Wellfounded Recursion with Copatterns

Andreas Abel[1] Brigitte Pientka[2]

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

[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:

$$\mathsf{nats} = 0 :: \mathsf{map}\ (1 + \_)\ \mathsf{nats}$$

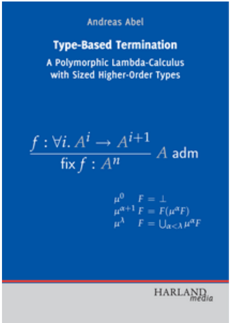
- Coq says: `Unguarded recursive call`.
- Agda sees red.

Better Productivity Checking with Sized Types?

John Hughes, Lars Pareto, and Amr Sabry.
Proving the correctness of reactive systems using sized types.
In *POPL'96*, pages 410–423, 1996.

Better Productivity Checking with Sized Types?

John Hughes, Lars Pareto, and Amr Sabry.
Proving the correctness of reactive systems using sized types.
In *POPL'96*, pages 410–423, 1996.



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/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.

$$\begin{array}{llllll} \text{repeat } a & \not\longrightarrow & & & & \\ (\text{repeat } a).\text{tail} & \longrightarrow & (a :: \text{repeat } a).\text{tail} & \longrightarrow & \text{repeat } a \end{array}$$

- Reduction is context-sensitive.

Issues with Context-Sensitive Reduction

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

$$\mathsf{fib} = 0 :: 1 :: \mathsf{adds}\ \mathsf{fib}\ (\mathsf{fib}.\mathsf{tail})$$

$$\begin{array}{lcl} \mathsf{fib}.\mathsf{tail} & \longrightarrow & 1 :: \mathsf{adds}\ \mathsf{fib}\ (\mathsf{fib}.\mathsf{tail}) \\ & \longrightarrow & 1 :: \mathsf{adds}\ \mathsf{fib}\ (1 :: \mathsf{adds}\ \mathsf{fib}\ (\mathsf{fib}.\mathsf{tail})) \\ & \longrightarrow & \dots \end{array}$$

- At POPL, we presented a solution:

A. Abel, B. Pientka, D. Thibodeau, and A. Setzer.
Copatterns: Programming infinite structures by observations.
In *POPL'13*, pages 27–38. ACM, 2013.

Copatterns — The Principle

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

$$
\begin{array}{lcl}
\mathsf{repeat}\ a\ .\mathsf{head} & = & a \\
\mathsf{repeat}\ a\ .\mathsf{tail} & = & \mathsf{repeat}\ a
\end{array}
$$

- 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:

$$
\begin{aligned}
\text{fib.head} &= 0 \\
\text{fib.tail.head} &= 1 \\
\text{fib.tail.tail} &= \text{adds fib (fib.tail)}
\end{aligned}
$$

- Now fib.tail is stuck. Good!

Depth	0	1	2	...
Observations	id	.head	.tail.head	...
		.tail	.tail.tail	...

Stream Productivity

Definition (Productive Stream)

A stream is productive if all observations on it converge.

- Example of non-productiveness:

$$\mathsf{bla} = 0 :: \mathsf{bla}.\mathsf{tail}$$

- Observation bla.tail diverges.
- This is apparent in copattern style...

$$\begin{array}{lcl} \mathsf{bla}\ .\mathsf{head} & = & 0 \\ \mathsf{bla}\ .\mathsf{tail} & = & \mathsf{bla}\ .\mathsf{tail} \end{array}$$

Proving Productivity

Theorem (repeat is productive)

repeat a .tail^n 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 $\longrightarrow$ (repeat a).tail^n which converges by induction hypothesis.

