On defining McCarthy f91 in CIC

J.-F. Monin

VERIMAG
Université Joseph Fourier, Grenoble, France

TYPES 2008, Torino

Outline

Problem

Defining a general provably terminating recursive function
and proving properties about it
without :

- proving termination at the same time
- thinking too much
- exotic features

Coq CIC

CIC = Calculus of Inductive Constructions

Defining (typed) functions, i.e. proofs (for formulæ)
Functions can be defined over types
$\Rightarrow$ discrimination between constructors

Basic constructs:

- typed λ-calculus
- inductive types $\Rightarrow$ higher-order term algebra
- induction $\subset$ pattern matching + fixpoint

Reduction relation:

- proof steps for free, inversions for free
- applications to computational reflection

Strong normalization

The reduction relation is nœtherian

Functions defined over terms in an inductive type
must have a clearly strictly decreasing argument

"clearly" = syntactically (after compile-time computations)
"decreasing" : for the subterm relation

Non interference

Two basic sorts :

- Set, for computationally meaningful data and functions
  e.g. *nat*: Set
- Prop, for logical comments
  e.g. True : Prop; *positive*: $nat \longrightarrow$ Prop

Rich types, e.g. $\{n : nat \mid n < 10\}$

Non-interference property: No information flow from Prop to Set
e.g. case analysis from a logical disjunction to a *nat* is not allowed.

Terminology: informative / non-informative objects

NB. Pattern matching over a proof having at most one constructor is still allowed (example $and\_intro : P \longrightarrow Q \longrightarrow P \wedge Q$)
provided the constructor has no informative argument
(counter-example: $ex\_intro : \forall x : A, P\,x \longrightarrow \exists x, P\,x$)

Program extraction

In $f : \{x : nat \mid P\,x\} \to \{y : nat \mid Q\,y\}$,

- y does not depend on the proof of $P\,x$
- program extraction = erasure of Prop parts
  $\mathcal{E}(f) : nat \to nat$

More: computational irrelevance

In $f : \{x : nat \mid P\,x\} \to \{y : nat \mid Q\,y\}$,

- we are not interested in a normal proof of $Q\,y$
- hence even within the original typing,
  no point to compute Prop parts

Markov Rule

Original rule

$$\mathit{decidable}\ P \to \neg\neg(\exists x : \mathit{nat}, P\,x) \to \exists y : \mathit{nat}, P\,y$$

More relevant here

$$\mathit{decidable}\ P \to \neg\neg\{x : \mathit{nat} \mid P\,x\} \to \{y : \mathit{nat} \mid P\,y\}$$

Obviously

$$(\exists x : A, P\,x) \to \neg\neg\{y : A \mid P\,y\}$$

$\Rightarrow$ Interesting part

$$\mathit{decidable}\ P \to (\exists x : \mathit{nat} \mid P\,x) \to \{y : \mathit{nat} \mid P\,y\}$$

- logical reading
- program extraction reading

Inductively defined domains

Defining a general provably terminating recursive function $f(x)$ without thinking too much

[A. Bove & V. Capretta, 2002] :

1. Derive an ad-hoc inductive dependent type $D\,x$ from the desired function
2. Add an argument t ot type $D\,x$ to f
3. Define f by structural recursion over t
4. Separately, define an auxiliary function $f_t : \forall x, D\,x$

In many cases, steps 3 and 4 and independent

Example: mergesort

```
merge_sort x [] = [x]
merge_sort x (y :: l) = merge (merge_sort x (splitleft l))
                              (merge_sort y (splitright l))

Inductive dom : list nat → Set  :=
  | Dnil : dom nil
  | Dcons : ∀ y l,
      dom (splitleft l) → dom (splitright l) → dom (y :: l).

Fixpoint merge_sort (x : nat) l (d : dom l) {struct d} : list nat :=
  match d with
  | Dnil ⇒ x :: nil
  | Dcons y l d1 d2 ⇒
      merge (merge_sort x (splitleft l) d1)
            (merge_sort y (splitright l) d2)
  end.
```

Two (known) problems

1) Does not extract the desired function

2) Does not scale up to nested recursion
(without induction-recursion, anyway unavailable in Coq)

Tackling pb 1: Prop-bounded fixed points

The syntactically decreasing argument can have sort Prop
Goes beyond structural recursion on an informative argument
Called termination certificate

However :
Reduction controlled by termination certificate
$\Rightarrow$ need to compute over termination certificates

Special case: well-founded recursion

Variable A : Set.
Variable R : $A \to A \to$ Prop.

