The guard condition of Coq

Bruno Barras

December 19, 2006

Why this talk ?

- Defining functions by recursion is very common
- Logical consistency relies heavily on termination
- Reference Manual of Coq refers to Gimenez' paper "Codifying guard definitions with recursive schemes" (94)
- This condition has been extended over the years to support more schemes

Why this talk ?

- Defining functions by recursion is very common
- Logical consistency relies heavily on termination
- Reference Manual of Coq refers to Gimenez' paper "Codifying guard definitions with recursive schemes" (94)
- This condition has been extended over the years to support more schemes
- Bugs (or scary error messages)

Why this talk ?

- Defining functions by recursion is very common
- Logical consistency relies heavily on termination
- Reference Manual of Coq refers to Gimenez' paper "Codifying guard definitions with recursive schemes" (94)
- This condition has been extended over the years to support more schemes
- Bugs (or scary error messages)
  `Uncaught exception: Assert_failure("kernel/inductive.ml",_)`

Overview of the talk

Overview

A long time ago...

- Recursion was made by recursors (Gödel T).
- Only allows recursive calls on *direct* subterms
- Cumbersome in a functional programming setting

Example

```
Definition half n :=
  fst(Rec (0,false)
       (fun (k,odd) ⇒ if odd then (k+1,false)
                      else (k,true))
       n)
```

instead of

```
Fixpoint half n :=
  match n with S(S k) ⇒ half k | _ ⇒ 0 end
```

Towards syntactic guard criterion

- Proposal by Coquand (92):
  recursor = pattern-matching + fixpoint
- Gimenez' paper (94): translation towards recursors.
  For $f : I \longrightarrow T$, define I_f similar to I such that every subterm of type I comes with its image by f. Then write $g : I \longrightarrow I_f$ and $h : I_f \longrightarrow T$.
- Blanqui (05), Calculus of Algebraic Constructions: reducibility proof (CC + higher order rewriting)
- Only work for simple criterion.

Positivity condition

- Also crucial for consistency
- Lists

```
Inductive list (A:Type) :  Type :=
   nil | cons (x:A) (l:list A).
```

- Ordinals `Inductive ord:Set :=`

```
   O | S(o:ord) | lim(f:nat→ord).
```

- Useful extension: nested inductive types

```
Inductive tree:Set := None(l:list tree).
```

Reuse list library

Positivity condition (more formally)

Definition (Terms)

$$\begin{aligned}&s \mid x \mid \Pi x : T.\, U \mid \lambda x : T.\, M \mid M\ N\\ &\mid \mathrm{Ind}(X : A)\{\vec{C}\} \mid \mathrm{Constr}(n, I) \mid \mathrm{Fix}\ F_k : T := M\\ &\mid \mathrm{Match}\ M\ \mathrm{with}\ \vec{p} \Rightarrow \vec{t}\ \mathrm{end}\end{aligned}$$

Definition (strict positivity)

$\Pi\vec{x} : \vec{t}.\, C$ is strictly positive w.r.t. X if forall i either:

(Norec) X does not occur free in t_i, or

(Rec) $t_i = \Pi\vec{y} : \vec{u}.\, X\ \vec{w}$ where X does not occur in $\vec{u}\vec{w}$, or

(Nested) $t_i = \Pi\vec{y} : \vec{u}.\, \mathrm{Ind}(Y : B)\{\vec{D}\}\ \vec{w}$ and

- X does not occur free in $\vec{u}\vec{w}$
- D_i is strictly positive w.r.t. X forall i

Impredicativity

Recursive calls cannot be allowed on *all* constructor arguments

```
Inductive I : Set := C (f:forall A:Set,A->A).
Fixpoint F (x:I) : False :=
  match x with
    C f => F (f I x)
  end
```

