial of degree $2n + 1$ has a root.
- The theory of algebraically closed fields is:
  - The theory of ordered fields
  - The axiom scheme: for all $n \in \mathbb{N}$, any polynomial of degree $n + 1$ has a root.

Examples of real closed fields

- (Classical) real numbers
- Real algebraic numbers
- The field of Puiseux series on a RCF R

  (generalizing formal power series)

Examples of algebraically closed fields

- (Classical) complex numbers
- Algebraic numbers
- Algebraic closure of finite fields

First order theory of real closed fields

Theorem (Tarski (1948))

The classical theory of real closed fields admits quantifier elimination and is hence decidable.

Same result holds for algebraically closed fields.

There exists an algorithm which proves or disproves any theorem of real algebraic geometry (which can be expressed in this first order language).

Remarks

- We can decide whether an arbitrary given polynomial with rational coefficients has a real root.

Remarks

- We can decide whether an arbitrary given polynomial with rational coefficients has a real root.
- But we do not know whether this root is an integer or a rational.

Remarks

- We can decide whether an arbitrary given polynomial with rational coefficients has a real root.
- But we do not know whether this root is an integer or a rational.
- There is indeed no algorithm to decide the solvability of Diophantine equations (Matiyasevitch, 1970).

Remarks

There is an algorithm which determines:

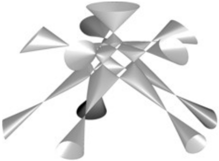
- If your piano can be moved through the stairs and then to your dinning room;
- If a (specified) robot can reach a desired position from an initial state;
- The solution to Birkhoff interpolation problem;
- ...

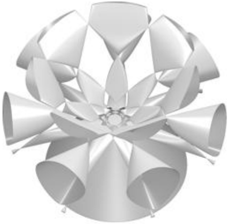
Remarks

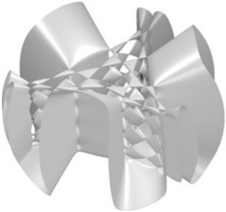
This algorithm gives the complete topological description of semi-algebraic varieties.

Remarks

This algorithm gives the complete topological description of semi-algebraic varieties.

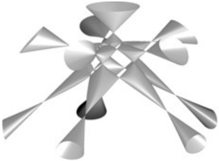
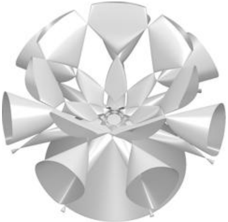
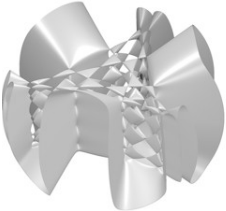






Remarks

This algorithm gives the complete topological description of semi-algebraic varieties.

Which seems a rather intricate problem...

Thanks to Oliver Labs for the pictures.

Formalization in the Coq system

The Coq system: a type theory based proof assistant.

- Coq is a (functional) programming language.
- Coq has such a rich type system that the types of objects can be theorem statements.
- In the absence of axiom, proofs should be intuitionistic.

Why formalizing these proofs in a proof assistant?

The idea is to adopt a reflexive approach:

- Implement the quantifier elimination procedure inside the system;
- Prove formally its correctness.

Hence:

- We obtain a certified decision procedure.

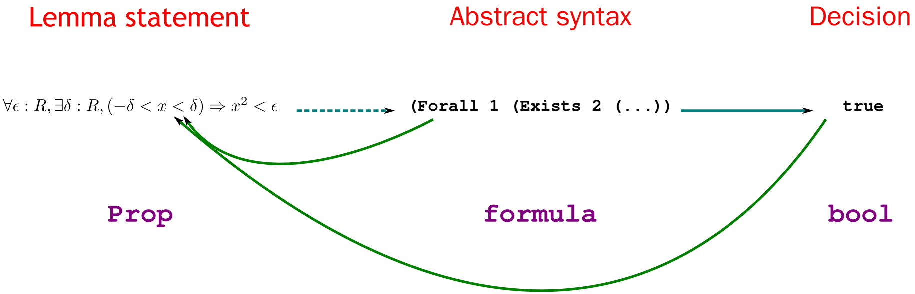
  (here for non linear arithmetics)

- We legitimate axiom-free classical reasoning in these logical fragments.

