

# Coinduction in Agda using Copatterns and Sized Types

Andreas Abel<sup>1</sup>

with James Chapman, Brigitte Pientka, Anton Setzer, David Thibodeau

<sup>1</sup>Department of Computer Science and Engineering  
Gothenburg University, Sweden

Workshop on Advances in Programming Languages and Systems  
Goethe Universität, Frankfurt a.M.  
15 December 2015

# Copatterns

- Copatterns: invented to integrate sized coinductive types with pattern matching.
- Inspired by coalgebraic approach to coinduction (Anton Setzer).
- Solved the subject reduction problem of dependent matching on codata.
- Operational semantics is WYSIWYG.
- Implemented in Agda since 2.4.0.

# Coalgebras

- Copatterns = pattern matching for coalgebras.

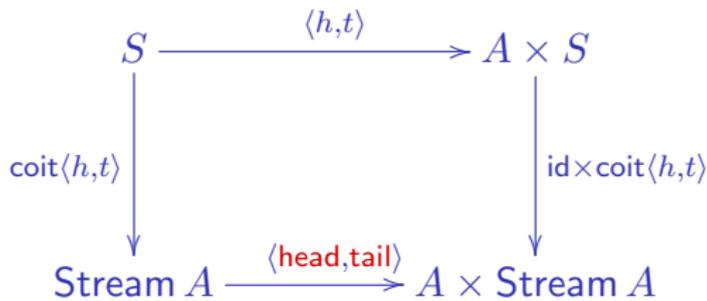
$$\begin{array}{ccc}
 S & \xrightarrow{f} & F(S) \\
 \text{coit } f \downarrow & & \downarrow F(\text{coit } f) \\
 \nu F & \xrightarrow{\text{force}} & F(\nu F)
 \end{array}$$

- Computation: Only unfold infinite object on demand.

$$\text{force}(\text{coit } f \ s) = F(\text{coit } f)(f \ s)$$

## Streams as Final Coalgebra

- Streams:  $F(S) = A \times S$



- Termination by induction on observation depth:

$$\text{head} (\text{coit} \langle h, t \rangle s) = h s$$

$$\text{tail} (\text{coit} \langle h, t \rangle s) = \text{coit} \langle h, t \rangle (t s)$$

# Copatterns: Syntax

- Elimination contexts (spines):

$$\begin{array}{l}
 E ::= \bullet \quad \text{head} \\
 \quad | Et \quad \text{application} \\
 \quad | \pi E \quad \text{projection}
 \end{array}$$

- Copatterns = pattern matching elimination contexts.

$$\begin{array}{l}
 Q ::= \bullet \quad \text{head} \\
 \quad | Qp \quad \text{application pattern} \\
 \quad | \pi Q \quad \text{projection pattern}
 \end{array}$$

- Rule  $Q[f] = t$  fires if copattern matches elimination context.

$$\frac{E = Q\sigma}{E[f] \longrightarrow t\sigma}$$

## Example: Fibonacci Stream

```
record Stream A : Set where
  coinductive
  field head : A
        tail : Stream A
open Stream; S = Stream
```

```
zipWith :  $\forall\{A B C\} \rightarrow (A \rightarrow B \rightarrow C) \rightarrow S A \rightarrow S B \rightarrow S C$ 
head (zipWith f s t) = f (head s) (head t)
tail (zipWith f s t) = zipWith f (tail s) (tail t)
```

```
fib : Stream  $\mathbb{N}$ 
(head fib) = 0
(head (tail fib)) = 1
(tail (tail fib)) = zipWith _+_ fib (tail fib)
```

## Sized Coinductive Types

- Track guardedness in the type system (Hughes Pareto Sabry 1996).
- Size = iteration stage towards greatest fixed point.
- Deflationary iteration ( $F$  need not be monotone).

$$\begin{array}{ccc}
 & \text{force} & \\
 & \curvearrowright & \\
 \nu^\alpha F & \cong & \bigcap_{\beta < \alpha} F(\nu^\beta F) \\
 & \curvearrowleft & \\
 & \text{delay} & 
 \end{array}$$

- $\nu^0 F = \top$  universe of terms / terminal object.
- Contravariant subtyping  $\nu^\alpha F \leq \nu^\beta F$  for  $\alpha \geq \beta$ .
- Stationary point  $\nu^{\infty+1} F = \nu^\infty F$  reached for some ordinal  $\infty$ .

