Wellfounded Recursion with Copatterns

Andreas Abel

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

Coinduction Workshop
Shonan Village, near Tokyo, Japan
7-10 October 2013

This is joint work with Brigitte Pientka.
Codata is described by observations.
Syntactically, these are copatterns.
Coinduction is induction on observation depth.
Observation depth is tracked via sized types.
Sized types realize a simple but powerful termination/productivity checker,
which scales to higher-order and abstraction.
Productivity Checking

- Coinductive structures: streams, processes, servers, continuous computation.
- Productivity: each request returns an answer after some time.
- Request on stream: *give me the next element.*
- Dependently typed languages have a productivity checker:
  \[ \text{nats} = 0 :: \text{map} \ (1 + \_ \) \ nats \]
  
- Rejected by Coq and Agda’s syntactic guardedness check.
Better Productivity Checking with Sized Types?

- **MiniAgda**: Prototypical implementation of sized types (with Karl Mehltretter).
  
  http://www.tcs.ifi.lmu.de/~abel/miniagda/

- On-paper approaches to sized types did not scale well to deep pattern matching.

- For corecursive definitions, a *dual to patterns* was called for:

  **Copatterns**
Coinduction and Dependent Types

- Consider the corecursively defined stream $a :: a :: a :: \ldots$

  $$\text{repeat } a = a :: \text{repeat } a$$

- A dilemma:
  - Checking dependent types needs **strong** reduction.
  - Corecursion needs **lazy** evaluation.

- The current compromise (Coq, Agda):
  Corecursive definitions are unfolded only under elimination.

  $$\begin{align*}
  \text{repeat } a & \not\rightarrow \\
  (\text{repeat } a).\text{tail} & \rightarrow (a :: \text{repeat } a).\text{tail} \rightarrow \text{repeat } a
  \end{align*}$$

- Reduction is context-sensitive.
Issues with Context-Sensitive Reduction

- Subject reduction is lost (Giménez 1996, Oury 2008).
- The Fibonacci stream is still diverging:

  \[
  \text{fib} = 0 :: 1 :: \text{adds fib (fib.tail)}
  \]

  \[
  \text{fib.tail} \rightarrow 1 :: \text{adds fib (fib.tail)}
  \]
  \[
  \rightarrow 1 :: \text{adds fib (1 :: adds fib (fib.tail))}
  \]
  \[
  \rightarrow \ldots
  \]

- At POPL, we presented a solution:

Copatterns — The Principle

- Define **infinite** objects (streams, functions) by observations.
- A function is defined by its applications.
- A stream by its head and tail.

\[
\begin{align*}
\text{repeat } a \ . \text{head} &= a \\
\text{repeat } a \ . \text{tail} &= \text{repeat } a
\end{align*}
\]

- These equations are taken as **reduction rules**.
- repeat \( a \) does not reduce by itself.
- No extra laziness required.
Deep Observations

- Any covering set of observations allowed for definition:

\[
\begin{align*}
  \text{fib.head} &= 0 \\
  \text{fib.tail.head} &= 1 \\
  \text{fib.tail.tail} &= \text{adds fib (fib.tail)}
\end{align*}
\]

- Now \text{fib.tail} is stuck. Good!

<table>
<thead>
<tr>
<th>Depth</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observations</td>
<td>id</td>
<td>.head</td>
<td>.tail.head</td>
<td>.tail.tail</td>
</tr>
</tbody>
</table>
Stream Productivity

Definition (Productive Stream)

A stream is **productive** if all observations on it converge.

- Example of non-productiveness:

  \[
  \text{bla} = 0 :: \text{bla}.\text{tail}
  \]

- Observation \( \text{bla}.\text{tail} \) diverges.

- This is apparent in copattern style...

  \[
  \begin{align*}
  \text{bla}.\text{head} & = 0 \\
  \text{bla}.\text{tail} & = \text{bla}.\text{tail}
  \end{align*}
  \]
Theorem (repeat is productive)

\[ \text{repeat } a . \text{tail}^n \text{ converges for all } n \geq 0. \]

Proof.

By induction on \( n \).

Base (repeat \( a \)).tail\(^0\) = repeat \( a \) does not reduce.

