Refinement: a reflection on proofs and computations

Cyril Cohen & Damien Rouhling

Based on previous work by Maxime Dénès, Anders Mörtberg & Vincent Siles

Université Côte d’Azur, Inria, France

March 6, 2017
Computers are increasingly used for mathematical proofs, especially for their computational power. For instance:

- The four color theorem [Appel, Haken 1977; Gonthier 2008].
- Kepler conjecture [Hales 2005].
- The odd order theorem [Gonthier et al. 2013].

Different tools with different purposes (really rough approximation):

- Computer algebra software: efficient computations.
- Automatic theorem provers: efficient logical reasoning.
- Interactive theorem provers: sound logical reasoning.

We want to ensure that efficient tools use sound techniques.

Ease of use matters.

We will focus on sound and efficient computations.
Motivations

Program verification closes the gap between paper proofs and implementations:

\[(aX^n + b)(cX^n + d) = acX^{2n} + ((a + b)(c + d) - ac - bd)X^n + bd.\]

↓

Program verification

↓

```ocaml
Fixpoint karatsuba_rec n p q := match n with
| 0 => p * q
| n'.+1 =>
    let sp := size p in let sq := size q in
    if (sp <= 2) || (sq <= 2) then p * q else
        let m := (minn sp./2 sq./2) in
        let (a,b) := splitp m p in
        let (c,d) := splitp m q in
        let ac := karatsuba_rec n' a c in
        let bd := karatsuba_rec n' b d in
        let apb := a + b in
        let cpd := c + d in
        let apb_cpd := karatsuba_rec n' apb cpd in
        (shiftp (2 * m) ac + (shiftp m (apb_cpd - ac - bd))) + bd
    end.
```

Definition karatsuba p q :=

karatsuba_rec (maxn (size p) (size q)) p q.
Computations shorten proof terms and make the users’ life easier.

- $1 + (2 + 3) = 6$ by reflexivity instead of using the rules:
  
  $n + 0 = n$.
  
  $n + (S \ m) = S (n + m)$.

- $M$ is invertible iff $\det M$ is not 0.
Separation of concerns

Issues:

- Efficient algorithms are often hard to prove correct. For instance: the Sasaki-Murao algorithm [Coquand, Mörtberg, Siles 2012].
- Structures that are adapted to proofs are often inefficient for computations. For instance in Coq: nat or Mathematical Components polynomials.
- We do not want to develop a theory for each representation of the same object.

Ideal world:

1. Develop one theory using well-adapted structures independently of what people want to compute with them.
2. Reuse this theory to get proofs on more complex structures.
Outline

1. CoqEAL’s refinement framework
2. Automation
3. Applications
Sequence of refinement steps

\[ P_1 \rightarrow P_2 \rightarrow \cdots \rightarrow P_n \]

where:

**In the literature**
- \( P_1 \) is an abstract version of the program.
- \( P_n \) is a concrete version of the program.

**In CoQ\textsc{EAL}**
- \( P_1 \) is a proof-oriented version of the program.
- \( P_n \) is a computation-oriented version of the program.

- Each \( P_i \) is correct w.r.t. \( P_{i-1} \).
Two kinds of refinement

We distinguish two kinds of refinement:

- **Program refinement**: improve the algorithms without changing the data structures.
- **Data refinement**: use the same algorithms on more efficient data representations and primitives.

An important property for data refinement: *compositionality*. 
Example: Karatsuba’s algorithm

Program refinement:

Karatsuba’s algorithm is an algorithm for fast polynomial multiplication \(O(n^{\log_2 3})\) inspired from the following equation:

\[(aX^n + b)(cX^n + d) = acX^{2n} + ((a + b)(c + d) - ac - bd)X^n + bd.\]

Specification

Lemma \textbf{karatsubaE} : forall p q : \{poly A\},
\[\text{karatsuba } p \ q = p \ast \{\text{poly A} \} \ q.\]
Example: Horner’s polynomials

Data refinement:

Inductive \texttt{hpoly} \ A :=
\hspace*{2cm} | \texttt{Pc} : \ A \to \texttt{hpoly} \ A
\hspace*{2cm} | \texttt{PX} : \ A \to \texttt{pos} \to \texttt{hpoly} \ A \to \texttt{hpoly} \ A.

\[ aX^n + b \rightarrow \begin{cases} 
\texttt{PX} \ b \ n \ (\texttt{Pc} \ a) \text{ if } n > 0, \\
\texttt{Pc} \ (a + b) \text{ otherwise.}
\end{cases} \]

Refinement relation

Definition \texttt{Rhpoly} \ A : \texttt{\{poly A\}} \to \texttt{hpoly A} \to \texttt{Type} :=
\hspace*{2cm} \texttt{fun} \ p \ \texttt{hp} \Rightarrow \texttt{to_poly hp} = p.
Example: Horner’s polynomials (cont.)