## Sized Fibonacci Stream

record Stream  $i A$  : Set where

coinductive

field head :  $A$

tail :  $\forall\{j : \text{Size} < i\} \rightarrow \text{Stream } j A$

open Stream; S = Stream

zipWith :  $\forall\{i A B C\} \rightarrow (A \rightarrow B \rightarrow C) \rightarrow S\ i\ A \rightarrow S\ i\ B \rightarrow S\ i\ C$

head (zipWith  $\{i\}$   $f\ s\ t$ ) =  $f$  (head  $s$ ) (head  $t$ )

tail (zipWith  $\{i\}$   $f\ s\ t$ )  $\{j\}$  = zipWith  $\{j\}$   $f$  (tail  $s\ \{j\}$ ) (tail  $t\ \{j\}$ )

fib :  $\forall\{i\} \rightarrow \text{Stream } i\ \mathbb{N}$

tail (tail (fib  $\{i\}$ )  $\{j\}$ )  $\{k\}$  = zipWith  $\{k\}$   $\_ + \_$  (fib  $\{k\}$ )  
 (tail (fib  $\{j\}$ )  $\{k\}$ )

## Fibonacci Stream (Sizes Inferred)

record Stream  $i$   $A$  : Set where

coinductive

field head :  $A$

tail :  $\forall\{j : \text{Size} < i\} \rightarrow \text{Stream } j$   $A$

open Stream; S = Stream

zipWith :  $\forall\{i A B C\} \rightarrow (A \rightarrow B \rightarrow C) \rightarrow \text{S } i$   $A \rightarrow \text{S } i$   $B \rightarrow \text{S } i$   $C$

head (zipWith  $f$   $s$   $t$ ) =  $f$  (head  $s$ ) (head  $t$ )

tail (zipWith  $f$   $s$   $t$ ) = zipWith  $f$  (tail  $s$ ) (tail  $t$ )

fib :  $\forall\{i\} \rightarrow \text{Stream } i$   $\mathbb{N}$

(head fib) = 0

(head (tail fib)) = 1

(tail (tail fib)) = zipWith  $\_ + \_$  fib (tail fib)

## Example: De Bruijn Lambda Terms and Values

```

data Tm (n : ℕ) : Set where
  var   : (x   : Fin n)      → Tm n
  abs   : (t   : Tm (suc n)) → Tm n
  app   : (r s : Tm n)      → Tm n

```

mutual

```

record Val : Set where
  inductive
  constructor clos
  field      {n} : ℕ
             body : Tm (suc n)
             env  : Env n

```

Env = Vec Val

# Running Example: Naive Call-By-Value Interpreter

Evaluator (draft).

mutual

$$\llbracket \_ \rrbracket \_ : \forall \{n\} \rightarrow \text{Tm } n \rightarrow \text{Env } n \rightarrow \text{Val}$$

$$\llbracket \text{var } x \quad \rrbracket \rho = \text{lookup } x \rho$$

$$\llbracket \text{abs } t \quad \rrbracket \rho = \text{clos } t \rho$$

$$\llbracket \text{app } r \ s \rrbracket \rho = \text{apply } (\llbracket r \rrbracket \rho) (\llbracket s \rrbracket \rho)$$

$$\text{apply} : \text{Val} \rightarrow \text{Val} \rightarrow \text{Val}$$

$$\text{apply } (\text{clos } t \rho) \ v = \llbracket t \rrbracket (v :: \rho)$$

Of course, termination check fails!

# The Coinductive Delay Monad

```
CoInductive Delay (A : Type) : Type :=
| return (a : A)
| later (a? : Delay A).
```

mutual

```
data Delay (A : Set) : Set where
  return : (a : A)      → Delay A
  later  : (a' : Delay' A) → Delay A
```

```
record Delay' (A : Set) : Set where
  coinductive
  constructor delay
  field      force : Delay A
```

open Delay' public

## The Coinductive Delay Monad (Ctd.)

Nonterminating computation.

`forever` :  $\forall\{A\} \rightarrow \text{Delay}' A$

`force forever` = `later forever`

Monad instance.

`mutual`

`_ >>= _` :  $\forall\{A B\} \rightarrow \text{Delay } A \rightarrow (A \rightarrow \text{Delay } B) \rightarrow \text{Delay } B$

`return a >>= k` = `k a`

`later a' >>= k` = `later (a' >>= k)`

`_ >>= ' _` :  $\forall\{A B\} \rightarrow \text{Delay}' A \rightarrow (A \rightarrow \text{Delay}' B) \rightarrow \text{Delay}' B$

`force (a' >>= ' k)` = `force a' >>= k`

## Evaluation In The Delay Monad

Monadic evaluator.  $\llbracket \_ \rrbracket \_ : \forall \{n\} \rightarrow \text{Tm } n \rightarrow \text{Env } n \rightarrow \text{Delay Val}$

$\llbracket \text{var } x \rrbracket \rho = \text{return (lookup } x \rho)$

$\llbracket \text{abs } t \rrbracket \rho = \text{return (clos } t \rho)$

$\llbracket \text{app } r s \rrbracket \rho = \text{apply } (\llbracket r \rrbracket \rho) (\llbracket s \rrbracket \rho)$

$\text{apply} : \text{Delay Val} \rightarrow \text{Delay Val} \rightarrow \text{Delay Val}$

$\text{apply } u^? v^? = u^? \gg= \lambda u \rightarrow$

$v^? \gg= \lambda v \rightarrow$

$\text{later (apply}' u v)$

$\text{apply}' : \text{Val} \rightarrow \text{Val} \rightarrow \text{Delay}' \text{Val}$

$\text{force (apply}' (\text{clos } t \rho) v) = \llbracket t \rrbracket (v :: \rho)$

Not guarded by constructors!

