

# Computation by Prophecy

Venanzio Capretta

University of Ottawa

with

Ana Bove

Chalmers University of Technology

TYPES 2006  
Nottingham, 18–21 April 2006

How do we define general recursive functions in Type Theory? Only structurally recursive functions are directly definable. Three different solutions:

- Bove/Capretta 2001:  
Inductive Domain Predicate
- Capretta 2005:  
Coinductive Partial Codomain
- Prophecy Method:  
Coinductive Abstract Output

Informal definition of the quicksort algorithm:

```
qs: [N] → [N]
qs [] = []
qs (x :: l) = (qs l≤x) ++ x :: (qs l>x)
```

where

$$l_{\leq x} = [y \leftarrow l \mid y \leq x]$$
$$l_{> x} = [y \leftarrow l \mid y > x]$$

Example:

```
qs [7, 9, 1, 8, 5, 2]
= (qs [1, 5, 2]) ++ 7 :: (qs [9, 8])
= [1, 2, 5] ++ 7 :: [8, 9]
= [1, 2, 5, 7, 8, 9]
```

Not acceptable in Type Theory:

$l_{\leq x}$  and  $l_{> x}$  not subterms of  $l$ .

## First Method (Bove)

Inductive Domain Predicate:  $D_{\text{qs}}: [\mathbb{N}] \rightarrow \text{Prop}$

$$\frac{}{d_{\text{nil}}: D_{\text{qs}} []} \quad \frac{x: \mathbb{N} \quad l: [\mathbb{N}] \quad h_1: D_{\text{qs}} l_{\leq x} \quad h_2: D_{\text{qs}} l_{> x}}{d_{\text{cons}} x l h_1 h_2: D_{\text{qs}} (x :: l)}$$

$$\begin{aligned} \text{qs}: (l: [\mathbb{N}]) (D_{\text{qs}} l) &\rightarrow [\mathbb{N}] \\ \text{qs } [] \ d_{\text{nil}} &= [] \\ \text{qs } (x :: l) \ (d_{\text{cons}} x l h_1 h_2) &= \\ &(\text{qs } l_{\leq x} h_1) ++ x :: (\text{qs } l_{> x} h_2) \end{aligned}$$

In this case we can prove that  $D_{\text{qs}}$  is always satisfied:

$$p: \forall l. D_{\text{qs}} l$$

$$\begin{aligned} \text{QS}: [\mathbb{N}] &\rightarrow [\mathbb{N}] \\ \text{QS } l &= \text{qs } l \ (p l) \end{aligned}$$

## **Advantages:**

- Close to Informal Definition
- Recursive Equations (erase proofs)
- Easy Proofs

## **Disadvantages:**

- No Partial Application
- We must always give a proof of the domain predicate  
(Proof = Trace of Computation)
- No Type of Partial Recursive Functions

## Second Method (Capretta 2005)

Coinductive type of partial elements:

$$\text{CoInductive} \frac{B : \text{Type}}{B^\nu : \text{Type}} \quad \frac{b : B}{\ulcorner b \urcorner : B^\nu} \quad \frac{x : B^\nu}{\triangleright x : B^\nu}$$

Examples of elements in  $\mathbb{N}^\nu$ :

$\ulcorner 5 \urcorner$   
 $\triangleright \triangleright \triangleright \ulcorner 3 \urcorner$   
 $\triangleright \triangleright \triangleright \triangleright \triangleright \dots$     infinite

Partial functions:

$f : A \multimap B$  means  $f : A \rightarrow B^\nu$

All general recursive functions can be formalized in this type.

Partial recursive functions are the arrows of the Kleisli category of the strong monad  $\multimap^\nu$ .

## **Advantages:**

- No Proofs Needed
- Partial Application
- Type of Partial Recursive Functions

## **Disadvantages:**

- Difficult to Program
- No Recursive Equations
- Difficult Proofs

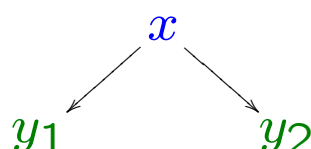
## Prophecy Method

Abstract coinductive representation of outputs:

$\text{CoInductive } [\mathbb{N}]^{\text{qs}} : \text{Type}$

$$\frac{}{\text{qsln}_{\text{nil}} : [\mathbb{N}]^{\text{qs}}} \quad \frac{x : \mathbb{N} \quad y_1, y_2 : [\mathbb{N}]^{\text{qs}}}{\text{qsln}_{\text{cons}} x y_1 y_2 : [\mathbb{N}]^{\text{qs}}}$$

Elements of  $[\mathbb{N}]^{\text{qs}}$  are binary trees  
(possibly non-wellfounded)

$$\text{qsln}_{\text{cons}} x y_1 y_2 =$$


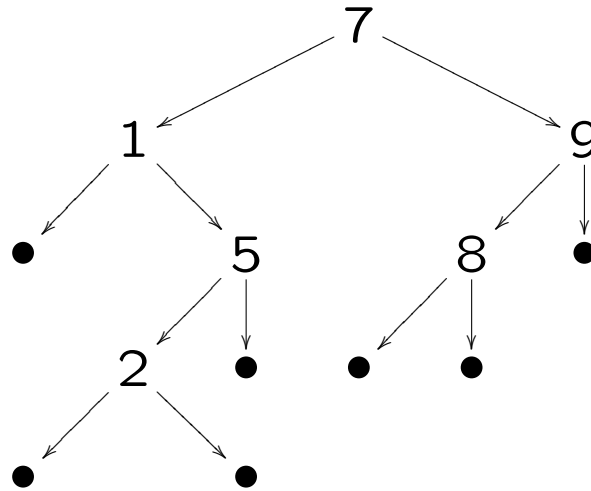
```
graph TD; x((x)) --> y1((y1)); x --> y2((y2));
```

