The Rewster: The Coq Proof Assistant with Rewrite Rules

Yann Leray, joint work with Gaëtan Gilbert, Nicolas Tabareau and Théo Winterhalter

14th June 2023

Usage

```
Symbol pplus : ℕ → ℕ → ℕ.
Infix "+" := pplus.

Rewrite Rule [ n ] ⊢ 0 + n ⟹ n.
Rewrite Rule [ m n ] ⊢ S m + n ⟹ S (m + n).
Rewrite Rule [ n ] ⊢ n + 0 ⟹ n.
Rewrite Rule [ m n ] ⊢ m + S n ⟹ S (m + n).

Eval compute in fun a b c ⇒ S a + b + S (S c).
(* fun a b c ⇒ S (S (S (a + b + c))) *)
```

```
Symbol raise : ∀ (A : Type), A.

Rewrite Rule [ A B x ] ⊢
 raise (∀ (a : A), B a) x ⟹ raise (B x).

Rewrite Rule [ P ] ⊢
 match raise 𝔹 as b return P b with
 | true ⇒ _ | false ⇒ _
 end ⟹ raise (P (raise 𝔹)).
```

Prototype available on GitHub (linked at the end)

Pattern expressiveness

```
pat ::= □ | Ind | Constr | Sort | Symb
  | pat pat
  | pat .(proj)
  | match pat in Ind pat… return pat with C1 ⇒ pat | … end
  | fun (x : pat) ⇒ pat
  | ∀ (x : pat), pat
```

Previous examples :

```
Rewrite Rule [ m n ] ⊢ m + S n ⟹ S (m + n).

Rewrite Rule [ A B x ] ⊢ raise (∀ (a : A), B a) x ⟹ raise (B x).

Rewrite Rule [ P ] ⊢
 match raise 𝔹 as b return P b with
 | true ⇒ _ | false ⇒ _
 end ⟹ raise (P (raise 𝔹)).
```

Plan

- Extensions from the previous work
  1. Higher order, deep symbols
  2. Non-linearity
- Useful/necessary properties for rewriting systems
  1. Confluence
  2. Type preservation
  3. Termination
- Implementation pitfalls

Based on previous work in *The Taming of the Rew* by Cockx, Tabareau and Winterhalter.

MetaCoq

Higher-order rules, deep symbols, ...

Higher order is introduced by `fun x ⇒ pat` but also
`match _ return pat with C1 ⇒ pat end`.

Current theory includes no occurence checking, matched terms have all bound variables in context.

We also allow symbols at any location in pattern, so we have to deal with deep critical pairs.

`Rewrite Rule [ t ] ⊢ dual (dual t) ⟹ t.`

#[`unfoldfix`] : a tag which allows the symbol to unfold fixpoints as if it were a constructor (outside of the theory)

Non-linearity

1. Forced non-linearity

```
Symbol J : ∀ (A : Type) (a : A) (P : A → Type), P a → ∀ (a' : A), eq A a a' → P a'.
Rewrite Rule [ A a P H (a' := a) (A' := A) (a'' := a) ] ⊢ J A a P H a' (eq_refl A' a'') ⟹ H.

Symbol f : ∀ b, if b then ℕ else 𝔹.
Rewrite Rule [ b := true ] ⊢ f b ⟹ 23.
```

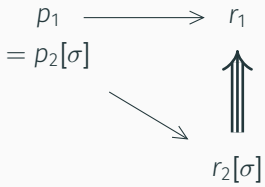
2. General non-linearity

Currently outside of our theory, but useful to represent $\mathrm{TT}^{\mathrm{Obs}}$. There is a proposal for the implementation, it may be added quite soon.

Confluence and the triangle criterion

Confluence of the system is ensured by checking that rewrite rules follow the triangle criterion. (not yet implemented)

1. The rules' LHSs must be closed under pattern unification
2. The RHS associated to one such unified pattern must be a reduct of all RHS associated to the unified patterns for parallel reduction



Rewrite rules and subject reduction

How to ensure that rewrite rules preserve subject reduction ?

Naive solution : check that the LHS and RHS have the same type.

Problem :

```
Symbol id : Type → Type.
Rewrite Rule [ t ] ⊢ id t ⟹ t.
```

Rewrite rules and subject reduction

How to ensure that rewrite rules preserve subject reduction ?

Naive solution : check that the LHS and RHS have the same type.

Problem :

```
Symbol id : Type → Type.
Rewrite Rule [ t ] ⊢ id t ⟹ t.
```

```
Definition U := id Type.
Check U : U. (* accepted *)
```

Rewrite rules and subject reduction

How to ensure that rewrite rules preserve subject reduction ?

Naive solution : check that the LHS and RHS have the same type.

Problem :

```
Symbol id : Type@{i} → Type@{u}.
Rewrite Rule [ t: Type@{i} ] ⊢ id t ⟹ t. (* at Type@{max(u,i)} *)
```

```
Definition U := id Type@{u}.
Check U : U. (* accepted *)
```

Termination

Do we really want termination ?

See Théo Winterhalter's talk on how to instead prove the typechecker correct without assuming termination.

