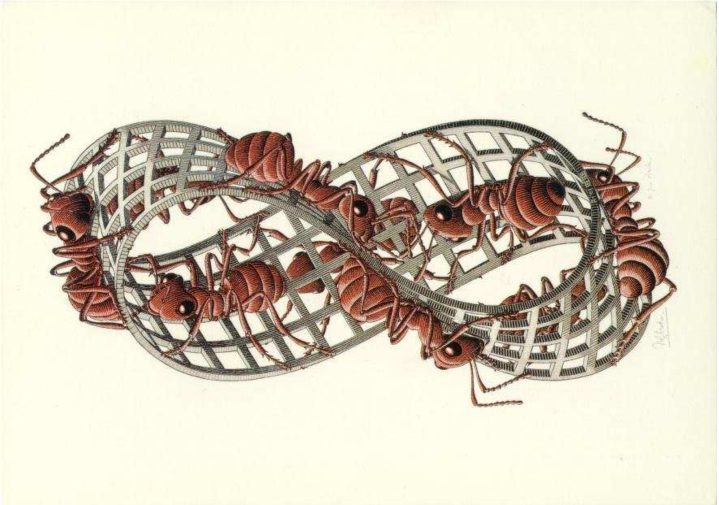
Coalgebras in Type Theory

Venanzio Capretta

CMCS 2010, Paphos, Cyprus

Corecursive Equations



Streams

Streams: infinite sequence over a domain D, $\mathbb{S}_D$.

$$\mathsf{nat} = 0 :: 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: \cdots : \mathbb{S}_{\mathbb{N}}$$
$$\mathsf{fib} = 0 :: 1 :: 1 :: 2 :: 3 :: 5 :: 8 :: \cdots : \mathbb{S}_{\mathbb{N}}$$

Streams

Streams: infinite sequence over a domain D, $\mathbb{S}_D$.

$$\mathsf{nat} = 0 :: 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: \cdots : \mathbb{S}_{\mathbb{N}}$$
$$\mathsf{fib} = 0 :: 1 :: 1 :: 2 :: 3 :: 5 :: 8 :: \cdots : \mathbb{S}_{\mathbb{N}}$$

Notation:

$$\mathsf{head}: \quad {}^{\mathfrak{h}}\mathsf{nat} = 0$$
$$\mathsf{tail}: \quad {}^{\mathfrak{t}}\mathsf{nat} = 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: 7 :: \cdots$$

Streams

Streams: infinite sequence over a domain D, $\mathbb{S}_D$.

$$\mathsf{nat} = 0 :: 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: \cdots : \mathbb{S}_{\mathbb{N}}$$
$$\mathsf{fib} = 0 :: 1 :: 1 :: 2 :: 3 :: 5 :: 8 :: \cdots : \mathbb{S}_{\mathbb{N}}$$

Notation:

$$\mathsf{head} : \quad \mathfrak{h}\,\mathsf{nat} = 0 \quad \mathfrak{h}^3\mathsf{nat} = 3 \quad \mathfrak{h}^3\mathsf{fib} = 2$$
$$\mathsf{tail} : \quad \mathfrak{t}\,\mathsf{nat} = 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: 7 :: \cdots$$

Streams

Streams: infinite sequence over a domain D, $\mathbb{S}_D$.

$$\mathsf{nat} = 0 :: 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: \cdots : \mathbb{S}_{\mathbb{N}}$$
$$\mathsf{fib} = 0 :: 1 :: 1 :: 2 :: 3 :: 5 :: 8 :: \cdots : \mathbb{S}_{\mathbb{N}}$$

Notation:

head : $\quad {}^{\mathfrak{h}}\mathsf{nat} = 0 \quad {}^{\mathfrak{h}3}\mathsf{nat} = 3 \quad {}^{\mathfrak{h}3}\mathsf{fib} = 2$

tail : $\quad {}^{\mathfrak{t}}\mathsf{nat} = 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: 7 :: \cdots$

$${}^{4\mathfrak{t}}\mathsf{nat} = 4 :: 5 :: 6 :: 7 :: 8 :: 9 :: 10 :: \cdots$$
$${}^{4\mathfrak{t}}\mathsf{fib} = 3 :: 5 :: 8 :: 13 :: 21 :: 34 :: 55 :: \cdots$$

Streams

Streams: infinite sequence over a domain D, $\mathbb{S}_D$.

$$\mathsf{nat} = 0 :: 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: \cdots : \mathbb{S}_{\mathbb{N}}$$
$$\mathsf{fib} = 0 :: 1 :: 1 :: 2 :: 3 :: 5 :: 8 :: \cdots : \mathbb{S}_{\mathbb{N}}$$

