Wellfounded Recursion with Copatterns

Andreas Abel\textsuperscript{1}  Brigitte Pientka\textsuperscript{2}

\textsuperscript{1}Department of Computer Science and Engineering
\textbf{Chalmers} and Gothenburg University, Sweden

\textsuperscript{2}School of Computer Science
McGill University, Montreal, Canada

International Conference on Functional Programming
Boston, MA, USA
26 September 2013
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**:

  \[ \text{nats} = 0 :: \text{map} (1 + _) \text{nats} \]

- Coq says: **Unguarded recursive call**.
- Agda sees **red**.
Better Productivity Checking with Sized Types?

John Hughes, Lars Pareto, and Amr Sabry.
Proving the correctness of reactive systems using sized types.
Better Productivity Checking with Sized Types?

John Hughes, Lars Pareto, and Amr Sabry.
Proving the correctness of reactive systems using sized types.

Andreas Abel, Type-Based Termination
ISBN 978-3-938363-04-1

Only 39.80 €
Order today!
Better Productivity Checking with Sized Types?

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

- 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.

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

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

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

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

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

- At POPL, we presented a solution:

  \[
  \begin{array}{l}
  \text{A. Abel, B. Pientka, D. Thibodeau, and A. Setzer.} \\
  \text{Copatterns: Programming infinite structures by observations.} \\
  \text{In POPL’13, pages 27–38. ACM, 2013.}
  \end{array}
  \]
Copatterns — The Principle

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

\[
\begin{align*}
\text{repeat } a \cdot \text{head} & \Rightarrow a \\
\text{repeat } a \cdot \text{tail} & \Rightarrow \text{repeat } a
\end{align*}
\]

- 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:

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

- Now \text{fib.} \text{.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>
<tr>
<td></td>
<td></td>
<td>.tail</td>
<td>.tail.tail</td>
<td>...</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)

repeat a .tail^n converges for all n ≥ 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 → (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 \text{adds} not sufficient for \text{fib}!
- Malicious \text{adds}:
  \[
  \text{adds'} s \ t = t.\text{tail}
  \]
  \[
  \text{fib}.\text{tail}.\text{tail} \rightarrow \text{adds'} \text{fib} (\text{fib}.\text{tail})
  \]
  \[
  \rightarrow \text{fib}.\text{tail}.\text{tail} \rightarrow \ldots
  \]
**i-Productivity**

**Definition (Productive Stream)**
A stream $s$ is $i$-productive if all observations of depth $< i$ converge.

Notation: $s : \text{Stream}^i$.

**Lemma**

$\text{adds} : \text{Stream}^i \rightarrow \text{Stream}^i \rightarrow \text{Stream}^i$ 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, $\text{adds}\ fib\ (\text{fib.tail})$ 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 $\Gamma \vdash t \Leftarrow A$.

\[
\begin{align*}
\Gamma \vdash r \Rightarrow \text{Stream}^i A \\
\Gamma \vdash r . \text{tail} \Rightarrow \forall j < i. \text{Stream}^j A \\
\Gamma \vdash a < i \\
\Gamma \vdash r . \text{tail} a : \text{Stream}^a A
\end{align*}
\]
Copattern typing

- Fibonacci again (official syntax with explicit sizes).

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

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

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

- Type of recursive call \( \text{fib} : \forall i' < i. \text{Stream}^{i'} \mathbb{N} \)
What else is in the paper?

- **Conference version:**
  - Full type checking rules.
  - Inductive types as inflationary fixed-points.
  - Patterns and pattern typing.
  - Transfinite size and depth.
  - Lexicographic termination measures.
  - Declarations and mutual recursion.
  - Example for mixed induction-coinduction.
  - Adaption of Girard’s reducibility candidates.
  - Strong normalization proof (sketch).

- **Full version:**
  - Declaration typing.
  - Kinding and subtyping rules.
  - Semantics of kinds and type constructors.
  - Strong normalization proof (full).
Conclusions

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