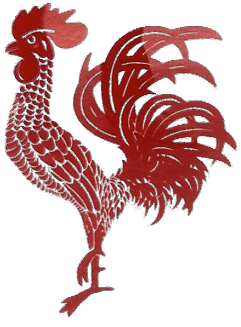
Equations
A toolbox for function definitions in Coq

Matthieu Sozeau & Cyprien Mangin

Inria Paris & IRIF, Université Paris Diderot

March 29th 2016
Dagstuhl, Germany



Overview

- Epigram/Agda-style pattern-matching definitions
- Homogeneous equality (UIP on a type-by-type basis)
- Logical handling of recursion for inductive families
- Propositional equations for defining clauses
- Functional elimination principle

Entirely elaborated to the vanilla kernel

Typical example

```
Equations equal (n m : nat) : { n = m } + { n ≠ m } :=
equal O O := left eq_refl ;
equal (S n) (S m) with equal n m := {
  equal (S n) (S ?(n)) (left eq_refl) := left eq_refl ;
  equal (S n) (S m) (right p) := right _ } ;
equal x y := right _.
```

- Patterns = **well-typed** refinements of the arguments
- We refine the **entire** context at each node
- Can rely on "Uniqueness of Identity Proofs" (axiom K)
- **Inaccessible** patterns + first-match semantics ensure operationality

Dependent Pattern-Matching

Automatically derive, for $\mathsf{I} : \Pi\Delta, \mathtt{Type}$:

- A signature: $\bar{\mathsf{I}} := \Sigma i : \overline{\Delta}.\mathsf{I}\ i$
- **NoConfusion**$_{\bar{\mathsf{I}}}$: discrimination and injectivity of constructors
- **NoCycle**$_{\bar{\mathsf{I}}}$: acyclicity of inductive values
- **EqDec**$_{\bar{\mathsf{I}}}$: decidable equality (if derivable)
- **Subterm**$_{\bar{\mathsf{I}}}$: subterm relation

Dependent Elimination Compilation

Heterogeneous vs homogeneous equality:
To eliminate v in

$$\Gamma = n : \mathbb{N}, v : \mathsf{vector}\ A\ (\mathsf{S}\ n) \vdash \tau$$

We generalize v and its index:

$$\Gamma' = n' : \mathbb{N}, v' : \mathsf{vector}\ A\ n'$$

We also add an equality to get a goal equivalent to the original:

- $\Gamma' \vdash \forall \Gamma, n' = \mathsf{S}\ n \rightarrow \mathsf{JMeq}_{(\mathsf{vector}\ A\ n')\ (\mathsf{vector}\ A\ (\mathsf{S}\ n))}\ v'\ v \rightarrow \tau$
- $\Gamma' \vdash \forall \Gamma, (n', v') =_{\Sigma n:\mathbb{N}.\mathsf{vector}\ A\ n}\ (\mathsf{S}\ n, v) \rightarrow \tau$

Dependent Elimination Compilation

The standard elimination principle applies to v' and we can simplify the equations:

- Using UIP on `Type` when using JMeq. Incompatible with e.g. univalence.
- Using UIP optionally on the index type $\mathbb{N}$ when using telescope equality.

Cockx et al (ICFP'14) give a restrictive criterion on unification to always avoid UIP (implemented in Agda).

We only require typeclass instances of UIP when it is required.

Recursion

- Syntactic guardness checks are too fragile (and buggy)
- Do not work well with abstraction/modularity
- Restricted to structural recursion on a single argument, with no currying allowed

Idea Use the logic instead !

Subterm relations and well-founded recursion

Use **well-founded** recursion on the subterm relation for inductive families $\mathsf{I} : \Pi\ \Delta, \mathtt{Type}$.

Subterm relations and well-founded recursion

Use **well-founded** recursion on the subterm relation for inductive families $\mathsf{I} : \Pi\ \Delta, \mathtt{Type}$.