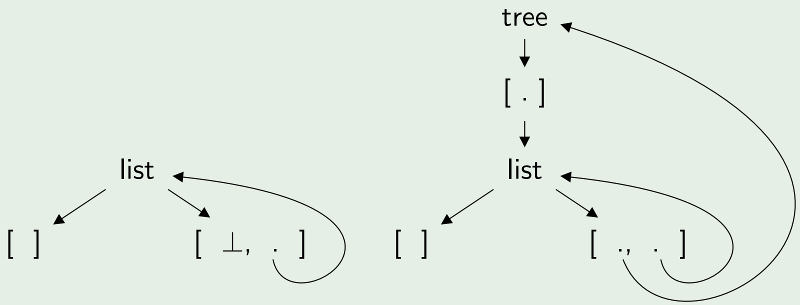
Definition (recursive positions)

constructors arguments that satisfy (Rec) or (Nested) clause of positivity.

Regular trees

- Different instances of the same inductive type may have different sets of recursive positions

Example ($\mathrm{Str}(\textit{list})$ and $\mathrm{Str}(\textit{tree})$)



Trees as sets of paths

While checking positivity, we build a regular tree that identifies recursive positions.
But: parameters not instanciated

Lemma

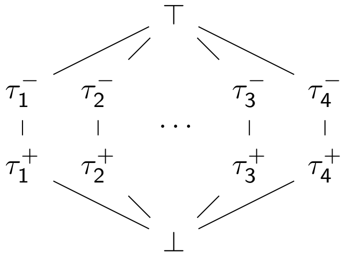
The computed tree is the set of paths that cannot contain an infinite number of inductive objects.

Overview

Size information

$$
\begin{aligned}
\text{(strict)}\quad & \sigma^- ::= \top \mid \tau^- \\
\text{(non-strict)}\quad & \sigma^+ ::= \bot \mid \tau^+ \\
\text{(size info)}\quad & \sigma ::= \sigma^+ \cup \sigma^-
\end{aligned}
$$

A map ρ associates size information to every variable



Guard condition in short

- A judgement $\rho \vdash^{S} M \Rightarrow \sigma$ meaning that M has size information σ, where ρ associates size information to variables
- A judgement $M \in \mathrm{Check}_{\rho}^{F,k}$ meaning that M does recursive calls to F only on strict subterms, as specified by ρ
- Pattern-matching propagates information on pattern variables

$$\mathrm{Constr}(i, I)\ x_1 \ldots x_k \mid \sigma = \{(x_j, \sigma.i.j^-) \mid j \leq k\}$$

Remarks

- Easy encoding of recursors as fix+match (non regression)
- Allow recursive calls on deep subterms

Definition of the condition (1)

Typing rule:

$$\frac{\Gamma\,(F : T) \vdash M : T \qquad M \in \mathrm{Guard}_k^F}{\Gamma \vdash (\mathrm{Fix}\ F_k : T := M) : T}$$

$$\frac{t_k = \mathrm{Ind}(X : A)\{\vec{C}\}\ \vec{u} \qquad \mathrm{Str}(X, \vec{C}) = \tau \qquad M \in \mathrm{Check}^{F,k}_{\{(x_k,\tau^+)\}}}{\lambda \vec{x} : \vec{t}.\, M \in \mathrm{Guard}_k^F}$$

Definition of the condition (2)

$$\frac{M \in \mathrm{Check}^{f,k}_{\rho} \quad \rho \vdash^{S} M \Rightarrow \sigma \quad \forall i.\, b_i \in \mathrm{Check}^{f,k}_{\rho \cup (p_i | \sigma)}}{\text{Match } M \text{ with } \vec{p} \Rightarrow \vec{b} \text{ end} \in \mathrm{Check}^{f,k}_{\rho}}$$

$$\frac{\rho \vdash^{S} t_k \Rightarrow \sigma^{-} \quad \forall i,\, t_i \in \mathrm{Check}^{f,k}_{\rho}}{f\ \vec{t} \in \mathrm{Check}^{f,k}_{\rho}}$$

Definition of the condition (boring cases)