Compositionality:

**Definition** \texttt{hpoly\_R} A B (R : A \to B \to \text{Type}) :
\texttt{hpoly} A \to \texttt{hpoly} B \to \text{Type} := \ldots

\texttt{Rhpoly} o (\texttt{hpoly\_R} R) : \{\text{poly} A\} \to \texttt{hpoly} B \to \text{Type}
Example: full refinement path

\[ \text{karatsubaE} : \forall A \ (p \ q : \{\text{poly} \ A\}) , \]
\[ \text{karatsuba} \ p \ q = p \ \ast_{\{\text{poly} \ A\}} \ q \]

\[ \text{Rhpoly} : \forall A, \{\text{poly} \ A\} \to \text{hpoly} \ A \to \text{Type} \]

\[ \text{hpoly}_{\ R} : \forall A \ B \ (R : A \to B \to \text{Type}) , \]
\[ \text{hpoly} \ A \to \text{hpoly} \ B \to \text{Type} \]

\[ p \ \ast_{\{\text{poly} \ A\}} \ q \quad \text{karatsuba} \ (\text{hpoly} \ B) \ h' \ h' q' \]

\[ \text{karatsuba} \ \{\text{poly} \ A\} \ p \ q \quad \text{data refinement (\{poly A\})} \]

\[ \text{karatsuba} \ (\text{hpoly} \ A) \ hp \ hq \]

\[ \text{karatsubaE} \] program refinement

\[ \text{data refinement (Rhpoly)} \]

\[ \text{data refinement by compositionality} \]
Outline

1. **CoqEAL’s refinement framework**

2. **Automation**

3. **Applications**
User input.

Requirement: correctness of primitives.

Type classes.

Plugin: PARAMCOQ [Keller, Lasson 2012].
The parametricity theorem \cite{Reynolds1983,Wadler1989}

Relational interpretation for types:

\[
\begin{align*}
[A \rightarrow B] & := \{(f, g) \mid \forall (x, y) \in [A]. (f \, x, g \, y) \in [B]\}, \\
[\forall X. A] & := \{(f, g) \mid \forall R. (f, g) \in [A] \{R/ [X]\}\}.
\end{align*}
\]

Parametricity theorem

For all closed type $A$ and all closed term $t$ of type $A$, there is a term $[t]$ of type $[A] \, t \, t$. Moreover, one can compute $[t]$. 
Example

**Inductive** \( \texttt{hpoly} \ A := \ldots \)

\[
[\texttt{hpoly}] : \forall A, \forall B, \forall R : A \to B \to \text{Type}, \ldots
\]

**Definition** \( \texttt{hpoly\textunderscore R} \ A B (R : A \to B \to \text{Type}) : \text{hpoly} A \to \text{hpoly} B \to \text{Type} := \ldots \)
Example

Inductive \texttt{hpoly} A := ...

\texttt{[hpoly]} : \forall A, \forall B, \forall R : A \to B \to \text{Type}, ...

```
Definition \texttt{hpoly\_R} A B (R : A \to B \to \text{Type}) :
  \texttt{hpoly} A \to \texttt{hpoly} B \to \text{Type} := \texttt{[hpoly]} R.
```
Example (cont.)

\[[\text{karatsuba}] : [\forall P, P \to P \to P] \text{karatsuba karatsuba karatsuba}\]

i.e.

\[[\text{karatsuba}] : \forall P, \forall C, \forall R : P \to C \to \text{Type}, (R \Rightarrow R \Rightarrow R) (\text{karatsuba } P) (\text{karatsuba } C)\]
Refinement inference

A type class for refinement:

```
Class refines P C (R : P -> C -> Type) (p : P) (c : C) :=
  refines_rel : R p c.
```

Program/term synthesis:
We solve by type class inference

```
```

e.g. with `input := 2 *: 'X`, we get

```
?relation := Rhpoly R,
?output := PX 0 1 (Pc 2),
?proof := prf :
  refines (Rhpoly R) (2 *: 'X) (PX 0 1 (Pc 2)).
```
Example

Global goal:

refines ?R (X + Y - (1 * Y)) ?P.