  (here for the first order theory of reals, complex...)

From proofs to proof-producing procedure

The (so-called) reflexion scheme:



Quantifier Elimination

A theory T on a language Σ with a set of variables $\mathcal{V}$ admits quantifier elimination if

- for every formula $\phi(\vec{x}) \in \mathcal{F}(\Sigma, \mathcal{V})$,
- there exists a quantifier free formula $\psi(\vec{x}) \in \mathcal{F}(\Sigma, \mathcal{V})$
- such that:

$$T \vdash \forall \vec{x}, ((\phi(\vec{x}) \Rightarrow \psi(\vec{x})) \wedge (\psi(\vec{x}) \Rightarrow \phi(\vec{x})))$$

Formal definition of a first order theory

For an arbitrary type term of terms, formulas are:

```
Inductive formula (term : Type) : Type :=
| Equal of term & term
| Leq of term & term
| Unit of term
| Not of formula
| And of formula & formula
| Or of formula & formula
| Implies of formula & formula
| Exists of nat & formula
| Forall of nat & formula.
```

Formal definition of the ring signature

Terms on the language of fields.

```
Inductive term : Type :=
| Var of nat
| Const0 : term
| Const1 : term
| Add of term & term
| Opp of term
| Mul of term & term
| Inv of term
```

Proving quantifier elimination on real closed fields

To state the theorem of quantifier elimination, we could:

- Build the list `T` of `formulas` describing the axioms of a real closed field structure.
- Formalize first order provability, $T \vdash \phi$, a predicate of type:

```
Definition entails
   (T : seq (formula R))(phi : formula R) : bool :=
      ...
```

But given our motivations, this not the most relevant approach.

Semantic quantifier elimination

A theory T on a language Σ with a set of variables $\mathcal{V}$ admits semantic quantifier elimination if

- for every $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$,
- there exists a quantifier free formula $\psi \in \mathcal{F}(\Sigma, \mathcal{V})$
- such that for any model M of T, and for any list e of values,

$$M, e \models \phi \text{ iff } M, e \models \psi$$

This is the (a priori weaker) quantifier elimination result we formalize.

Theory of real closed fields

We use a record type to define a type which is simultaneously equipped with a field signature and a theory of real closed fields.

```
Record rcf := RealClosedField{
  carrier : Type;
  Req : carrier -> carrier -> bool;
  zero : carrier;
  one : carrier
  opp : carrier -> carrier;
  add : carrier -> carrier -> carrier;
  mul : carrier -> carrier -> carrier;
  inv : carrier -> carrier;
  _ : associative add;
  _ : commutative add;
  _ : left_id zero add;
  _ : left_inverse zero opp add;
  ...}.
```

Instances of the theory of real closed fields

An instance of this theory is constructed when:

- We have formed a concrete type
  for instance the type `Ralg` of real algebraic numbers
- We have defined field constants and implemented field operations
  Zero, one, addition, ...
- We have proved the theorems specifying these operations
  Addition is commutative, ...
- We have gathered all this in an element of the record type

```
Definition Ralg_rcf :=
    RealClosedField Ralg Ralg0 Ralg1 Ralg_opp ...
```

What do we formalize?

- A signature Σ (of rings)

  The type `term`

- The terms on Σ

  The elements `t : term`

- The first order statements $\mathcal{F}(\Sigma, \mathbb{N})$

  The elements `f : formula`

- The definition of Σ-structure

  The type `rcf` (which contains specifications).

- The Σ-structures themselves

  The elements `MyRcf : rcf`

What do we formalize?