Inductive Acc : $A \to$ Prop:=
Acc_intro : $\forall\ x{:}A,\ (\forall\ y{:}A,\ R\ y\ x \to Acc\ y) \to Acc\ x$.

A relation is well-founded if every element is accessible

Definition $well\_founded$:= $\forall\ a{:}A,\ Acc\ a$.

Special case: cheating

```
Inductive cool : Prop:=
      cool_intro : cool → cool

Fixpoint syracuse (n:nat) (t:cool) {struct t} : nat :=
   let t' := match t with cool_intro x ⇒ x end in
   if even n then syracuse (half n) t' else syracuse (3 × n + 1) t' .

Extraction syracuse.

Fixpoint loop (n:nat) (t:cool) {struct t} : nat  :=
   loop n (match t with cool_intro x ⇒ x end).

Extraction loop.
```

⇒ Program extraction cannot be the only motivation

Myth about fixpoints in CIC

Termination certificates have only one constructor

Counter-example: proof of Markov rule in Coq

[Britany folklore, last century]

Realizing Markov rule

```
Inductive before_wit : nat→ Prop  :=
  | stop : ∀ n, P n → before_wit n
  | next : ∀ n, before_wit (S n) → before_wit n.

Definition inv_before_wit :
  ∀ n, before_wit n → ~(P n) → before_wit (S n).
intros n before_n; case before_n; clear before_n n.
  intros n P_n not_P_n; case (not_P_n P_n).
  intros n before_Sn _; exact before_Sn.
Defined.

Fixpoint linear_search n (t : before_wit n) {struct t} : {x | P x} :=
  match P_dec n with
    | left yes ⇒ exist (fun x ⇒ P x) n yes
    | right no ⇒ linear_search (S n) (inv_before_wit n t no)
  end.
```

Termination certificate for Markov rule

```
Definition O_wit : ∀ n, before_wit n → before_wit O.
induction n as [| n IHn].
  exact (fun x ⇒ x).
  exact (fun bSn ⇒ IHn (next n bSn)).
Defined.

Definition ex_wit : (∃ n, P n) → before_wit O.
intros (x, Px). exact (O_wit x (stop x Px)).
Defined.

Definition markov (e : ∃ x, P x) : {x:nat | P x} :=
    linear_search O (ex_wit e).
```

Tricks

1) (*inv_before_wit n t no*) reduces to a subterm of *t* in all cases

```
Definition inv_before_wit :
  ∀ n, before_wit n → ~(P n) → before_wit (S n).
intros n before_n; case before_n; clear before_n n.
  intros n P_n not_P_n; case (not_P_n P_n).
  intros n before_Sn _; exact before_Sn.
Defined.
```

2) *t* abstracted before case analysis

```
Fixpoint linear_search n (t : before_wit n) {struct t} : {x | P x} :=
  match P_dec n with
    | left yes ⇒ exist (fun x ⇒ P x) n yes
    | right no ⇒ linear_search (S n) (inv_before_wit n t no)
  end.
```

Generalisation

before_wit is just an inductive definition of the domain of *linear_search*

But in Prop instead of Set

$\Rightarrow$ Use Bove-Capretta technique + tricks 1 and 2 [Coq folklore]

Back to mergesort (1)

```
Inductive dom : list nat → Prop  :=
  | Dnil : dom nil
  | Dcons : ∀ y l,
       dom (splitleft l) → dom (splitright l) → dom (y :: l).

Definition dom_left :
  ∀ l, dom l → non_empty l → dom (splitleft (tail l)).
destruct 1 as [| y l dl dr]; simpl; intro ne_l.
  case ne_l.
  exact dl.
Defined.

Definition dom_right :
  ∀ l, dom l → non_empty l → dom (splitright (tail l)).
```

Back to mergesort (2)

```
Fixpoint merge_sort
  (x : nat) (l : list nat) (t : dom l) {struct t} : list nat :=
  match l as l0
  return ((non_empty l0 → non_empty l) → list nat) with
  | nil ⇒ fun _ ⇒ x :: l
  | y :: _ ⇒
      fun f ⇒
      merge (merge_sort x (splitleft (tail l)) (dom_left l t (f l)))
            (merge_sort y (splitright (tail l)) (dom_right l t (f l)))
  end (fun ne ⇒ ne).
```