Current goal(s):

refines ?R (X + Y - (1 * Y)) ?P.
Global goal:

refines \(?R\) (X + Y - (1 \* Y)) (\?f \?P1).

Current goal(s):

refines (\(?S\) == \(?R\)) (fun P => X + P) \?f,
refines \(?S\) (Y - (1 \* Y)) \?P1.
Example

Global goal:

refines $?R (X + Y - (1 * Y)) (?g ?P2 ?P1).

Current goal(s):

refines (?T ==> ?S ==> ?R) + ?g,
refines ?T X ?P2,
Global goal:

\[ \text{refines } R \ (X + Y - (1 \times Y)) \ (\text{?P2} + \text{'} \text{?P1}). \]

Assuming

\[ \text{refines } (R \implies R \implies R) \ + \ +\text{'} . \]

Current goal(s):

\[ \text{refines } R \ X \ \text{?P2}, \]
\[ \text{refines } R \ (Y - (1 \times Y)) \ ?P1. \]
Global goal:

refines R (X + Y - (1 * Y)) (X’ +’ ?P1).

Assuming

refines (R ==> R ==> R) + +’,
refines R X X’.

Current goal(s):

refines R (Y - (1 * Y)) ?P1.
Example

Proven:

\[\text{refines } R \ (X + Y - (1 \ast Y)) \ (X' +' Y' -' (1' \ast' Y'))\].

Assuming

\[\text{refines } (R \Rightarrow R \Rightarrow R) \ + \ +',\]
\[\text{refines } (R \Rightarrow R \Rightarrow R) \ - \ -',\]
\[\text{refines } (R \Rightarrow R \Rightarrow R) \ast \ast',\]
\[\text{refines } R \ X \ X',\]
\[\text{refines } R \ Y \ Y',\]
\[\text{refines } R \ 1 \ 1'.\]
Logic programming for refinement

Rules to decompose expressions, such as

**Instance** *refines_apply*

\[
P \ C \ (R : P \rightarrow C \rightarrow \text{Type}) \ P' \ C' \ (R' : P' \rightarrow C' \rightarrow \text{Type}) : \\
\text{forall} \ (f : P \rightarrow P') \ (g : C \rightarrow C'), \\
\text{refines} \ (R \Rightarrow R') \ f \ g \rightarrow \\
\text{forall} \ (p : P) \ (c : C), \text{refines} \ R \ p \ c \rightarrow \\
\text{refines} \ R' \ (f \ p) \ (g \ c).
\]

**Lemma** *refines_trans*

\[
P \ I \ C \ (rPI : P \rightarrow I \rightarrow \text{Type}) \\
(rIC : I \rightarrow C \rightarrow \text{Type}) \ (rPC : P \rightarrow C \rightarrow \text{Type}) \\
(p : P) \ (i : I) \ (c : C) : \\
rPI \circ rIC \leq rPC \rightarrow \\
\text{refines} \ rPI \ p \ i \rightarrow \text{refines} \ rIC \ i \ c \rightarrow \\
\text{refines} \ rPC \ p \ c.
\]
refines ?R (matrix_of_fun (fun i j => i + (i * j))) ?m
refines ?R (matrix_of_fun (fun i j => i + (i * j))) ?m

Assume

refines Rord i i’,
refines Rord j j’.

Global goal:

refines ?R (i + (i * j)) (?f i’ j’).

Current goal(s):

refines ?R (i + (i * j)) (?f i’ j’).
Example

refines ?R (matrix_of_fun (fun i j => i + (i * j))) ?m

Assume

refines Rord i i',
refines Rord j j'.

Global goal:

refines ?R (i + (i * j)) (?f i' j').