- An interpretation function $[t(\mathbf{x})]_{R,e}$ of terms in $\mathcal{L}(\Sigma, \mathbb{N}))$ in a Σ-structure

  `eval: (seq (carrier R))-> term -> (carrier R)`

- An interpretation function $[\phi(\mathbf{x})]_{R,e}$ of formulas in $\mathcal{F}(\Sigma, \mathbb{N}))$ in Coq statements

  `holds: (seq (carrier R))-> formula -> Prop`

- R is a model of the theory of (discrete) real closed fields

  $R : rcf$

- $R, e \models f$

  A proof of the Coq statement `(holds e f)`

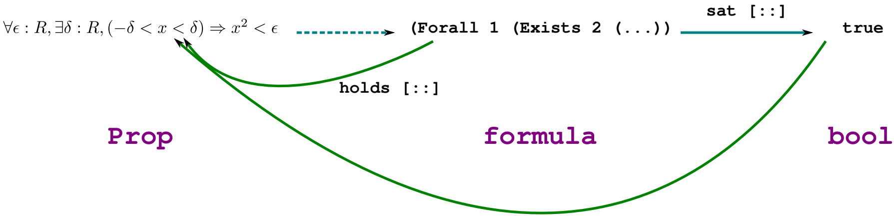
- Quantifier elimination is valid

```
forall (f : formaula R)(e : seq (carrier R)),
                        (holds e f)<-> (holds e (qe f))
```

From proofs to proof-producing procedure

Decidability comes from the implementation of a correct sat operator:

Lemma statement — Abstract syntax — Decision



apply sat_correct.

Summary

- A formal definition of terms and first order formulas in Coq
- A definition of structures and theories

  (example of real closed fields)
- An interpretation of abstract formulas as Coq formulas
- A reflexion scheme to prove a Coq formula by computation

  (computations on abstract formulas)
- We study the theories for which the `sat` operator is implemented by quantifier elimination.

Can this be more modular?

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.
    - Consider $\exists x, F$ where F is quantifier free

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.
  - Consider $\exists x, F$ where F is quantifier free
  - F is equivalent to its disjunctive normal form: $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} L_{i,j}$

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.
  - Consider $\exists x, F$ where F is quantifier free
  - F is equivalent to its disjunctive normal form: $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} L_{i,j}$
  - The existential quantifier distributes over the disjunctions.

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.
  - Consider $\exists x, F$ where F is quantifier free
  - F is equivalent to its disjunctive normal form: $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} L_{i,j}$
  - The existential quantifier distributes over the disjunctions.
  - The hypothesis is applied to every conjunction $\bigwedge_{j=1}^{m} L_{i,j}$

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.

- Then elimination holds for any formula, by induction on its structure:

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.

- Then elimination holds for any formula, by induction on its structure:
  - All cases are trivial except for quantified formulas.

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.

- Then elimination holds for any formula, by induction on its structure:
  - All cases are trivial except for quantified formulas.
  - Existential case:

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.

- Then elimination holds for any formula, by induction on its structure:
  - All cases are trivial except for quantified formulas.
  - Existential case:
    - $\exists x, F$ where F can be considered qf (by induction).

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.

- Then elimination holds for any formula, by induction on its structure:
  - All cases are trivial except for quantified formulas.
  - Existential case:
    - ⋆ $\exists x, F$ where F can be considered qf (by induction).
    - ⋆ The first lemma applies.

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.

- Then elimination holds for any formula, by induction on its structure:
  - All cases are trivial except for quantified formulas.
  - Existential case: Ok

A standard trick when atoms are decidable

Suppose it is possible to eliminate the $\exists$ in $\exists x, \bigwedge_{i=1}^{n} L_i$.

- Then it is possible to eliminate a single prenex $\exists$.

- Then elimination holds for any formula, by induction on its structure:
  - All cases are trivial except for quantified formulas.
  - Existential case: Ok
  - Universal case: $\forall x, F$ where F can be considered qf (by induction).
    - Since $(F \vee \neg F)$ holds, $\forall x, F$ is equivalent to $\neg \exists x, \neg F$.
    - $\neg F$ is quantifier free: the lemma applies to $\exists \neg F$.
    - The outermost negation does not introduce quantifiers.