McCarthy f91

$$\text{let rec } f91\ n = \text{if } 100 < \text{n then } n \text{ - } 10 \text{ else } f91\ (f91\ (n + 11))$$

Let us try

$$\begin{array}{l}\text{Inductive } dom91 : nat \to nat \to \text{Prop} := \\ \quad \mid D10 : \forall\ n,\ 100 < n \to dom91\ n \\ \quad \mid D11 : \forall\ n,\ n \leq 100 \to \\ \qquad\qquad dom91\ (11 + n) \to dom91\ (f91\ (11 + n)) \to dom91\ n.\end{array}$$

Does not work because $f91$ is still undefined.

Use the whole specification?

$$\begin{aligned}&\text{Inductive } \mathit{spec91} : \mathit{nat} \to \mathit{nat} \to \text{Prop} := \\ &\quad | \ \mathit{F10} : \forall\ n,\ 100 < n \to \mathit{spec91}\ n\ (n - 10) \\ &\quad | \ \mathit{F11} : \forall\ n,\ n \leq 100 \to \\ &\qquad\qquad \forall\ x\ y,\ \mathit{spec91}\ (11 + n)\ x \to \mathit{spec91}\ x\ y \to \mathit{spec91}\ n\ y.\end{aligned}$$

Failure again

- We only have the argument in our pocket, not the result.
- $\exists y, \mathit{spec91}\ x\ y$ yields unrelated numbers that should be equal at some point

Squelettic specification

Idea: simple inductive types are simpler to handle
$\Rightarrow$ remove arguments of *spec91*

```
Inductive squel91 : Prop :=
  | S10 : squel91
  | S11 : squel91 → squel91 → squel91.
```

In order to justify subterm decreasing, need to relate a *squel91* to arguments

```
Inductive consist_squel : squel91 → nat → nat → Prop :=
   | CS10 : ∀ n, 100 < n → ∀ s, consist_squel s n (n-10)
   | CS11 : ∀ n, n ≤ 100 → ∀ s x t y,
                consist_squel s (11+n) x → consist_squel t x y →
                consist_squel (S11 s t) n y.
```

Coq definition of f91

```
Definition f91sq (n: nat) (s: squel91) :
           (∃ y, consist_squel s n y) → {y | consist_squel s n y}.
fix 2.
intros n s yc.
case (le_lt_dec n 100).
   intros ln100.
   case (f91sq (11 + n) (left_squel91 s n yc ln100)
         (pc_left n s yc ln100)).
etc.
```

Projections

Definition *left_squel91* :
$\forall\ s\ x,\ (\exists\ y,\ consist\_squel\ s\ x\ y) \rightarrow x \leq 100 \rightarrow squel91.$

Definition *right_squel91* :
$\forall\ s\ x,\ (\exists\ y,\ consist\_squel\ s\ x\ y) \rightarrow x \leq 100 \rightarrow squel91.$

Consistency

Definition *pc_left* :
$\forall\ n\ s\ (yc : \exists\ y : nat,\ consist\_squel\ s\ n\ y)\ (ln100 : n \leq 100),$
$\exists\ x : nat,\ consist\_squel\ (left\_squel91\ s\ n\ yc\ ln100)\ (11 + n)\ x.$

Definition *pc_right* :
$\forall\ n\ s\ (yc : \exists\ y : nat,\ consist\_squel\ s\ n\ y)\ (ln100 : n \leq 100),$
$\forall\ x\ (cx : consist\_squel\ (left\_squel91\ s\ n\ yc\ ln100)\ (11 + n)\ x),$
$\exists\ y : nat,\ consist\_squel\ (right\_squel91\ s\ n\ yc\ ln100)\ x\ y.$

Remarks and conclusion

1. Rich typing of f91: correct by construction (no choice)
2. Need for a functional specification (in *pc_right*)
3. Need for trick 1 + simple termination certificate ⇒
   discriminate constructors of *squel91*

Inductive *discr_Prop* : Prop := *trueP* : *discr_Prop* | *falseP* : *discr_Prop*.

Axiom *discr* : *trueP* ≠ *falseP*.

Definition *isS10 s* :=
  match *s* with *S10* ⇒ *trueP* | *S11* _ _ ⇒ *falseP* end.

Lemma *discr_squel91* : ∀ *s t*, *S11 s t* ≠ *S10*.

Proof relevance useful

Set < TC < Prop