

On defining McCarthy f91 in CIC

J.-F. Monin

VERIMAG

Université Joseph Fourier, Grenoble, France

TYPES 2008 (Torino), DCS 2008 (Col de Portes)

Outline

Problem and Background

- Problem

- Coq CIC

- Computational relevance

Fixpoints

- Inductively defined domains

- Example: mergesort

- Prop-bounded fixed points

- Myth about fixpoints in CIC

- Key points

- Back to mergesort

Nested recursion

- McCarthy f91

- Conclusion

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

⇒ discrimination between constructors

Basic constructs:

- ▶ typed λ -calculus
- ▶ inductive types ⇒ 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 noetherian

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. $True : Prop$; $positive : nat \rightarrow Prop$

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 $and_intro : P \rightarrow Q \rightarrow P \wedge Q$)
provided the constructor has no informative argument
(counter-example: $ex_intro : \forall x : A, P x \rightarrow \exists x, P x$)

Program extraction

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

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

More: computational irrelevance

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

- ▶ 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

$$\text{decidable } P \rightarrow \neg\neg(\exists x : \text{nat}, P x) \rightarrow \exists y : \text{nat}, P y$$

More relevant here

$$\text{decidable } P \rightarrow \neg\neg\{x : \text{nat} \mid P x\} \rightarrow \{y : \text{nat} \mid P y\}$$

Obviously

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

⇒ Interesting part

$$\text{decidable } P \rightarrow (\exists x : \text{nat} \mid P x) \rightarrow \{y : \text{nat} \mid P y\}$$

- ▶ 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 Dx from the desired function
2. Add an argument t of type Dx to f
3. Define f by structural recursion over t
4. Separately, define an auxiliary function $f_t : \forall x, Dx$

In many cases, steps 3 and 4 are independent

Example: mergesort

$merge_sort\ x\ [] = [x]$

$merge_sort\ x\ (y :: l) = merge\ (merge_sort\ x\ (splitleft\ l))$
 $(merge_sort\ y\ (splitright\ l))$

Inductive $dom : list\ nat \rightarrow Set :=$

| $Dnil : dom\ nil$

| $Dcons : \forall\ y\ l,$

$dom\ (splitleft\ l) \rightarrow dom\ (splitright\ l) \rightarrow dom\ (y :: l).$

Fixpoint $merge_sort\ (x : nat)\ l\ (d : dom\ l)\ \{struct\ d\} : list\ nat :=$
match d **with**

| $Dnil \Rightarrow x :: nil$

| $Dcons\ y\ l\ d1\ d2 \Rightarrow$

$merge\ (merge_sort\ x\ (splitleft\ l)\ d1)$
 $(merge_sort\ y\ (splitright\ l)\ d2)$

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

⇒ **need to compute over termination certificates**

Special case: well-founded recursion

Variable A : Set .

Variable R : $A \rightarrow A \rightarrow \text{Prop}$.

Inductive Acc : $A \rightarrow \text{Prop} :=$

Acc_intro : $\forall x:A, (\forall y:A, R\ y\ x \rightarrow Acc\ y) \rightarrow Acc\ x$.

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* : $\text{nat} \rightarrow \text{Prop} :=$

| *stop* : $\forall n, P\ n \rightarrow \text{before_wit}\ n$

| *next* : $\forall n, \text{before_wit}\ (S\ n) \rightarrow \text{before_wit}\ n.$

Definition *inv_before_wit* :

$\forall n, \text{before_wit}\ n \rightarrow \sim(P\ n) \rightarrow \text{before_wit}\ (S\ n).$

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* \Rightarrow **exist** (**fun** *x* \Rightarrow *P* *x*) *n yes*

| *right no* \Rightarrow *linear_search* (*S* *n*) (*inv_before_wit* *n t no*)

end.

Termination certificate for Markov rule

Definition $O_wit : \forall n, \text{before_wit } n \rightarrow \text{before_wit } O$.

induction n as $[[\mid n \mid IHn]]$.

exact (fun $x \Rightarrow x$).

exact (fun $bSn \Rightarrow IHn (\text{next } n \ bSn)$).

Defined.