A modular formalization for quantifier elimination

- We have formalized the previous remark using abstract formulas.
- The proof is parameterized by a single existential operator:

```
Parameter proj : nat -> formula -> formula.
```

- And its correctness hypotheses:
  - Its output should be quantifier-free.
  - Its output should be equivalent to its input.

Projection theorems

- In a candidate theory, we need to show that we can eliminate a single existential quantifier on conjunctions of literals.
- With geometer eyes, this is the typical shape of a projection operator.

Emptiness of a one dimensional basic semi-algebraic set

If there is no free variable (no parameter), we want to decide whether

$$\{x \in R \mid P(x) = 0 \wedge \bigwedge_{Q \in \mathcal{Q}} Q(x) > 0\}$$

is empty or not, for $P \in R[X]$ and $\mathcal{Q} \subset R[X]$ (finite).

Emptiness of a one dimensional basic algebraic set

If there is no free variable (no parameter), we want to decide whether

$$\{x \in R \mid P(x) = 0 \wedge \bigwedge_{Q \in \mathcal{Q}} Q(x) \neq 0\}$$

is empty or not, for $P \in C[X]$ and $\mathcal{Q} \subset C[X]$ (finite).

Algebraic characterization

The typical proof of such an emptiness test gives an algebraic characterization of non-emptiness:

$$\{x \in R \mid P(x) = 0 \wedge \bigwedge_{Q \in \mathcal{Q}} Q(x) > 0\}$$

$$\Leftrightarrow \mathrm{degree}\left(\mathrm{gdco}_{(\prod_{i=1}^{m} Q_i)}(P)\right) \geq 1 \quad (2)$$

where $\mathrm{gdco}_Q(P)$ is the greatest divisor of P coprime to Q.

Algebraic characterization is not enough

How does this scale to the parametric case? Example.

Back to quantifier elimination

- We need a model-independent description of a finite partition of the space of parameters C^k into cells described by a quantifier-free formula
- Each cell corresponds to a possible value for the quantifier free equivalent formula.
- This description is obtained by analyzing the tree of successive zero (or sign) tests performed when computing the algebraic characterization
- How to implement the construction of this formula?

Back to quantifier elimination

- We have designed a reflexion scheme in Coq for the decidability of first order theories.
- We want to implement decision procedures based on quantifier elimination.
- We have reduced full quantifier elimination to single existential elimination, to be provided by the theory of interest:

```
Parameter proj : nat -> formula -> formula.
```

- Single existential elimination is a projection theorem for the theory.
- Projection typically comes from the existence of an algebraic emptiness test.
- How does this helps for the implementation of proj?

Abstract polynomials

Consider the formula with a single existential quantifier:

$$\exists x, \alpha x^2 + (\beta x + 1) + \alpha\gamma = 0$$

- The atom is a sign condition on the term $\alpha x^2 + \beta x + \gamma$;
- The single quantifier binds the variable x;
- The term in the atom should be understood as a polynomial, element of $R[\alpha, \beta, \gamma][x]$

Abstract polynomials

Consider the formula with a single existential quantifier:

$$\exists x, \alpha x^2 + (\beta + 1)x + \alpha\gamma = 0$$

- The term embedded in such an atom can be seen as an abstract univariate polynomial, with abstract polynomial coefficients.
- An abstract univariate polynomial is represented by lists of terms.
  $[\alpha, \beta + 1, \alpha\gamma]$: `(seq (term R))`
- An abstract coefficient is only a term.

Abstract polynomials

- From a `(t : term)` in an atom, and the name *i* of the variable bound by the existential, we can extract the abstract univariate polynomial in the variable x_i thanks to the function:

```
Fixpoint abstrX (i : nat) (t : term R) : (seq term) :=
    ...
```

- In a given context, an abstract univariate polynomial can be interpreted by a usual univariate polynomial:

```
Fixpoint eval_polyF (e : seq R) (ap : (seq term)) :=
  match ap with
  |c :: qf => (eval_polyF e qf)*'X + (eval e c)
  |[::] => 0
end.
```

- We want the diagram to commute.