## Sized Coinductive Delay Monad

mutual

```

data Delay {i : Size} (A : Set) : Set where
  return  : (a : A)                → Delay {i} A
  later   : (a' : Delay' {i} A)    → Delay {i} A

```

```

record Delay' {i : Size} (A : Set) : Set where
  coinductive
  constructor delay
  field      force : ∀{j : Size < i} → Delay {j} A

```

open Delay' public

- Size = depth = how often can we force?
- Not to be confused with “number of later’s”?

## Sized Coinductive Delay Monad (II)

Corecursion = induction on depth.

**forever** :  $\forall\{i\} A \rightarrow \text{Delay}' \{i\} A$

**force** (**forever**  $\{i\}$ )  $\{j\} = \text{later}$  (**forever**  $\{j\}$ )

Since  $j < i$ , the recursive call **forever**  $\{j\}$  is justified.

## Sized Coinductive Delay Monad (III)

Monadic bind preserves depth.

mutual

$$\_ \gg = \_ : \forall \{i A B\} \rightarrow \text{Delay } \{i\} A \rightarrow (A \rightarrow \text{Delay } \{i\} B) \rightarrow \text{Delay } \{i\} B$$

$$\text{return } a \gg = k = k a$$

$$\text{later } a' \gg = k = \text{later } (a' \gg = k)$$

$$\_ \gg = ' \_ : \forall \{i A B\} \rightarrow \text{Delay}' \{i\} A \rightarrow (A \rightarrow \text{Delay } \{i\} B) \rightarrow \text{Delay}' \{i\} B$$

$$\text{force } (a' \gg = ' k) = \text{force } a' \gg = k$$

Depth of  $a' \gg = k$  is at least minimum of depths of  $a'$  and  $k a$ .

## Sized Corecursive Evaluator

Add sizes to type signatures.

$$\llbracket \_ \rrbracket \_ : \forall \{i\} n \rightarrow \mathbf{Tm} \ n \rightarrow \mathbf{Env} \ n \rightarrow \mathbf{Delay} \ \{i\} \ \mathbf{Val}$$

$$\llbracket \mathbf{var} \ x \ \_ \rrbracket \rho = \mathbf{return} \ (\mathbf{lookup} \ x \ \rho)$$

$$\llbracket \mathbf{abs} \ t \ \_ \rrbracket \rho = \mathbf{return} \ (\mathbf{clos} \ t \ \rho)$$

$$\llbracket \mathbf{app} \ r \ s \ \_ \rrbracket \rho = \mathbf{apply} \ (\llbracket r \rrbracket \rho) \ (\llbracket s \rrbracket \rho)$$

$$\mathbf{apply} : \forall \{i\} \rightarrow \mathbf{Delay} \ \{i\} \ \mathbf{Val} \rightarrow \mathbf{Delay} \ \{i\} \ \mathbf{Val} \rightarrow \mathbf{Delay} \ \{i\} \ \mathbf{Val}$$

$$\mathbf{apply} \ u^? \ v^? = u^? \ \gg= \ \lambda \ u \ \rightarrow$$

$$v^? \ \gg= \ \lambda \ v \ \rightarrow$$

$$\mathbf{later} \ (\mathbf{apply}' \ u \ v)$$

$$\mathbf{apply}' : \forall \{i\} \rightarrow \mathbf{Val} \rightarrow \mathbf{Val} \rightarrow \mathbf{Delay}' \ \{i\} \ \mathbf{Val}$$

$$\mathbf{force} \ (\mathbf{apply}' \ (\mathbf{clos} \ t \ \rho) \ v) = \llbracket t \rrbracket (v :: \rho)$$

Termination checker is happy!

## Conclusions

- Copatterns and type-based termination allow for natural corecursive programming.
- Available from Agda 2.4.0.
- Not completely for free; user needs to refine type signatures.
- Size constraint solver could be more powerful.

## Related Work

- 1980/90s: Mendler, Pareto, Amadio, Giménez.
- 2000s: Barthe, Uustalu, Blanqui, Riba, Roux, Gregoire, ...
- Sacchini: LICS 2013,  $\text{Coq}^\wedge$ .
- Coalgebraic types: Hagino (1987), Cockett: Charity (1992).