Notation:

$$\mathsf{head}: \quad \mathfrak{h}\,\mathsf{nat} = 0 \qquad \mathfrak{h}^3\mathsf{nat} = 3 \qquad \mathfrak{h}^3\mathsf{fib} = 2$$
$$\mathsf{tail}: \quad \mathfrak{t}\,\mathsf{nat} = 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: 7 :: \cdots$$
$${}^{4}\mathfrak{t}\,\mathsf{nat} = 4 :: 5 :: 6 :: 7 :: 8 :: 9 :: 10 :: \cdots$$
$${}^{4}\mathfrak{t}\,\mathsf{fib} = 3 :: 5 :: 8 :: 13 :: 21 :: 34 :: 55 :: \cdots$$

Corecursive equations on streams: [Rutten 2007]

$$\mathsf{nat} = 0 :: \mathsf{nat} + 1 \qquad \mathsf{fib} = 0 :: \mathsf{fib} + (1 :: \mathsf{fib})$$
$$(x :: s_1) \ltimes s_2 = x :: s_2 \ltimes s1$$
$$\mathsf{even}\,(x :: s) = x :: \mathsf{odd}\ s \qquad \mathsf{odd}\,(x :: s) = \mathsf{even}\,s$$

Harder corecursive equations

Equations that are more difficult to solve [Zantema 2009]

Three functions of type $\mathbb{S} \to \mathbb{S}$:

$$\phi\ s = {}^{\mathfrak{h}}s :: \phi(\mathsf{even}\ {}^{\mathfrak{t}}s) \ltimes \phi(\mathsf{odd}\ {}^{\mathfrak{t}}s)$$

Harder corecursive equations

Equations that are more difficult to solve [Zantema 2009]

Three functions of type $\mathbb{S} \to \mathbb{S}$:

$$\phi\ s = \mathfrak{h}s :: \phi(\mathsf{even}\ \mathfrak{t}s) \ltimes \phi(\mathsf{odd}\ \mathfrak{t}s)$$

$$\chi\ s = \mathfrak{h}s :: \mathfrak{t}s \ltimes \mathfrak{t}(\chi\ \mathfrak{t}s)$$

Harder corecursive equations

Equations that are more difficult to solve [Zantema 2009]

Three functions of type $\mathbb{S} \to \mathbb{S}$:

$$\phi\ s = {}^{\mathfrak{h}}s :: \phi(\mathsf{even}\ {}^{\mathfrak{t}}s) \ltimes \phi(\mathsf{odd}\ {}^{\mathfrak{t}}s)$$

$$\chi\ s = {}^{\mathfrak{h}}s :: {}^{\mathfrak{t}}s \ltimes {}^{\mathfrak{t}}(\chi\ {}^{\mathfrak{t}}s)$$

$$\psi\ s = {}^{\mathfrak{h}}s :: \mathsf{even}(\psi(\mathsf{odd}\ {}^{\mathfrak{t}}s)) \ltimes \mathsf{odd}(\psi(\mathsf{even}\ {}^{\mathfrak{t}}s))$$

Harder corecursive equations

Equations that are more difficult to solve [Zantema 2009]

Three functions of type $\mathbb{S} \to \mathbb{S}$:

$$\phi\ s = \mathfrak{h}s :: \phi(\mathsf{even}\ \mathfrak{t}s) \ltimes \phi(\mathsf{odd}\ \mathfrak{t}s)$$

$$\chi\ s = \mathfrak{h}s :: \mathfrak{t}s \ltimes \mathfrak{t}(\chi\ \mathfrak{t}s)$$

$$\psi\ s = \mathfrak{h}s :: \mathsf{even}(\psi(\mathsf{odd}\ \mathfrak{t}s)) \ltimes \mathsf{odd}(\psi(\mathsf{even}\ \mathfrak{t}s))$$

Puzzle: Find equation $f\ s = C[s, f]$ that generates:

$$\begin{aligned} f\ \mathsf{nat} = \ & 0 :: 0 :: 1 :: 0 :: 2 :: 1 :: 3 :: 0 :: 4 :: 2 :: 5 :: 1 :: 6 :: 3 :: 7 :: 0 :: 8 \\ & :: 4 :: 9 :: 2 :: 10 :: 5 :: 11 :: 1 :: 12 :: 6 :: 13 :: 3 :: 14 :: 7 :: 15 \\ & :: 0 :: 16 :: 8 :: 17 :: 4 :: 18 :: 9 :: 19 :: 2 :: 20 :: 10 :: 21 :: 5 :: 22 \\ & :: 11 :: 23 :: 1 :: 24 :: 12 :: 25 :: 6 :: 26 :: 13 :: 27 :: 3 :: 28 :: 14 \\ & :: 29 :: 7 :: 30 :: 15 :: 31 :: 0 :: 32 :: 16 :: 33 :: 8 :: 34 :: 17 :: 35 \\ & :: 4 :: 36 :: 18 :: 37 :: 9 :: 38 :: \cdots \end{aligned}$$

Images of streams of Booleans [Zantema]