Current goal(s):

refines (?R' ==> ?R) (fun k => i + k) (?f i'),
refines ?R' (i * j) j'.
Example

```
refines ?R (matrix_of_fun (fun i j => i + (i * j))) ?m
```

Assume

```
refines Rord i i',
refines Rord j j'.
```

**Solution:**

```
Class unify A (x y : A) := unify_rel : x = y.
Instance unifyxx A (x : A) : unify x x := erefl.
```

With the goal:

```
refines (?R o unify) (i + (i * j)) (?f i' j'),
```

which splits into

```
refines ?R (i + (i * j)) ?e,
refines unify ?e (?f i' j').
```
Outline

1. CoqEAL’s refinement framework
2. Automation
3. Applications
Proofs by computation

**Definition ctmat1** : \( \text{M}[\text{int}](3, 3) := \)
\[
\text{\textbackslash matrix}_(i, j) ([:: [:: 1 \ ; 1 \ ; 1 ] \\
; [:: -1 \ ; 1 \ ; 1 ] \\
; [:: 0 \ ; 0 \ ; 1 ] ]^i)^j.
\]

**Lemma det_ctmat1** : \( \text{\textbackslash det \ ctmat1} = 2. \)

**Proof.**
by do ![rewrite (expand_det_row _ ord0) //=; rewrite ?(big_ord_recl,big_ord0) //= ?mxE //=; rewrite /cofactor /= ?(addn0, add0n, expr0, exprS); rewrite ?(mul1r,mulr1,mulN1r,mul0r,mul1r,addr0) /=; do ?rewrite [row’ _ _]mx11_scalar det_scalar1 !mxE /=]. Qed.
Proofs by computation

**Definition ctmat1** : 'M[int]_(3, 3) :=
\matrix_((i, j)) ([:: [:: 1 ; 1 ; 1 ]
; [:: -1 ; 1 ; 1 ]
; [:: 0 ; 0 ; 1 ]]_'_i)_'_j.

**Lemma det_ctmat1** : \det ctmat1 = 2.
Proof. by coqeval. Qed.

or

**Definition det_ctmat1** :=
[coqeval vm_compute of \det ctmat1].
--> det_ctmat1 : \det ctmat1 = 2
Lemma \texttt{refines_spec} \( R \ p \ c : \text{refines} \ R \ p \ c \rightarrow p = \text{spec} \ c. \)

\[
\begin{align*}
X + Y - (1 \times Y) \\
\text{rewrite} \ \text{refines_spec} \\
\text{spec} \ (X' + Y' - (1' \times Y')) \\
\text{vm\_compute} \ \text{under} \ \text{spec} \\
\text{spec} \ X' \\
\text{simpl} \\
X
\end{align*}
\]
Lemma **refines_spec** $R \, p \, c : \text{refines } R \, p \, c \rightarrow p = \text{spec } c.$
Proof by reflection

- Use computation to automate and to shorten proofs.
- Issue: ad-hoc computation-oriented data-structures and problem-specific implementations make it hard to maintain and improve reflection-based tactics.
- Our contribution: a modular reflection methodology that uses generic tools to minimise the code specific to a given tactic.
- Our example case:
  - The `ring Coq` tactic: a reflection-based tactic to reason modulo ring axioms (and a bit more).
  - Generic tools: the `Mathematical Components` library and `CoqEAL` refinement framework.
  - Code specific to our prototype: around 200 lines.
Reflection

Source structure \rightarrow \text{metaification} \rightarrow \text{Target structure (inductive type)}

Source structure \leftarrow \text{reflection/interpretation} \leftarrow \text{Target structure (canonical form)}

\[\text{computation}\]
Metaification:
Symbolic arithmetic expressions in a ring (using \+, \-, \* and \(^n\)) can be represented as multivariate polynomials over integers, together with a variable map.
\(a + b - (1 \* b) \rightarrow X + Y - (1 \* Y)\) with variable map \([a; b]\).

Computation:
The goal of the computation step is to normalise the obtained polynomials.
\(X + Y - (1 \* Y) \rightarrow X\).

Reflection:
The polynomials in normal form are evaluated on the variable map to get back ring expressions.
\(X[a; b] \rightarrow a\).
The ring of integers is a canonical choice since there is a canonical injection from integers to any ring: the ring of integers is an initial object of the category of rings.

