Coinduction in Agda using Copatterns and Sized Types

Andreas Abel[1]
with James Chapman, Brigitte Pientka, Anton Setzer, David Thibodeau

[1]Department of Computer Science and Engineering
Gothenburg University, Sweden

Workshop on Certification of High-Level and Low-Level Programs
Part of IHP Trimester on Proofs
7 July 2014

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 2.4.0.

Coalgebras

- Copatterns = pattern matching for coalgebras.

$$
\begin{tikzcd}
S \arrow[r, "f"] \arrow[d, "\mathsf{coit}\ f"'] & F(S) \arrow[d, "F(\mathsf{coit}\ f)"] \\
\nu F \arrow[r, "\mathsf{force}"] & F(\nu F)
\end{tikzcd}
$$

- Computation: Only unfold infinite object on demand.

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

Streams as Final Coalgebra

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

$$
\begin{tikzcd}
S \arrow[r, "\langle h,t\rangle"] \arrow[d, "\mathsf{coit}\langle h,t\rangle"'] & A \times S \arrow[d, "\mathsf{id}\times\mathsf{coit}\langle h,t\rangle"] \\
\mathsf{Stream}\ A \arrow[r, "\langle\mathsf{head},\mathsf{tail}\rangle"] & A \times \mathsf{Stream}\ A
\end{tikzcd}
$$

- Termination by induction on observation depth:

$$
\begin{array}{lcl}
\mathsf{head}\ (\mathsf{coit}\ \langle h, t\rangle\ s) & = & h\ s \\
\mathsf{tail}\ (\mathsf{coit}\ \langle h, t\rangle\ s) & = & \mathsf{coit}\ \langle h, t\rangle\ (t\ s)
\end{array}
$$

Copatterns: Syntax

- Elimination contexts (spines):

$$\begin{array}{rcll} E & ::= & \bullet & \text{head} \\ & \mid & E\,t & \text{application} \\ & \mid & \pi\,E & \text{projection} \end{array}$$

- Copatterns = pattern matching elimination contexts.

$$\begin{array}{rcll} Q & ::= & \bullet & \text{head} \\ & \mid & Q\,p & \text{application pattern} \\ & \mid & \pi\,Q & \text{projection pattern} \end{array}$$

- 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 : ∀{A B C} → (A → B → C) → S A → S B → 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 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).

$$\begin{tikzcd} \nu^\alpha F \arrow[r, bend left, "\mathsf{force}"] \arrow[r, phantom, "\cong"] & \bigcap_{\beta<\alpha} F(\nu^\beta F) \arrow[l, bend left, "\mathsf{delay}"] \end{tikzcd}$$

- $\nu^0 F = \top$ 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 ∞.

Sized Fibonacci Stream

```
record Stream i A : Set where
  coinductive
  field head  : A
        tail  : ∀{j : Size< i} → Stream j A
open Stream; S = Stream

zipWith : ∀{i A B C} → (A → B → C) → S i A → S i B → 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 : ∀{i} → Stream i 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 : ∀{j : Size< i} → Stream j A
open Stream; S = Stream

zipWith : ∀{i A B C} → (A → B → C) → S i A → S i B → 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 : ∀{i} → Stream i 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
    constructor  clos
    field        {n}   : ℕ
                 body  : Tm (suc n)
                 env   : Env n
  Env  = Vec Val
```

Running Example: Naive Call-By-Value Interpreter

Evaluator (draft).

$$
\begin{array}{l}
\mathsf{mutual} \\
\quad [\![\_]\!]\_ : \forall\{n\} \to \mathsf{Tm}\ n \to \mathsf{Env}\ n \to \mathsf{Val} \\
\quad [\![\ \mathsf{var}\ x\ ]\!]\ \rho = \mathsf{lookup}\ x\ \rho \\
\quad [\![\ \mathsf{abs}\ t\ ]\!]\ \rho = \mathsf{clos}\ t\ \rho \\
\quad [\![\ \mathsf{app}\ r\ s\ ]\!]\ \rho = \mathsf{apply}\ ([\![\ r\ ]\!]\ \rho)\ ([\![\ s\ ]\!]\ \rho) \\
\\
\quad \mathsf{apply} : \mathsf{Val} \to \mathsf{Val} \to \mathsf{Val} \\
\quad \mathsf{apply}\ (\mathsf{clos}\ t\ \rho)\ v = [\![\ t\ ]\!]\ (v :: \rho)
\end{array}
$$

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.

$$\mathsf{forever} : \forall\{A\} \to \mathsf{Delay}'\ A$$
$$\mathsf{force}\ \mathsf{forever} = \mathsf{later}\ \mathsf{forever}$$

Monad instance.

mutual

$$\_\gg\!=\_ : \forall\{A\ B\} \to \mathsf{Delay}\ A \to (A \to \mathsf{Delay}\ B) \to \mathsf{Delay}\ B$$
$$\mathsf{return}\ a \quad \gg\!= k = k\ a$$
$$\mathsf{later}\ a' \quad \gg\!= k = \mathsf{later}\ (a' \gg\!=' k)$$

$$\_\gg\!='\_ : \forall\{A\ B\} \to \mathsf{Delay}'\ A \to (A \to \mathsf{Delay}\ B) \to \mathsf{Delay}'\ B$$
$$\mathsf{force}\ (a' \gg\!=' k) = \mathsf{force}\ a' \gg\!= k$$

Evaluation In The Delay Monad

Monadic evaluator.

