Refining the `ring` tactic

Cyril Cohen & Damien Rouhling

Université Côte d'Azur, Inria, France

January 6, 2017

Motivations

- 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.

Sequence of refinement steps

$$P_1 \to P_2 \to \cdots \to P_n$$

where:

In the literature

- P_1 is an abstract version of the program,
- P_n is a concrete version of the program,

In CoqEAL

- P_1 is an 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}.

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

```
?proof : refines ?relation input ?output.
```

Back and forth translation:

```
Lemma refines_spec R p c : refines R p c -> p = spec c.
```

The coqeal_simpl tactic

```
Lemma refines_spec R p c : refines R p c -> p = spec c.
```

X + Y - (1 * Y)

↓ rewrite refines_spec

spec (X' +' Y' -' (1' *' Y'))

↓ Computation under spec

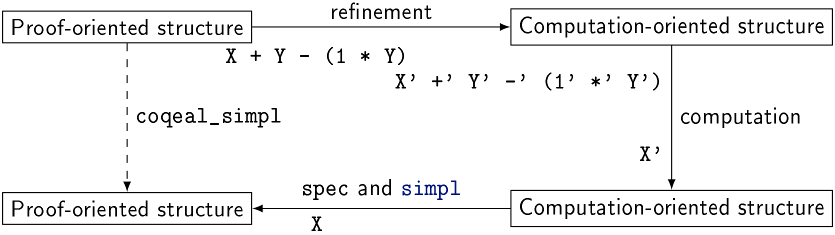
spec X'

↓ simpl

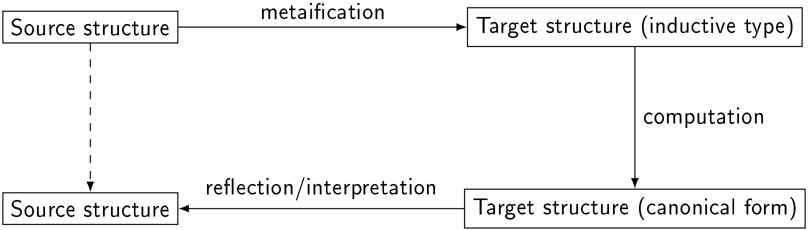
X

The coqeal_simpl tactic

```
Lemma refines_spec R p c : refines R p c -> p = spec c.
```



Reflection [Boutin 97]



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)` $\longrightarrow$ `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)` $\longrightarrow$ `X`.

Reflection:

The polynomials in normal form are evaluated on the variable map to get back ring expressions.

`X[a; b]` $\longrightarrow$ `a`.

The coqeal_ring tactic

`a + b - (1 * b)`

↓ rewriting

`eval (X + Y - (1 * Y)) [a; b]`

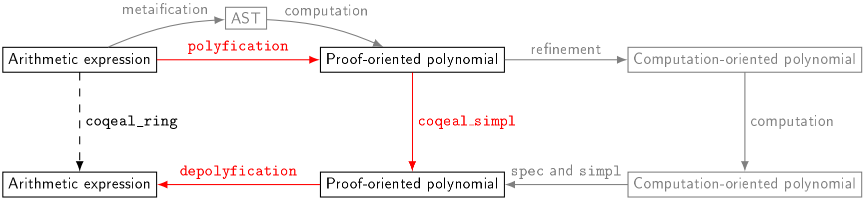
↓ coqeal_simpl

`eval X [a; b]`

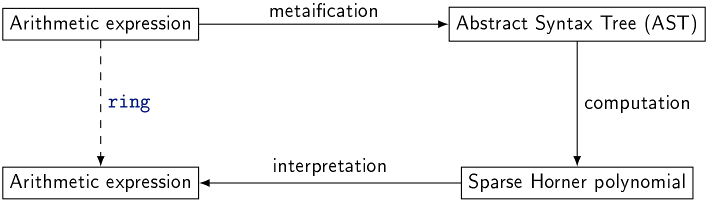
↓ rewriting

`a`

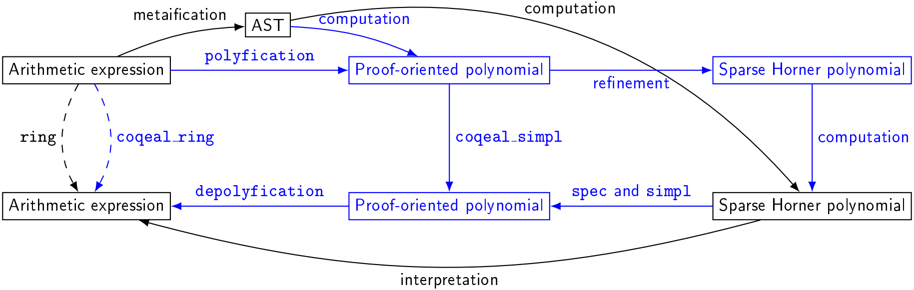
The coqeal_ring tactic



The `ring` tactic [Grégoire, Mahboubi 2005]