$\square$

Productive Functions

Definition (Productive Function)

A function on streams is productive if it maps productive streams to productive streams.

$$
\begin{aligned}
(\mathsf{adds}\ s\ t).\mathsf{head} &= s.\mathsf{head} + t.\mathsf{head} \\
(\mathsf{adds}\ s\ t).\mathsf{tail} &= \mathsf{adds}\ (s.\mathsf{tail})\ (t.\mathsf{tail})
\end{aligned}
$$

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

$$
\begin{array}{lcl}
\mathsf{adds}'\ s\ t & = & t.\mathsf{tail} \\
\mathsf{fib}.\mathsf{tail}.\mathsf{tail} & \longrightarrow & \mathsf{adds}'\ \mathsf{fib}\ (\mathsf{fib}.\mathsf{tail}) \\
 & \longrightarrow & \mathsf{fib}.\mathsf{tail}.\mathsf{tail} \longrightarrow \dots
\end{array}
$$

i-Productivity

Definition (Productive Stream)

A stream s is i-productive if all observations of depth $< i$ converge.
Notation: $s : \mathrm{Stream}^i$.

Lemma

$\mathrm{adds} : \mathrm{Stream}^i \to \mathrm{Stream}^i \to \mathrm{Stream}^i$ *for all* i*.*

Theorem

fib *is* i*-productive for all* i*.*

Proof, case $i + 2$: Show fib is $(i + 2)$-productive.

Show fib.tail.tail is i-productive.
IH: fib is $(i + 1)$-productive, so fib is i-productive. (Subtyping!)
IH: fib is $(i + 1)$-productive, so fib.tail is i-productive.
By Lemma, adds fib (fib.tail) is i-productive. $\square$

Type System for Productivity

- "Church F^ω with inflationary and deflationary fixed-point types".
- Coinductive types = deflationary iteration:

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

- Bidirectional type-checking:
- Type inference $\boxed{\Gamma \vdash r \rightrightarrows A}$ and checking $\boxed{\Gamma \vdash t \leftleftarrows A}$.

$$\dfrac{\dfrac{\Gamma \vdash r \rightrightarrows \mathsf{Stream}^i A}{\Gamma \vdash r\,.\mathsf{tail} \rightrightarrows \forall j{<}i.\,\mathsf{Stream}^j A} \qquad \Gamma \vdash a < i}{\Gamma \vdash r\,.\mathsf{tail}\ a : \mathsf{Stream}^a A}$$

Copattern typing

- Fibonacci again (official syntax with explicit sizes).

$$
\begin{array}{lllll}
\mathsf{fib} & : & \forall i.\ |i| \Rightarrow \mathsf{Stream}^i \mathbb{N} & & \\
\mathsf{fib}\ i\ .\mathsf{head}\ j & & & = & 0 \\
\mathsf{fib}\ i\ .\mathsf{tail} & j\ .\mathsf{head}\ k & & = & 1 \\
\mathsf{fib}\ i\ .\mathsf{tail} & j\ .\mathsf{tail} & k & = & \mathsf{adds}\ k\ (\mathsf{fib}\ k)\ (\mathsf{fib}\ j\ .\mathsf{tail}\ k)
\end{array}
$$

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

$$
\begin{array}{rrcrl}
\cdot \mid & \mathsf{Stream}^k \mathbb{N} & \vdash & \cdot \rightrightarrows & \mathsf{Stream}^k \mathbb{N} \\
\hline
k{<}j \mid & \forall k{<}j.\, \mathsf{Stream}^k \mathbb{N} & \vdash & k \rightrightarrows & \mathsf{Stream}^k \mathbb{N} \\
\hline
k{<}j \mid & \mathsf{Stream}^j \mathbb{N} & \vdash & .\mathsf{tail}\ k \rightrightarrows & \mathsf{Stream}^k \mathbb{N} \\
\hline
j{<}i, k{<}j \mid & \forall j{<}i.\, \mathsf{Stream}^j \mathbb{N} & \vdash & j\ .\mathsf{tail}\ k \rightrightarrows & \mathsf{Stream}^k \mathbb{N} \\
\hline
j{<}i, k{<}j \mid & \mathsf{Stream}^i \mathbb{N} & \vdash & .\mathsf{tail}\ j\ .\mathsf{tail}\ k \rightrightarrows & \mathsf{Stream}^k \mathbb{N}
\end{array}
$$

- Type of recursive call $\mathsf{fib} : \forall i'{<}i.\ \mathit{Stream}^{i'} \mathbb{N}$

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.
- Adaption 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)
- Focusing sequent calculus: Zeilberger & Licata & Harper (2008)
- Form of termination measures taken from Xi (2002)
- Guarded types: next talk!