Simply check recursively that subexpressions are guarded

$$\frac{f \notin FV(M)}{M \in \mathrm{Check}_\rho^{f,k}} \qquad \frac{T \in \mathrm{Check}_\rho^{f,k} \quad U \in \mathrm{Check}_\rho^{f,k}}{\Pi x : T\ U \in \mathrm{Check}_\rho^{f,k}}$$

$$\frac{T \in \mathrm{Check}_\rho^{f,k} \quad U \in \mathrm{Check}_\rho^{f,k}}{\lambda x : T\ U \in \mathrm{Check}_\rho^{f,k}} \qquad \frac{M \in \mathrm{Check}_\rho^{f,k} \quad N \in \mathrm{Check}_\rho^{f,k}}{M\ N \in \mathrm{Check}_\rho^{f,k}}$$

Subterms

$$\frac{(x, \sigma) \in \rho}{\rho \vdash^{S} x\ \vec{t} \Rightarrow \sigma} \qquad \frac{\rho \vdash^{S} M \Rightarrow \sigma}{\rho \vdash^{S} \lambda x : A.\, M \Rightarrow \sigma}$$

Overview

Checking guard modulo reduction

In fact, the typing rule for fixpoints is:

$$\frac{\Gamma\,(F : T) \vdash M : T \quad M \rightarrow_\beta^* M' \quad M' \in \mathrm{Guard}_k^F}{\Gamma \vdash (\mathrm{Fix}\ F_k : T := M) : T}$$

Breaks strong normalization!

Example

```
Fixpoint F n := let x := F n in 0.
Eval compute in (F 0).
```

Pattern-matching

$$\frac{\forall i, \rho \vdash^S b_i \Rightarrow \sigma_i}{\rho \vdash^S \text{Match } M \text{ with } \vec{p} \Rightarrow \vec{b} \text{ end} \Rightarrow \sqcap\vec{\sigma}}$$

Example

```
Definiton pred n (H:n<>0) :=
  match n with
    0 ⇒ match H _ with end
  | S k ⇒ k
  end.
Fixpoint F x :=
  if eq_nat_dec x 0 then 0 else F (pred x)
```

Fixpoints as argument of F

- A fix returns a strict subterm if its body does
- Size information of recursive argument is propagated

$$\frac{\rho \vdash^S u_n \Rightarrow \sigma \quad \rho \cup \{(G, \tau^-), (x_n, \sigma)\} \vdash^S M \Rightarrow \tau^-}{\rho \vdash^S (\mathrm{Fix}\ G_n : T := \lambda \vec{x} : \vec{t}.\, M)\ \vec{u} \Rightarrow \tau^-}$$

Example

```
Fixpoint F x y :=
  if ‘‘x ≤ y’’ then x else F (x-S(y)) y
```

Nested fixpoints

$$\frac{\rho \vdash^{S} u_n \Rightarrow \sigma \quad M \in \mathrm{Check}^{F,k}_{\rho\{(x_k,\sigma)\}} \quad T \in \mathrm{Check}^{F,k}_{\rho} \quad \vec{u} \in \mathrm{Check}^{F,k}_{\rho}}{(\mathrm{Fix}\ G_n : T := M)\ \vec{u} \in \mathrm{Check}^{F,k}_{\rho}}$$

Example (size of a tree)

```
Fixpoint size (t:tree) :=
  match t with
    Node l ⇒ fold_right (fun t' n ⇒ n+size t') 1 l
  end.
```

Overview

Nested vs. mutual inductive types

Example (Guard violated)

```
Fixpoint size (t:tree) :=
  match t with
     Node l ⇒ S(size_forest l)
  end
with size_forest (l:list tree) :=
  match l with
    nil ⇒ 0
  | t::l' ⇒ size t + size l'
  end.
```

Mutual inductive types can be used in the context of both mutual fixpoints and nested fixpoints.
Nested inductive types cannot be used in the context of mutual fixpoints.

Overview

- Many extensions already,
- Many are still missing (syntactic criterion)

So why this talk ?

- An opportunity to stop and think
- A highly critical (implementation) bug found: apply the patch!
- Syntactic criterions are dead: Gimenez, Blanqui, Barthe (and...) moved to type-based guard verification (size annotation)