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
Definition test (p : {poly R}) : bool := lcoef p > 0
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
Definition test (p : {poly R}) : bool := lcoef p > 0

Fixpoint cps_lcoef                    (p : {poly R}) :       :=




Definition cps_test (p : {poly R}) : bool :=
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
Definition test (p : {poly R}) : bool := lcoef p > 0

Fixpoint cps_lcoef                    (p : {poly R}) :       :=
  match p with
  | [::] →
  | c :: q →
  end.
Definition cps_test (p : {poly R}) : bool :=
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
Definition test (p : {poly R}) : bool := lcoef p > 0

Fixpoint cps_lcoef (k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] →
  | c :: q →
  end.
Definition cps_test (p : {poly R}) : bool :=
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
Definition test (p : {poly R}) : bool := lcoef p > 0

Fixpoint cps_lcoef (k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q →
  end.
Definition cps_test (p : {poly R}) : bool :=
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
Definition test (p : {poly R}) : bool := lcoef p > 0

Fixpoint cps_lcoef (k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q →  cps_lcoef                                  q
  end.
Definition cps_test (p : {poly R}) : bool :=
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
Definition test (p : {poly R}) : bool := lcoef p > 0

Fixpoint cps_lcoef (k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q →  cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.
Definition cps_test (p : {poly R}) : bool :=
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
   match p with
   | [::] → 0
   | c :: q → if (q == 0) then c else (lcoef q)
   end.
Definition test (p : {poly R}) : bool := lcoef p > 0

Fixpoint cps_lcoef (k : R → bool) (p : {poly R}) : bool :=
   match p with
   | [::] → (k 0)
   | c :: q →  cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
   end.
Definition cps_test (p : {poly R}) : bool :=
cps_lcoef                                     p
```

Inside out

```
Fixpoint lcoef (p : {poly R}) : R :=
  match p with
  | [::] → 0
  | c :: q → if (q == 0) then c else (lcoef q)
  end.
Definition test (p : {poly R}) : bool := lcoef p > 0

Fixpoint cps_lcoef (k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q →  cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.
Definition cps_test (p : {poly R}) : bool :=
cps_lcoef (fun r ⇒ if r > 0 then true else false) p
```

Continuation passing style

- This is not (meant to be) code obfuscation.
- We have exposed the control operations by the mean of a continuation.
- This version of the code is ready to be translated at the formula level:
  - By turning boolean outputs into formulas outputs
  - By turning polynomials and coefficients into terms
- Remark : we can define a branching formula:
  Definition ifF (condF thenF elseF : formula R) : formula R :=
  ((condF $\wedge$ thenF) $\vee$ ((condF) $\wedge$ elseF)).

Formula level programs

```
Fixpoint cps_lcoef
(k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q → cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.
```

Formula level programs

```
Fixpoint cps_lcoef
(k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q → cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.

Fixpoint cps_lcoefF
(k :          →              ) (p   :              )) :             :=
  match p   with
  | [::] →
  | c :: q → cps_lcoefF                                              q
  end.
```

Formula level programs

```
Fixpoint cps_lcoef
(k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q → cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.

Fixpoint cps_lcoefF
(k :          →            ) (pF :              )) :            :=
  match pF with
  | [::] →
  | c :: q → cps_lcoefF                                             q
  end.
```

Formula level programs

```
Fixpoint cps_lcoef
(k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q → cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.

Fixpoint cps_lcoefF
(k :          →                ) (pF : (seq (term R))) :              :=
  match pF with
  | [::] →
  | c :: q → cps_lcoefF                                              q
  end.
```

Formula level programs

```
Fixpoint cps_lcoef
(k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q → cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.

Fixpoint cps_lcoefF
(k : term R →                    ) (pF : (seq (term R))) :              :=
  match pF with
  | [::] →
  | c :: q → cps_lcoefF                                                  q
  end.
```

Formula level programs

```
Fixpoint cps_lcoef
(k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q → cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.