Definition $ex_wit : (\exists n, P n) \rightarrow \text{before_wit } O$.

intros (x, Px). **exact** ($O_wit x (\text{stop } x \ Px)$).

Defined.

Definition $markov (e : \exists x, P x) : \{x:nat \mid P x\} :=$
 $\text{linear_search } O (ex_wit e)$.

Tricks

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

Definition inv_before_wit :

$\forall n, before_wit\ n \rightarrow \sim(P\ n) \rightarrow before_wit\ (S\ n).$

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 $linear_search\ n\ (t : before_wit\ n)\ \{struct\ t\} : \{x \mid P\ x\} :=$

match $P_dec\ n$ **with**

| $left\ yes \Rightarrow exist\ (fun\ x \Rightarrow P\ x)\ n\ yes$

| $right\ no \Rightarrow linear_search\ (S\ n)\ (inv_before_wit\ n\ t\ no)$

end.

Generalisation

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

But in **Prop** instead of **Set**

⇒ 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

```
let rec f91 n = if 100 < n then n - 10 else f91 (f91 (n + 11))
```

Let us try

Inductive $dom91 : nat \rightarrow nat \rightarrow Prop :=$

| D10 : $\forall n, 100 < n \rightarrow dom91 n$

| D11 : $\forall n, n \leq 100 \rightarrow$

$dom91 (11 + n) \rightarrow dom91 (f91 (11 + n)) \rightarrow dom91 n.$

Does not work because $f91$ is still undefined.

Use the whole specification?

Inductive *spec91* : nat → nat → Prop :=
| F10 : ∀ n, 100 < n → *spec91* n (n - 10)
| F11 : ∀ n, n ≤ 100 →
 ∀ x y, *spec91* (11 + n) x → *spec91* x y → *spec91* n y.

Failure again

- ▶ We only have the argument in our pocket, not the result.
- ▶ $\exists y, \text{spec91 } x \ y$ 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* \rightarrow *squel91* \rightarrow *squel91*.

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

Inductive *consist_squel* : *squel91* \rightarrow nat \rightarrow nat \rightarrow Prop :=
| *CS10* : $\forall n, 100 < n \rightarrow \forall s, \text{consist_squel } s \ n \ (n-10)$
| *CS11* : $\forall n, n \leq 100 \rightarrow \forall s \ x \ t \ y,$
 consist_squel *s* (11+n) *x* \rightarrow *consist_squel* *t* *x* *y* \rightarrow
 consist_squel (*S11* *s* *t*) *n* *y*.

Coq definition of f91

Definition *f91sq* (*n*: *nat*) (*s*: *squel91*) :
 $(\exists y, \text{consist_squel } s \ n \ y) \rightarrow \{y \mid \text{consist_squel } s \ n \ y\}$.

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* :

$$\forall s x, (\exists y, \text{consist_squel } s x y) \rightarrow x \leq 100 \rightarrow \text{squel91}.$$

Definition *right_squel91* :

$$\forall s x, (\exists y, \text{consist_squel } s x y) \rightarrow x \leq 100 \rightarrow \text{squel91}.$$

Consistency

Definition *pc_left* :

$$\forall n s (yc : \exists y : \text{nat}, \text{consist_squel } s n y) (ln100 : n \leq 100), \\ \exists x : \text{nat}, \text{consist_squel } (\text{left_squel91 } s n yc ln100) (11 + n) x.$$

Definition *pc_right* :

$$\forall n s (yc : \exists y : \text{nat}, \text{consist_squel } s n y) (ln100 : n \leq 100), \\ \forall x (cx : \text{consist_squel } (\text{left_squel91 } s n yc ln100) (11 + n) x), \\ \exists y : \text{nat}, \text{consist_squel } (\text{right_squel91 } s n yc ln100) x y.$$

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 \Rightarrow
discriminate constructors of `squel91`

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

Axiom `discr` : `trueP` \neq `falseP`.

Definition `isS10` `s` :=

`match s with S10 \Rightarrow trueP` | `S11 _ _ \Rightarrow falseP end`.

Lemma `discr_squel91` : $\forall s t, S11 s t \neq S10$.

Proof **relevance** useful

Set < **TC** < **Prop**