Abstract definition of quicksort:

$$\begin{aligned} \text{qs}^* &: [\mathbb{N}] \rightarrow [\mathbb{N}]^{\text{qs}} \\ \text{qs}^* [] &= \text{qsln}_{\text{nil}} \\ \text{qs}^* (x :: l) &= \text{qsln}_{\text{cons}} x (\text{qs}^* l_{\leq x}) (\text{qs}^* l_{> x}) \end{aligned}$$

Recursive calls guarded by constructor  $\text{qsln}_{\text{cons}}$

Example  $qs^*[7, 9, 1, 8, 5, 2]$  gives the tree:



We must evaluate the tree to get a (partial) list, an element of  $[\mathbb{N}]^\nu$ :

$$\text{evaluate}_{qs} : [\mathbb{N}]^{qs} \rightarrow [\mathbb{N}]^\nu$$

- Turn an element of  $[\mathbb{N}]^{qs}$  into a partial well-founded tree.
- Evaluate well-founded trees by structural recursion.

Well-founded trees are characterized by an inductive predicate:

Inductive  $\frac{y : [\mathbb{N}]^{\text{qs}}}{\text{Finite}_{\text{qsln}} y : \text{Prop}}$

$\frac{}{\text{finite}_{\text{nil}} : \text{Finite}_{\text{qsln}} \text{qsln}_{\text{nil}}}$

$\frac{h_1 : \text{Finite}_{\text{qsln}} y_1 \quad h_2 : \text{Finite}_{\text{qsln}} y_2}{\text{finite}_{\text{cons}} x y_1 y_2 h_1 h_2 : \text{Finite}_{\text{qsln}} (\text{qsln}_{\text{cons}} x y_1 y_2)}$

Evaluation of finite trees:

$\text{eval}_{\text{Finite}} : (y : [\mathbb{N}]^{\text{qs}}) (\text{Finite}_{\text{qsln}} y) \rightarrow [\mathbb{N}]$   
 $\text{eval}_{\text{Finite}} \text{qsln}_{\text{nil}} \text{finite}_{\text{nil}} = []$   
 $\text{eval}_{\text{Finite}} (\text{qsln}_{\text{cons}} x y_1 y_2) (\text{finite}_{\text{cons}} x y_1 y_2 h_1 h_2)$   
 $\quad = (\text{eval}_{\text{Finite}} y_1 h_1) ++ x :: (\text{eval}_{\text{Finite}} y_2 h_2)$

Trees are potentially infinite (if the computation doesn't terminate).

So  $\text{eval}_{\text{Finite}}$  is not always applicable.

Idea: scan the tree at progressively higher depths, letting the clock tick  $\triangleright$  at each step.

$$\text{scan}_{\text{depth}} : (y : [\mathbb{N}]^{\text{qs}}) \mathbb{N} \rightarrow \text{Maybe}(\text{Finite}_{\text{qsIn}} y)$$

Apply  $\text{scan}_{\text{depth}}$  at progressively increasing depths:

If we get `Some`, use  $\text{eval}_{\text{Finite}}$ ;

If we get `None`, tick  $\triangleright$  and try at higher depth.

We obtain:

$$\text{evaluate}_{\text{qs}} : [\mathbb{N}]^{\text{qs}} \rightarrow [\mathbb{N}]^{\nu}$$

$$\text{qs} : [\mathbb{N}] \rightarrow [\mathbb{N}]^{\nu}$$

$$\text{qs } l = \text{evaluate}_{\text{qs}} (\text{qs}^* l)$$

Recursive Equations:

$$\text{qs } [] \rightsquigarrow []$$

$$\frac{\text{qs } l_{\leq x} \rightsquigarrow r_1 \quad \text{qs } l_{> x} \rightsquigarrow r_2}{\text{qs}(x :: l) \rightsquigarrow r_1 \mathbin{++} x :: r_2}$$

## **Advantages:**

- Partial Application
- Type of Partial Recursive Functions
- Recursive Equations

## **Disadvantages:**

- Inefficient Evaluation
- Proof of Equations Very Hard

Example with a partial function:

$$\begin{aligned} f &: \mathbb{N} \rightarrow \mathbb{N} \\ f\ 0 &= 0 \\ f\ (S\ n) &= f\ (S\ n) + f\ n \end{aligned}$$

CoInductive  $C_f$ : Type

$$\frac{}{c_0 : C_f} \quad \frac{y_1, y_2 : C_f}{c_S\ y_1\ y_2 : C_f}$$

$$\begin{aligned} f^* &: \mathbb{N} \rightarrow C_f \\ f^*\ 0 &= c_0 \\ f^*\ (S\ n) &= c_S\ (f^*\ (S\ n))\ (f^*\ n) \end{aligned}$$

$$f^*\ 1 =$$