Step (repeat \( a \)).tail\(^{n+1}\) = (repeat \( a \)).tail.tail\(^n\) \(\rightarrow\) (repeat \( a \)).tail\(^n\) which converges by induction hypothesis.
Productive Functions

Definition (Productive Function)
A function on streams is productive if it maps productive streams to productive streams.

\[(\text{adds } s \ t).\text{head} = s.\text{head} + t.\text{head}\]
\[(\text{adds } s \ t).\text{tail} = \text{adds} (s.\text{tail}) (t.\text{tail})\]

- *Productivity* of *adds* not sufficient for *fib*!
- Malicious *adds*:

  \[\text{adds'} s \ t = t.\text{tail}\]
  \[\text{fib}.\text{tail}.\text{tail} \rightarrow \text{adds'} \text{fib} (\text{fib}.\text{tail})\]
  \[\rightarrow \text{fib}.\text{tail}.\text{tail} \rightarrow \ldots\]
Definition (Productive Stream)
A stream \( s \) is \( i \)-productive if all observations of depth \( < i \) converge.
Notation: \( s : \text{Stream}^i \).

Lemma
adds : \( \text{Stream}^i \rightarrow \text{Stream}^i \rightarrow \text{Stream}^i \) for all \( i \).

Theorem
\( \text{fib} \) is \( i \)-productive for all \( i \).

Proof, case \( i + 2 \): Show \( \text{fib} \) is \( (i + 2) \)-productive.

Show \( \text{fib.tail.tail} \) is \( i \)-productive.
IH: \( \text{fib} \) is \( (i + 1) \)-productive, so \( \text{fib} \) is \( i \)-productive. (Subtyping!)
IH: \( \text{fib} \) is \( (i + 1) \)-productive, so \( \text{fib.tail} \) is \( i \)-productive.
By Lemma, adds \( \text{fib} \) (fib.tail) is \( i \)-productive.
Type System for Productivity

- “Church $F_\omega$ with inflationary and deflationary fixed-point types”.
- Coinductive types $=$ deflationary iteration:

$$\text{Stream}^i A = \bigcap_{j<i} (A \times \text{Stream}^j A)$$

- Bidirectional type-checking:
- Type inference $\Gamma \vdash r \Rightarrow A$ and checking $\Gamma \vdash t \Leftarrow A$.

$$\Gamma \vdash r \Rightarrow \text{Stream}^i A$$

$$\Gamma \vdash r . \text{tail} \Rightarrow \forall j<i. \text{Stream}^j A \quad \Gamma \vdash a < i$$

$$\Gamma \vdash r . \text{tail} a : \text{Stream}^a A$$
Copattern typing

- Fibonacci again (official syntax with explicit sizes).

\[
\begin{align*}
\text{fib} : \forall i. |i| & \Rightarrow \text{Stream}^{i\mathbb{N}} \\
\text{fib} i \cdot \text{head} j & = 0 \\
\text{fib} i \cdot \text{tail} j \cdot \text{head} k & = 1 \\
\text{fib} i \cdot \text{tail} j \cdot \text{tail} k & = \text{adds} k (\text{fib} k) (\text{fib} j \cdot \text{tail} k)
\end{align*}
\]

- Copattern inference \( \Delta \mid A \vdash \vec{q} \Rightarrow C \) (linear).

\[
\begin{align*}
\text{·} & \mid \text{Stream}^{k\mathbb{N}} \vdash \text{·} \Rightarrow \text{Stream}^{k\mathbb{N}} \\
\forall k < j & \mid k < j \Rightarrow \text{Stream}^{k\mathbb{N}} \vdash k \Rightarrow \text{Stream}^{k\mathbb{N}} \\
\forall j < i & \mid j < i, k < j \Rightarrow \text{Stream}^{j\mathbb{N}} \vdash j \cdot \text{tail} k \Rightarrow \text{Stream}^{k\mathbb{N}} \\
\forall j < i & \mid j < i, k < j \Rightarrow \text{Stream}^{i\mathbb{N}} \vdash \text{.tail} j \cdot \text{tail} k \Rightarrow \text{Stream}^{k\mathbb{N}}
\end{align*}
\]

- Type of recursive call \( \text{fib} : \forall i' < i. \text{Stream}^{i'\mathbb{N}} \)
Pattern typing rules

\[ \Delta; \Gamma \vdash_{\Delta_0} p \equiv A \]  Pattern typing (linear).