$$
\begin{array}{l}
[\![\_]\!]\_ : \forall\{n\} \to \mathsf{Tm}\ n \to \mathsf{Env}\ n \to \mathsf{Delay}\ \mathsf{Val} \\
[\![\ \mathsf{var}\ x\ ]\!]\ \rho = \mathsf{return}\ (\mathsf{lookup}\ x\ \rho) \\
[\![\ \mathsf{abs}\ t\ ]\!]\ \rho = \mathsf{return}\ (\mathsf{clos}\ t\ \rho) \\
[\![\ \mathsf{app}\ r\ s\ ]\!]\ \rho = \mathsf{apply}\ ([\![\ r\ ]\!]\ \rho)\ ([\![\ s\ ]\!]\ \rho)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{apply} : \mathsf{Delay}\ \mathsf{Val} \to \mathsf{Delay}\ \mathsf{Val} \to \mathsf{Delay}\ \mathsf{Val} \\
\mathsf{apply}\ u?\ v? = \ u? \gg\!= \lambda\ u \to \\
\qquad\qquad\qquad\ \ v? \gg\!= \lambda\ v \to \\
\qquad\qquad\qquad\ \ \mathsf{later}\ (\mathsf{apply}'\ u\ v)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{apply}' : \mathsf{Val} \to \mathsf{Val} \to \mathsf{Delay}'\ \mathsf{Val} \\
\mathsf{force}\ (\mathsf{apply}'\ (\mathsf{clos}\ t\ \rho)\ v) = [\![\ t\ ]\!]\ (v :: \rho)
\end{array}
$$

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 laters”?

Sized Coinductive Delay Monad (II)

Corecursion = induction on depth.

$$\mathsf{forever} : \forall\{i\ A\} \to \mathsf{Delay}'\ \{i\}\ A$$
$$\mathsf{force}\ (\mathsf{forever}\ \{i\})\ \{j\} = \mathsf{later}\ (\mathsf{forever}\ \{j\})$$

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

Sized Coinductive Delay Monad (III)

Monadic bind preserves depth.

$$
\begin{array}{l}
\textsf{mutual} \\
\quad \_\!\gg\!=\!\_ \quad : \ \forall\{i\ A\ B\} \to \\
\qquad\qquad\qquad \mathsf{Delay}\ \{i\}\ A \to (A \to \mathsf{Delay}\ \{i\}\ B) \to \mathsf{Delay}\ \{i\}\ B \\
\quad \mathsf{return}\ a \ \ \gg\!= k = k\ a \\
\quad \mathsf{later}\ a' \quad \gg\!= k = \mathsf{later}\ (a' \gg\!=' k) \\
\\
\quad \_\!\gg\!='\!\_ \quad : \ \forall\{i\ A\ B\} \to \\
\qquad\qquad\qquad \mathsf{Delay}'\ \{i\}\ A \to (A \to \mathsf{Delay}\ \{i\}\ B) \to \mathsf{Delay}'\ \{i\}\ B \\
\quad \mathsf{force}\ (a' \gg\!=' k) = \mathsf{force}\ a' \gg\!= k
\end{array}
$$

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

Sized Corecursive Evaluator

Add sizes to type signatures.

$$
\begin{array}{l}
[\![\_]\!]\_ : \forall\{i\ n\} \to \mathsf{Tm}\ n \to \mathsf{Env}\ n \to \mathsf{Delay}\ \{i\}\ \mathsf{Val} \\
[\![\ \mathsf{var}\ \ x\ \ ]\!]\ \rho = \mathsf{return}\ (\mathsf{lookup}\ x\ \rho) \\
[\![\ \mathsf{abs}\ \ t\ \ ]\!]\ \rho = \mathsf{return}\ (\mathsf{clos}\ t\ \rho) \\
[\![\ \mathsf{app}\ \ r\ s\ ]\!]\ \rho = \mathsf{apply}\ ([\![\ r\ ]\!]\ \rho)\ ([\![\ s\ ]\!]\ \rho)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{apply} : \forall\{i\} \to \mathsf{Delay}\ \{i\}\ \mathsf{Val} \to \mathsf{Delay}\ \{i\}\ \mathsf{Val} \to \mathsf{Delay}\ \{i\}\ \mathsf{Val} \\
\mathsf{apply}\ u?\ v? = \ u?\ \ \gg\!=\ \lambda\ u\ \to \\
\qquad\qquad\qquad\quad v?\ \ \gg\!=\ \lambda\ v\ \to \\
\qquad\qquad\qquad\quad \mathsf{later}\ (\mathsf{apply}'\ u\ v)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{apply}' : \forall\{i\} \to \mathsf{Val} \to \mathsf{Val} \to \mathsf{Delay}'\ \{i\}\ \mathsf{Val} \\
\mathsf{force}\ (\mathsf{apply}'\ (\mathsf{clos}\ t\ \rho)\ v) = [\![\ t\ ]\!]\ (v :: \rho)
\end{array}
$$

Termination checker is happy!

Conclusions

- Type-based termination allows for natural corecursive programming.
  - Well-founded induction works around termination checker.
  - Nice work-around productivity checker?! (Danielsson 2010: DSLs, invasive.)
- Compatible with Isomorphism-as-Equality (HoTT).
- Available now!
- 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, $\widehat{\text{Coq}}$.
- Coalgebraic types: Hagino (1987), Cockett: Charity (1992).
- Acknowledgments:
  - Invitations to McGill (Pientka), Tallinn (Uustalu).
  - Slides generated by Stevan Andjelkovic's LaTeX backend to Agda.