

Inductive Types

Part 3-1 :

Generalizations and definitions

Benjamin Werner

INRIA-Rocquencourt

Proofs-as-Programs Summer School
Eugene, Oregon, July 1st 2002

General Form of Inductive Definitions

$I : (\underline{x}_i : A_i) \rightarrow$ the inductive type/predicate
 $C_j : (\underline{y}_j : T_j) \rightarrow I(\underline{a}_i)$ type of each constructor

positivity :

- either $I \notin T_j^k$, or
- $T_j^k = (z_l : U_l) \rightarrow I(b)$ with $I \notin U_l$

Examples :

$\text{cons} : A \rightarrow \text{list} \rightarrow \text{list}$, $\text{left} : A \rightarrow (or\ A\ B)$

Beyond data-types

Inductive ord : Set :=

0o : ord

| So : ord -> ord

| Lim : (nat->ord) -> ord.

(Lim f) is a **canonical** "ordinal".

For any $n:\text{nat}$, $(f\ n)$ is structurally smaller than $(\text{Lim } f)$.

These are infinitely branching trees,

they cannot be (finitely) printed.

Even further

Inductive $\text{Ens} : \text{Type} :=$
 $\text{sup} : (A : \text{Type}) (A \rightarrow \text{Ens}) \rightarrow \text{Ens}.$

Here we even branch w.r.t. an arbitrary
type !

Not very intuitive. . .

Very powerful : this type encodes sets of
Zermelo set theory

Wait for Alexandre's lectures

Restrictions w.r.t. sorts

Consider :

Inductive capture : Set :=
c_i : Set -> capture.

This definition is correct

But if we allow projection :
proj : capture -> Set with

(proj (c_i A)) > A

then we can "encode" Set: Set

The reason is that c_1 quantifies over Sets.
 \Rightarrow elimination towards Set is forbidden for
such types.

The real typing of Fms :

Inductive $\text{Fms} : \text{Type}(\mathbb{N}+1) :=$

$\text{sup} : (A : \text{Type}(\mathbb{N})) (A \rightarrow \text{Fms}) \rightarrow \text{Fms}.$

The sorts Set and Prop

They are “twins”

The “Prop part” can be erased for a
realisability interpretation.

For example, Harrop formulas should be of
type Prop.

We should not use Prop terms to compute
Set terms.

```

Program extraction 1 : dividing by two
Fixpoint D [n:nat] : nat :=
  Cases n of 0 => 0
  | (S p) => (S(D p))
end.
Inductive even : nat -> Prop :=
  e0 : (even 0)
  | es : (n:nat) (even n) -> (even (S(S n))).

```

Two existentials

$(\text{EX } p:\text{nat} \mid n=(D \ p)) : \text{Prop}$
 $\{ p:\text{nat} \mid n = (D \ p) \} : \text{Set}$

$(n:\text{nat})(\text{Even } n) \rightarrow (\text{EX } p:\text{nat} \mid n=(D \ p)) : \text{Prop}$
 $(n:\text{nat})(\text{Even } n) \rightarrow \{ p:\text{nat} \mid n = (D \ p) \} : \text{Set}$

First one can be proved by induction over
the proof of (even n), the second one

cannot.

```
let rec ins l a x =  
  match l with
```

```
  nil -> cons (a, nil)
```

```
  | cons (n, l0) ->
```

```
    (match compare_dec.le_dec a n with
```

```
      left -> cons (a, (cons (n, l0)))
```

```
      | right -> cons (n, (ins l0 a prop)))
```