Fixpoint cps_lcoefF
(k : term R → (formula R)) (pF : (seq (term R))) : (formula R) :=
  match pF with
  | [::] →
  | c :: q → cps_lcoefF                                              q
  end.
```

Formula level programs

```
Fixpoint cps_lcoef
(k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q → cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.

Fixpoint cps_lcoefF
(k : term R → (formula R)) (pF : (seq (term R))) : (formula R) :=
  match pF with
  | [::] → (k (Const 0))
  | c :: q → cps_lcoefF                                         q
  end.
```

Formula level programs

```
Fixpoint cps_lcoef
(k : R → bool) (p : {poly R}) : bool :=
  match p with
  | [::] → (k 0)
  | c :: q → cps_lcoef (fun l ⇒ if (q == 0) then (k c) else (k l)) q
  end.

Fixpoint cps_lcoefF
(k : term R → (formula R)) (pF : (seq (term R))) : (formula R) :=
  match pF with
  | [::] → (k (Const 0))
  | c :: q → cps_lcoefF (fun l ⇒ ifF (Equal l (Const 0)) (k c) (k l)) q
  end.
```

Formula level programs

```
Definition cps_test (p : {poly R}) : bool :=
cps_lcoef
(fun r ⇒ if r > 0 then true else false)
p
```

Formula level programs

```
Definition cps_test (p : {poly R}) : bool :=
cps_lcoef
(fun r => if r > 0 then true else false)
p

Definition cps_testF (p : seq (term R)) : formula R :=
cps_lcoefF
(fun r => ifF (Lt (Const r) (Const 0)) trueF falseF)
p
```

What happened in this transformation?

Consider an abstract polynomial `pF : seq (term R))`, extracted from a basic formula:

- The concrete shape of this polynomial depends on the values instantiating the parameters.

`(eval_polyF e pF)` denoted `[pF]_e`

What happened in this transformation?

Consider an abstract polynomial `pF : seq (term R))`, extracted from a basic formula:

- The concrete shape of this polynomial depends on the values instantiating the parameters.

  `(eval_polyF e pF)` denoted `[pF]_e`

- Any operation `f` on polynomials has a formula CPS counterpart `fF`.

  `cps_lcoef` and `cps_lcoefF`

What happened in this transformation?

Consider an abstract polynomial `pF : seq (term R))`, extracted from a basic formula:

- The concrete shape of this polynomial depends on the values instantiating the parameters.

  `(eval_polyF e pF)` denoted `[pF]_e`

- Any operation `f` on polynomials has a formula CPS counterpart `fF`.

  `lcoef` and `cps_lcoefF`

- Any test `c` on such a polynomial expression has a formula CPS counterpart `(fF kc)`.

  `cps_testF`

Correctness as observational equivalence

Now we have commutation:

```
Lemma cps_lcoefFP : forall k pF e, acceptable_cont k ->
  qf_sat e (cps_lcoefF k pF)
  =
  qf_sat e (k (Const (lcoef [pF]_e))).
```

A generic and uniform process

- Program the concrete emptiness test for polynomials in $R[X]$;
- For every elementary program used in the previous phase:
  - Turn the concrete program into a CPS-formula one;
  - State the lemma corresponding to its correctness with respect to the concrete program;
  - Prove this lemma by executing symbolically the code of the concrete program in the proof.

Gluing the programs, and the proofs

- Combine the CPS-formula programs in the same way they are combined in the concrete emptiness test program;
- The quantifier elimination procedure of a single $\exists$ follows.
- Combine the CPS-formula correctness lemmas accordingly.
- The correctness proof follows.

Summary

- We have programmed in Coq a generic framework for certified first-order quantifier elimination.
- This framework only requires the theory-dependent proof that a single existential can be eliminated.
- We have found a generic way of designing the proof of this theory-dependant part.
- We have programmed and proved in Coq the cases of algebraically closed fields and real closed fields.

Perspectives

- The formal library on real closed fields is a significant byproduct.
- Some interesting formalization issues are raised by the construction of instances of algebraically closed and real closed fields.
- The main remaining issue is to relate this with the correctness proof of (existing) efficient versions of quantifier elimination algorithms.