However it may happen that another ring \((\mathbb{Z}/n\mathbb{Z}, \text{rational numbers} \ldots)\) is a better choice. For instance \(a + a = 0\) is provable in the ring of booleans, using the ring of booleans itself as the ring of coefficients.

\[
a + a \rightarrow (X + X)[a] \rightarrow ((1 + 1)X)[a] \rightarrow (0X)[a] \rightarrow 0.
\]
The coqeqal\_ring tactic [Cohen, Rouhling 2017]

\[
a + b - (1 \times b)
\]

rewriting

\[
eval (X + Y - (1 \times Y)) [a; b]
\]

[coqeqal \texttt{vm\_compute of } _]}

rewriting

\[
eval X [a; b]
\]

rewriting

\[
a
\]
The coqear\_ring tactic [Cohen, Rouhling 2017]

- Arithmetic expression
  \[\text{coqear}_\text{ring}\]
  \[\text{depolyification}\]

- Proof-oriented polynomial
  \[[\text{coqear \text{vm\_compute of \_}}]\]
  \[\text{spec and simpl}\]

- Computation-oriented polynomial
  \[\text{vm\_compute}\]

- AST
  \[\text{computation}\]
  \[\text{polyfication}\]

- Metafication

Damien Rouhling
Proofs vs Computations
March 6, 2017
The ring tactic [Grégoire, Mahboubi 2005]

Arithmetic expression  \rightarrow \text{metaification} \rightarrow \text{Abstract Syntax Tree (AST)}

\text{ring} \downarrow

\downarrow \text{interpretation}

Arithmetic expression \rightarrow \text{Sparse Horner polynomial}
Comparison

Arithmetic expression → AST → Proof-oriented polynomial → Sparse Horner polynomial

- Meta-interpretation
- Polyfication
- Computation
- Depolyfication
- Specification and simplification
- Interpretation

- Ring
- CoqEqRing
- [CoqEq vm_compute of ]
Further work

- On coqeqal_ring:
  - Catch up with `ring`: operations such as the power function, ring of coefficients as parameter, non-commutative rings, semi-rings...
  - Make coqeqal_ring efficient: refinement of the translation $\text{AST} \rightarrow \text{polynomial}$, improved depolyfication.
  - Implement new features: morphisms, Gröbner bases (Théry, using multivariate polynomials by Strub, and a refinement by Martin-Dorel, Roux), user-defined operations...
  - Generalise to other decision procedures: `field? lra`??

- On CoQEAL:
  - More refinements, especially outside algebra, e.g. finite sets (Dagand, Gallego Arias).
  - Improve CoQEAL’s interface, e.g. a better debugging system.
  - Make refinement faster, in particular on nested structures.
Conclusion

- Efficient computations require proofs, refinement simplifies them.
- Proofs are automated by computations, reflection does that.
- Refinement is not so far from reflection.
Conclusion

- Efficient computations require proofs, refinement simplifies them.
- Proofs are automated by computations, reflection does that.
- Refinement is not so far from reflection.

Thank you!
Generic programming

From

```plaintext
Record rat : Set := Rat {
  valq : int * int ;
  _ : (0 < valq.2) && coprime '|valq.1| |valq.2|
}
```

to

```plaintext
Definition Q Z := Z * Z.
```

Generic operation

```plaintext
Definition addQ Z +_ *_ : Q Z -> Q Z -> Q Z :=
  fun x y => (x.1 *_ y.2 +_ y.1 *_ x.2, x.2 *_ y.2).
```
Correctness of addQ

- Proof-oriented correctness: instantiate $\mathbb{Z}$ with $\mathbb{int}$.
- Relation $R_{rat}$: $\text{rat} \rightarrow \mathbb{Q} \quad \text{int} \rightarrow \text{Type}$.
- Prove the following theorem:

  Lemma $R_{rat\_addQ}$:
  \[(R_{rat} \implies R_{rat} \implies R_{rat}) +_{rat} (\text{addQ int} +_{\mathbb{int}} *_{\mathbb{int}}).\]
Correctness of addQ (cont.)

Generalization using compositionality: from the refinement relation $\text{Rint} : \text{int} \to \mathbb{C} \to \text{Type}$,

Definition $\text{RratC} : \text{rat} \to \mathbb{C} \times \mathbb{C} \to \text{Type} := \text{Rrat} \circ (\text{Rint} \times \text{Rint})$.