- General definition of direct subterm:
  $\mathsf{I}_{sub} : \Pi\ \Delta_l\ \Delta_r, \mathsf{I}\ \overline{\Delta_l} \to \mathsf{I}\ \overline{\Delta_r} \to \mathtt{Prop}$
- Define the subterm relation on telescopes:
  $\mathsf{I}_{sub} : \mathrm{relation}\ (\Sigma\ \Delta, \mathsf{I}\ \overline{\Delta})$

Subterm relation example: vectors

```
Derive Subterm for vector.
```

Subterm relation example: vectors

```
Derive Subterm for vector.

Inductive vector_strict_subterm (A : Type)
  : ∀ n m : nat, vector A n → vector A m → Prop :=
    vector_strict_subterm_1_1 : ∀ (a : A) (n : nat) (v : vector A n),
      vector_strict_subterm A n (S n) v (Vcons a v).

Check vector_subterm : ∀ A : Type, relation {index : nat & vector A index}.
```

Subterm relation example: vectors

```
Derive Subterm for vector.

Inductive vector_strict_subterm (A : Type)
  : ∀ n m : nat, vector A n → vector A m → Prop :=
    vector_strict_subterm_1_1 : ∀ (a : A) (n : nat) (v : vector A n),
      vector_strict_subterm A n (S n) v (Vcons a v).

Check vector_subterm : ∀ A : Type, relation {index : nat & vector A index}.

Equations unzip {A B n} (v : vector (A × B) n)
  : vector A n × vector B n :=
unzip A B n v by rec v :=
unzip A B ?(O) Vnil := (Vnil, Vnil) ;
unzip A B ?(S n) (Vcons (pair x y) n v) with unzip v := {
  | (pair xs ys) := (Vcons x xs, Vcons y ys) }.
```

Reasoning support: equations

Goal: keep an abstract view of definitions if desired.

- Equations for the clauses hold definitionally in CCI.
- If UIP is used, only propositionally.
- All put together in a rewrite database, f can be considered opaque.

Reasoning support: elimination principle

```
Equations filter {A} (l : list A) (p : A → bool) : list A :=
filter A nil p := nil ;
filter A (cons a l) p with p a := {
  | true := a :: filter l p ;
  | false := filter l p }.
```

Reasoning support: elimination principle

```
Equations filter {A} (l : list A) (p : A → bool) : list A :=
filter A nil p := nil ;
filter A (cons a l) p with p a := {
  | true := a :: filter l p ;
  | false := filter l p }.

Check (filter_elim :
  ∀ (P : ∀ (A : Type) (l : list A) (p : A → bool), filter_comp l p → Prop),
  let P0 := fun (A : Type) (a : A) (l : list A) (p : A → bool)
      (refine : bool) (res : filter_comp (a :: l) p) ⇒
      p a = refine → P A (a :: l) p res
  in
  (∀ (A : Type) (p : A → bool), P A [] p []) →
  (∀ (A : Type) (a : A) (l : list A) (p : A → bool),
    P A l p (filter l p) → P0 A a l p true (a :: filter l p)) →
  (∀ (A : Type) (a : A) (l : list A) (p : A → bool),
    P A l p (filter l p) → P0 A a l p false (filter l p)) →
  ∀ (A : Type) (l : list A) (p : A → bool), P A l p (filter l p)).
```

Summary

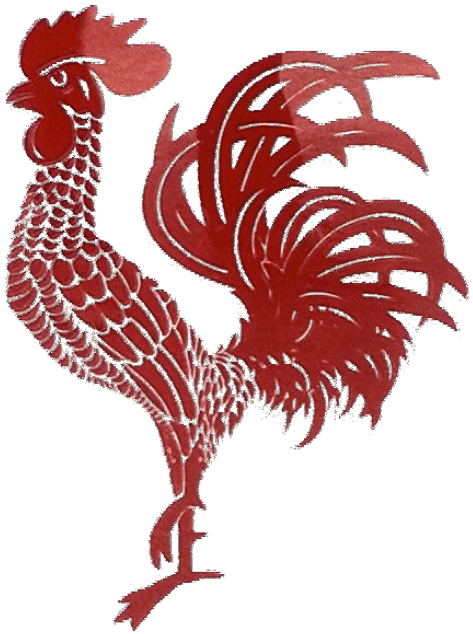
A function definition package handling:

- Full, nested dependent pattern-matching, axiom-free
- Structural and well-founded recursion on inductive families
- Generation of useful support lemmas for reasoning a posteriori

Example formalizations:

- Hereditary substitution for predicative system F
- A. Chlipala's CPDT chapters on dependent pattern-matching
- Data structure of polynomials for a reflexive ring tactic

The End



http://github.com/mattam82/Coq-Equations