Nested, Well-founded & Mutual Recursion in Equations

Matthieu Sozeau, Cyprien Mangin
Inria Paris & IRIF, Université Paris 7 Diderot
1. Equations Reloaded
2. “La Belle et la Bête”¹: Coq’s Guard Condition
3. Logic to the Rescue…

¹ Gabrielle-Suzanne Barbot de Villeneuve (1685 – 1755)
Equations Reloaded

• Dependent Pattern-Matching à la Epigram, Agda
• Compiled-down to CIC using telescope simplification (à la Cockx circa 2016)
• Optional typeclass instances of K/decidable equality
• **Smart** case compilation for smaller proof terms
• Structural and well-founded recursion (i.e. Function/Program like)
• Derive Signature NoConfusion Subterm EqDec for I
• Generates graph, unfolding lemma and **elimination** principles
Coq’s Guard Condition

• Goal: ensure termination statically
• Relatively concise syntactic check (compared to SCT)
• Handles naturally mutual and **nested** fixpoints, e.g:

  ```coq
  Inductive t : Set :=
  | leaf (a : A) : t
  | node (l : list t) : t.
  
  Fixpoint size (r : t) :=
  match r with
  | leaf a ⇒ 1
  | node l ⇒ S (list_size size l)
  end.
  
• Handles **fix-match** decomposition of eliminators, hard(er) with sized-types (A. Abel, B. Grégoire, …)
Trouble with the Guard Condition

- Guard Condition (should) ensure termination
- Slightly hard to understand syntactic criterion. Initial formal justification: Gimenez’94, gradually “sophisticated” since.
- Guard check needs to reduce definitions (!??!) (SN for call-by-name reduction only, WIP fix)
- Buggiest part of the system
  Last bug & fix: #6649 - 24/1/18 - 13:20
- DPM-elimination involves equality manipulations, ...

A Recipe for Disaster
• **Inconsistency with propext (fixed in 2013):**

Hypothesis \( \text{Heq} : (\text{False} \rightarrow \text{False}) = \text{True} \).

Fixpoint \( \text{loop} (u : \text{True}) : \text{False} := \)

\[
\text{loop } \begin{cases} 
\text{match Heq in } (_ = T) & \text{return } T \text{ with} \\
| \text{eq_refl} & \rightarrow \text{fun } f : \text{False} \rightarrow \text{match } f \text{ with end end).}
\end{cases}
\]

• **Typical DPM compilation:**

Inductive \( \text{Split} \{X : \text{Type}\}\{m \ n : \text{nat}\} : \text{vector } X \ (m + n) \rightarrow \text{Type} := \)

append : \( \forall \text{ (xs : vector } X \ m)(\text{ys : vector } X \ n). \text{ Split } (\text{vapp xs ys}) \).

Equations \( \text{split_struct} \{X\} \{m \ n\} (\text{xs : vector } X \ (m + n)) : \text{Split } m \ n \ \text{xs} := \)

\[\text{split_struct} \{m := 0\} \ \text{xs} := \text{append } \text{nil} \ \text{xs} ; \]

\[\text{split_struct} \{m := (S \ m)\} (\text{cons } x \ _ \ \text{xs}) \leftrightarrow \text{split_struct } \text{xs} \Rightarrow \{} \]

\(| \text{append } \text{xs'} \ \text{ys'} := \text{append } (\text{cons } x \ \text{xs'}) \ \text{ys'} \} \).

Not structural on vectors, due to uses of \( J \)
Still, we can handle mutual & nested rec!

http://mattam82.github.io/Coq-Equations/examples/nested_mut_rec.html

Functional elimination is good for you!
OUTLINE

1. Equations Reloaded
2. Beauty & The Beast: Coq’s Guard Condition
3. Logic to the Rescue...
structurally recursive

\[ \subseteq \]

well-founded on subterm relation

1) Derive Subterm for I relation on (computational/hType) inductive families
2) Prove well-foundedness by structural rec
3) Profit! “by rec I\_subterm x”

- Define split on vectors by rec on the vector!
- Extract to general fixpoints
The Beauty of Logic

Equations elements' \( (r : t) \) : list A :=

\[
\text{elements'} l \text{ by rec } r (\text{MR } l \text{t size}) :=
\]

\[
\text{elements'} (\text{leaf } a) := [a];
\]

\[
\text{elements'} (\text{node } l) := fn l \text{ hidebody}
\]

where

\[
\begin{align*}
\text{fn } & (x : \text{list } t) (H : \text{list_size size x } < \text{ size } (\text{node } l)) : \text{list A} := \\
\text{fn } & x H \text{ by rec } x (\text{MR } l \text{t } (\text{list_size size})) := \\
\text{fn } & \text{nil } _ := \text{nil}; \\
\text{fn } & (\text{cons x xs}) _ := \text{elements'} x ++ \text{ fn xs hidebody}.
\end{align*}
\]

- Use the weapon of your choice
- Equations generates unfolding lemma
- Eliminator abstracts away from the w.f. relation: do the work only once.
Closed calls still reduce to the same normal forms: $I_{\text{subterm}}$ is closed

Make it fast by adding $2^n \text{Acc_intro’s}$ to the well-foundedness proof.

For calls on open terms:

- Proofs: unfolding lemma or derived equalities (more control)
- Programs: still reduces, unfolding might be unwieldy though.

Functional extensionality is used to prove the unfolding lemma (easier to automate)
Playtime: Regexp matching

- Implement regexp matching using continuations instead of derivatives or automata (Harper’99 - “Proof-directed debugging”)
- Needs dependent types, well-founded recursion, and eliminator for recursive calls “under binders”...

Demo
More examples

- Hereditary substitution for Predicative System F (Mangin & Sozeau, LFMTP’15)
  Nested recursion, well-founded multiset ordering on types.
- Ordinal measures (Castéran)
- Reflexive ring-like tactic on polynomials. WF subterm order on indexed polynomials
- Prototyping without verifying termination using functional eliminator

mattam82.github.io/Coq-Equations/examples
# opam install coq-equations