Goal:

Lemma $\text{RratC_add} :$

$(\text{RratC} \implies \text{RratC} \implies \text{RratC}) +_{\text{rat}} (\text{addQ C} +_{\text{C}} \times_{\text{C}})$. 
Correctness of addQ (cont.)

Generalization using compositionality: from the refinement relation
Rint : int -> C -> Type,

Definition \textbf{RratC} : rat -> C * C -> Type :=
Rrat o (Rint * Rint).

Goal:

Lemma \textbf{RratC_add} :
(RratC ==> RratC ==> RratC) +_{\text{rat}} (addQ C +_{C} *_{C}).

This splits into

(Rrat ==> Rrat ==> Rrat) +_{\text{rat}} (addQ int +_{\text{int}} *_{\text{int}}),

already proven and

(Rint * Rint ==> Rint * Rint ==> Rint * Rint)
(addQ int +_{\text{int}} *_{\text{int}}) (addQ C +_{C} *_{C}).
Correctness of addQ (end)

Goal:

\[(\text{Rint} \ast \text{Rint} \Rightarrow \text{Rint} \ast \text{Rint} \Rightarrow \text{Rint} \ast \text{Rint})
  \quad (\text{addQ} \int +_{\int} *_{\int})
  \quad (\text{addQ} \ C +_{\ C} *_{\ C}).\]
Correctness of addQ (end)

Goal:

\[(\text{Rint} \times \text{Rint} \Rightarrow \text{Rint} \times \text{Rint}) \Rightarrow \text{addQ} \times \text{addQ} \times \text{addQ})(\text{Rint} \times \text{Rint} \Rightarrow \text{Rint} \times \text{Rint})(\text{addQ} \times \text{addQ} \times \text{addQ})\].

By parametricity:

\[\forall Z. (Z \rightarrow Z \rightarrow Z) \rightarrow (Z \rightarrow Z \rightarrow Z) \rightarrow Z \times Z \rightarrow Z \times Z \rightarrow Z \times Z]\ addQ addQ,

i.e.

\[\forall Z : \text{Type}. \forall Z' : \text{Type}. \forall R : Z \rightarrow Z' \rightarrow \text{Type}. \forall \text{addZ} : Z \rightarrow Z \rightarrow Z. \forall \text{addZ'} : Z' \rightarrow Z' \rightarrow Z'. \forall \text{mulZ} : Z \rightarrow Z \rightarrow Z. \forall \text{mulZ'} : Z' \rightarrow Z' \rightarrow Z'. \forall \text{addZ} \text{addZ'} \rightarrow
\forall \text{mulZ} \text{mulZ'} \rightarrow
(R \Rightarrow R \Rightarrow R) \text{addZ} \text{addZ'} \rightarrow
(R \Rightarrow R \Rightarrow R) \text{mulZ} \text{mulZ'} \rightarrow
\text{(addQ} \text{Z addZ mulZ)} \text{(addQ} \text{Z' addZ'} \text{mulZ')}\].
Soundness of polyfication

Lemma **polyfication**\(^\text{P}\) (R : comRingType) (env : seq R) N p : size env == N \(\rightarrow\)

\[
PExpr_to_Expr\ env\ p = \text{Nhorner}\ env\ (PExpr_to_poly\ N\ p).
\]

Proof.

elim: p\(\Rightarrow\) [n\|n]\ p\ \text{IHp}\ q\ \text{IHq}\ p\ \text{IHq}\ q\ \text{IHp}\ p\ \text{IHp}\ n] \(\neq\).

- by rewrite NhornerE !rmorph_int.
- rewrite NhornerE; elim: N env n\(\Rightarrow\) [\[N\ \text{IHN}\] \[\|a\ \text{env}\] \[\|n\] \(\neq\) \ senv.
  
  by rewrite map_polyX hornerX [RHS]NhornerRC.
  by rewrite map_polyC hornerC !IHN.
- by move\(\Rightarrow\) senv; rewrite (IHp senv) (IHq senv) !NhornerE !rmorphD.
- by move\(\Rightarrow\) senv; rewrite (IHp senv) (IHq senv) !NhornerE !rmorphM.
- by move\(\Rightarrow\) senv; rewrite (IHp senv) !NhornerE !rmorphN.
- by move\(\Rightarrow\) senv; rewrite (IHp senv) !NhornerE !rmorphX.

Qed.
Example of user-defined operation: factoring

Where $P[a] = 0$. 