Coq's reduction machines

Coq has (at least) 6 reduction machines (not counting partial reduction machines) :

- Kernel / lazy
- Tactics shared (simpl)
- cbn
- cbv
- Native compute
- VM compute

Coq's reduction machines

Coq has (at least) 6 reduction machines (not counting partial reduction machines) :

- Kernel / lazy (full integration)
- Tactics shared (simpl) (some integration)
- cbn (some integration)
- cbv (some integration)
- Native compute (no support)
- VM compute (no support)

Rewrite rules support varies among them

Pitfall: reduction on arguments

How should pattern-matching and the reduction machine interact, on `a + b + S c` :

Pitfall: reduction on arguments

How should pattern-matching and the reduction machine interact, on `a + b + S c` :

- Weak-head normal form : `Symb +` | `[@a; @(b + S c)]`

Pitfall: reduction on arguments

How should pattern-matching and the reduction machine interact, on $\mathsf{a} + \mathsf{b} + \mathsf{S}\ \mathsf{c}$:

- Weak-head normal form : $\mathtt{Symb} + \ \big|\ [@\mathsf{a}; @(\mathsf{b} + \mathsf{S}\ \mathsf{c})]$
- Fetch associated rewrite rules : $\square + \mathsf{S}\ \square$; as eliminations $[@\square; @(\mathsf{S}\ \square)]$

Pitfall: reduction on arguments

How should pattern-matching and the reduction machine interact, on $\mathtt{a} + \mathtt{b} + \mathtt{S}\ \mathtt{c}$:

- Weak-head normal form : $\mathtt{Symb} + \ \Big|\ [@\mathtt{a};\ @(\mathtt{b} + \mathtt{S}\ \mathtt{c})]$
- Fetch associated rewrite rules : $\square + \mathtt{S}\ \square$; as eliminations $[@\square;\ @(\mathtt{S}\ \square)]$

- Match the arguments :

$@\square$	$@\mathtt{a}$	$\rightarrow$ OK
$@(\mathtt{S}\ \square)$	$@(\mathtt{b} + \mathtt{S}\ \mathtt{c})$	$\rightarrow$ Fail

Pitfall: reduction on arguments

How should pattern-matching and the reduction machine interact, on $\mathtt{a} + \mathtt{b} + \mathtt{S}\ \mathtt{c}$:

- Weak-head normal form : $\mathtt{Symb} + \ \Big|\ [@\mathtt{a};\ @(\mathtt{b} + \mathtt{S}\ \mathtt{c})]$
- Fetch associated rewrite rules : $\square + \mathtt{S}\ \square$; as eliminations $[@\square;\ @(\mathtt{S}\ \square)]$
- Put the arguments in whnf themselves : $[@\ (\mathtt{a}, [])\ ;\ @\ (\mathtt{Symb} +, [@\mathtt{b};\ @(\mathtt{S}\ \mathtt{c})])]$

- Match the arguments :

$@\square$	$@\mathtt{a}$	$\rightarrow$ OK
$@(\mathtt{S}\ \square)$	$@(\mathtt{b} + \mathtt{S}\ \mathtt{c})$	$\rightarrow$ Fail

Pitfall: reduction on arguments

How should pattern-matching and the reduction machine interact, on a + b + S c :

- Weak-head normal form : Symb + | [@a; @(b + S c)]
- Fetch associated rewrite rules : □ + S □ ; as eliminations [@□; @(S □)]
- Put the arguments in whnf themselves : [@ (a, []) ; @ (Symb +, [@b; @(S c)])]
  $\longrightarrow$ [@ (a, []) ; @(S <b + c>)])]
- Match the arguments :

@□	@a	→ OK
@(S □)	@(b + S c)	→ Fail

Pitfall: reduction on arguments

How should pattern-matching and the reduction machine interact, on `a + b + S c` :

- Weak-head normal form : `Symb +` | `[@a; @(b + S c)]`
- Fetch associated rewrite rules : `□ + S □` ; as eliminations `[@□; @(S □)]`
- Put the arguments in whnf themselves : `[@ (a, []) ; @ (Symb +, [@b; @(S c)])]`
  $\longrightarrow$ `[@ (a, []) ; @(S <b + c>)])]`
- Match the arguments :

`@□`	`@a`	$\rightarrow$ OK
`@(S □)`	`@(S <b + c>)`	$\rightarrow$ OK

Future developments

Prototype available at `github.com/Yann-Leray/coq`

(install with `opam pin "git+https://github.com/Yann-Leray/coq#rewrite-rules"`)

1. Complete the work on higher order (confluence, implementation)
2. Implement the confluence checker
3. Develop a criterion for type preservation
4. Other requested extensions (e.g. general non-linearity)

Pitfall: Call-by-value reduction and deep redexes

Normal terms are either neutral or partial redex at (near) toplevel; this is crucial for call-by-value and broken by rewrite rules.

Take a normal term `t` matching pattern `p`, and a rule $\mathtt{p}\ \square \longrightarrow_{\mathsf{rew}} \mathtt{r}$.

Let's study the reduction of (`fun x ⇒ x a`) `t` with cbv :

1. Immediately, `fun x ⇒ x a` is in (weak-)normal form
2. Through recursive examination, `t` is in normal form
3. Beta-reduction to get `(x a)[x := t]`

Problem : cbv expects redexes to appear near toplevel, but `p` (and the head symbol in `t`) may be arbitrarily deep.