In: \( \Delta_0, p, A \) with \( \Delta_0 \vdash A \). Out: \( \Delta, \Gamma \) with \( \Delta_0, \Delta; \Gamma \vdash p \equiv A \).

\[
\begin{align*}
\therefore x: A \vdash_{\Delta_0} x & \equiv A \\
\therefore \cdot \vdash_{\Delta_0} () & \equiv 1
\end{align*}
\]

\[
\begin{align*}
\Delta_1; \Gamma_1 \vdash_{\Delta_0} p_1 & \equiv A_1 \\
\Delta_2; \Gamma_2 \vdash_{\Delta_0} p_2 & \equiv A_2
\end{align*}
\]

\[
\Delta_1, \Delta_2; \Gamma_1, \Gamma_2 \vdash_{\Delta_0} (p_1, p_2) \equiv A_1 \times A_2
\]

\[
\begin{align*}
\Delta; \Gamma \vdash_{\Delta_0} p & \equiv \exists j < a^\uparrow. S_c (\mu^j S) \\
\Delta; \Gamma \vdash_{\Delta_0} c p & \equiv \mu^a S
\end{align*}
\]

\[
\begin{align*}
\Delta; \Gamma \vdash_{\Delta_0, X: \kappa} p & \equiv F @^\kappa X \\
X: \kappa, \Delta; \Gamma \vdash_{\Delta_0} X p & \equiv \exists \kappa F
\end{align*}
\]
Copattern typing rules

Pattern spine typing. In: $\Delta_0, A, \bar{q}$ with $\Delta_0 \vdash A$.
Out: $\Delta, \Gamma, C$ with $\Delta_0, \Delta; \Gamma \vdash C$ and $\Delta_0, \Gamma, z:A \vdash z \bar{q} \Rightarrow C$.

$$
\begin{align*}
\Delta; \Gamma \mid A \vdash_{\Delta_0} \bar{q} \Rightarrow C \\
\Delta_1; \Gamma_1 \vdash_{\Delta_0} p \Leftarrow A \\
\Delta_2; \Gamma_2 \mid B \vdash_{\Delta_0} \bar{q} \Rightarrow C \\
\Delta_1, \Delta_2; \Gamma_1, \Gamma_2 \mid A \rightarrow B \vdash_{\Delta_0} p \bar{q} \Rightarrow C
\end{align*}
$$

$$
\begin{align*}
\Delta; \Gamma \mid \forall j < a^{\uparrow}. R_d (\nu^j R) \vdash_{\Delta_0} \bar{q} \Rightarrow C \\
\Delta; \Gamma \mid \nu^a R \vdash_{\Delta_0} d \bar{q} \Rightarrow C \\
\Delta; \Gamma \mid F @^\kappa X \vdash_{\Delta_0, X: \kappa} \bar{q} \Rightarrow C \\
X: \kappa, \Delta; \Gamma \mid \forall \kappa F \vdash_{\Delta_0} X \bar{q} \Rightarrow C
\end{align*}
$$
Semantics

- **Reduction:**
  \[
  \begin{align*}
  \bar{e} / \bar{q} \xrightarrow{\sigma} & \quad \lambda\{\bar{q} \rightarrow t\} \bar{e} \bar{e}' \mapsto t\sigma\bar{e}' \\
  \lambda D_k \bar{e} \mapsto t & \quad f \bar{e} \mapsto t \\
  (f : A = \bar{D}) \in \Sigma
  \end{align*}
  \]

- **Types are reducibility candidates** \(A\):
  - \(A\) is a set of strongly normalizing terms.
  - \(A\) is closed under reduction.
  - \(A\) is closed under addition of well-behaved neutrals (redexes and terminally stuck terms).
  - \(A\) is closed under simulation:
    - \(r\) is simulated by \(r_{1..n}\) if \(r \bar{e} \mapsto t\) implies \(r_k \bar{e} \mapsto t\) for some \(k\).
Conclusions

- A unified approach to termination and productivity: Induction.
  - Recursion as induction on data size.
  - Corecursion as induction on observation depth.
- Adaption of sized types to deep (co)patterens:
  - Shift to in-/deflationary fixed-point types.
  - Bounded size quantification.
- Implementations:
  - MiniAgda: ready to play with!
  - Agda: under development.

Andreas Abel and Brigitte Pientka.
Wellfounded recursion with copatterns:
A unified approach to termination and productivity.
Some Related Work

- Sized types: many authors (1996–)
- Inflationary fixed-points: Dam & Sprenger (2003)
- Observation-centric coinduction and coalgebras: Hagino (1987), Cockett & Fukushima (Charity, 1992)
- Form of termination measures taken from Xi (2002)
- Guarded types: next talk!