Comparison



metaification
AST
computation
computation
Arithmetic expression
polyfication
Proof-oriented polynomial
Sparse Horner polynomial
refinement
ring
coqeal_ring
coqeal_simpl
computation
Arithmetic expression
depolyfication
Proof-oriented polynomial
spec and simpl
Sparse Horner polynomial
interpretation

Further work

- Catch up with `ring`: operations such as the power function, ring of coefficients as parameter, non-commutative rings, semi-rings...
- Make `coqeal_ring` efficient: refinement of nested data-structures, improved `depolyfication`.
- Implement new features: morphisms, Gröbner bases, other reduction strategies, user-defined operations...
- Generalise to other decision procedures: `field`? `lra`???

Conclusion

- A more modular reflection methodology.
- Refinement makes the reduction step easier to prove.
- Semantic vs syntactic translation.
- A prototype still in its early conception phase.

Conclusion

- A more modular reflection methodology.
- Refinement makes the reduction step easier to prove.
- Semantic vs syntactic translation.
- A prototype still in its early conception phase.

Thank you!

Logic programming for refinement

Rules to decompose expressions, such as

```
Instance refines_apply
A B (R : A -> B -> Type) A' B' (R' : A' -> B' -> Type) :
  forall (f : A -> A') (g : B -> B'),
  refines (R ==> R') f g ->
    forall (a : A) (b : B), refines R a b ->
      refines R' (f a) (g b).

Lemma refines_trans A B C (rAB : A -> B -> Type)
(rBC : B -> C -> Type) (rAC : A -> C -> Type)
(a : A) (b : B) (c : C) :
  composable rAB rBC rAC ->
    refines rAB a b -> refines rBC b c ->
      refines rAC a c.
```

Example

Global goal:

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

Current goal(s):

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

Example

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,
refines ?S (Y - (1 * Y)) ?P1.
```

Example

Global goal:

```
refines R (X + Y - (1 * Y)) (?P2 +' ?P1).
```

Assuming

```
refines (R ==> R ==> R) + +'.
```

Current goal(s):

```
refines R X ?P2,
refines R (Y - (1 * Y)) ?P1.
```

Example

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:

```
refines R (X + Y - (1 * Y)) (X' +' Y' -' (1' *' Y')).
```

Assuming

```
refines (R ==> R ==> R) + +',
refines (R ==> R ==> R) - -',
refines (R ==> R ==> R) * *',
refines R X X',
refines R Y Y',
refines R 1 1'.
```

The ring of coefficients

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.

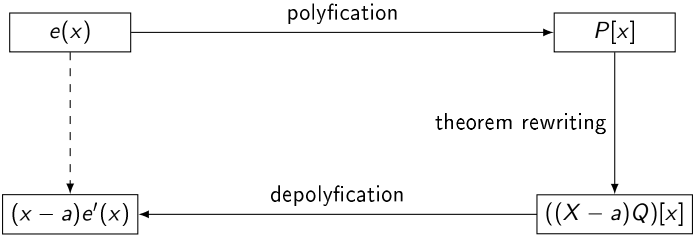
However it may happen that another ring ($\mathbb{Z}_{/n\mathbb{Z}}$, rational numbers...) is a better choice. For instance `a + a = 0` is provable in the ring of booleans, using the ring of booleans itself as the ring of coefficients.

$$\texttt{a + a} \longrightarrow \texttt{(X + X)[a]} \longrightarrow \texttt{((1 + 1)X)[a]} \longrightarrow \texttt{(0X)[a]} \longrightarrow \texttt{0.}$$

Soundness of polyfication

```
Lemma polyficationP (R : comRingType) (env : seq R) N p : size env == N ->
  PExpr_to_Expr env p = Nhorner env (PExpr_to_poly N p).
Proof.
elim: p=> [n|n|p IHp q IHq|p IHp q IHq|p IHp|p IHp n] /=.
- by rewrite NhornerE !rmorph_int.
- rewrite NhornerE; elim: N env n=> [|N IHN] [|a env] [|n] //= senv.
    by rewrite map_polyX hornerX [RHS]NhornerRC.
  by rewrite map_polyC hornerC !IHN.
- by move=> senv; rewrite (IHp senv) (IHq senv) !NhornerE !rmorphD.
- by move=> senv; rewrite (IHp senv) (IHq senv) !NhornerE !rmorphM.
- by move=> senv; rewrite (IHp senv) !NhornerE !rmorphN.
- by move=> senv; rewrite (IHp senv) !NhornerE !rmorphX.
Qed.
```

Example of user-defined operation: factoring



Where $P[a] = 0$.