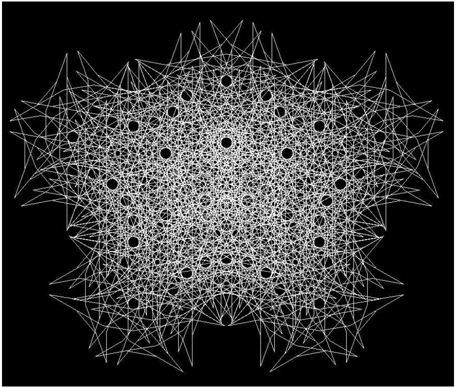
The Boolean Fibonacci stream:

$$f\,(0 :: s) = 0 :: 1 :: f\,s \qquad \mathrm{bfib} = f\,\mathrm{bfib}$$
$$f\,(1 :: s) = 0 :: f\,s$$

Images of streams of Booleans [Zantema]

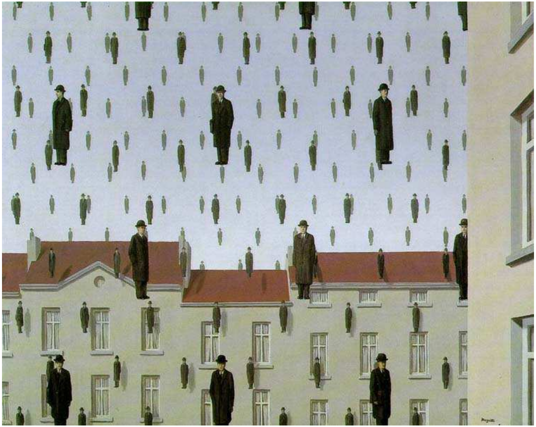
The Boolean Fibonacci stream:

$$f\,(0 :: s) = 0 :: 1 :: f\,s \qquad \mathrm{bfib} = f\,\mathrm{bfib}$$
$$f\,(1 :: s) = 0 :: f\,s$$



bfib = 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0
:1 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1
:0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0
:1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 1
:0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1
:0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0
:1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 1
:0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0
:0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0
:1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1
:0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0
:0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0
:1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1
:0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0
:0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0
:1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0 : 0 : 1
:0 : 0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : 0
:0 : 1 : 0 : 1 : 0 : 0 : 1 : 0 : 0 : 1 : 0 : 1 : · · ·

CoInductive Types



CoInductive Types: [Hagino 1987, Aczel & Mendler 1989]

Type-theoretic implementation of final coalgebras.

CoInductive Definitions

Puzzle: $0:0:1:0:2:1:3:0:4:2:5:1:6:3:7:0:8:4:9:2:10:5:11:1:12:6:13:3:14:7:15$

CoInductive Types: [Hagino 1987, Aczel & Mendler 1989]

Type-theoretic implementation of final coalgebras.

codata $\mathbb{S}_D$: Set
  $(::) : D \to \mathbb{S}_D \to \mathbb{S}_D$

Final Coalgebra:
$\langle {}^{\mathfrak{h}}\_, {}^{\mathfrak{t}}\_ \rangle : \mathbb{S}_D \to D \times \mathbb{S}_D$

CoInductive Types: [Hagino 1987, Aczel & Mendler 1989]

Type-theoretic implementation of final coalgebras.

Final Coalgebra:

codata $\mathbb{S}_D$: Set
$(::) : D \to \mathbb{S}_D \to \mathbb{S}_D$

$\langle {}^{\mathfrak{h}}\_, {}^{\mathfrak{t}}\_ \rangle : \mathbb{S}_D \to D \times \mathbb{S}_D$

codata $\mathbb{T}_{A,B}$: Set
leaf : $B \to \mathbb{T}_{A,B}$
node : $(A \to \mathbb{T}_{A,B}) \to \mathbb{T}_{A,B}$

leaf $b \mapsto$ inl b
node $f \mapsto$ inr f
$: \mathbb{T}_{A,B} \to B + (A \to \mathbb{T}_{A,B})$

CoInductive Types: [Hagino 1987, Aczel & Mendler 1989]

Type-theoretic implementation of final coalgebras.

Final Coalgebra:

codata $\mathbb{S}_D$: Set
  $(::) : D \rightarrow \mathbb{S}_D \rightarrow \mathbb{S}_D$

$\langle {}^{\mathfrak{h}}\_, {}^{\mathfrak{t}}\_ \rangle : \mathbb{S}_D \rightarrow D \times \mathbb{S}_D$

codata $\mathbb{T}_{A,B}$: Set
  leaf : $B \rightarrow \mathbb{T}_{A,B}$
  node : $(A \rightarrow \mathbb{T}_{A,B}) \rightarrow \mathbb{T}_{A,B}$

leaf $b \mapsto$ inl b
node $f \mapsto$ inr f
$: \mathbb{T}_{A,B} \rightarrow B + (A \rightarrow \mathbb{T}_{A,B})$

$(::)$, leaf and node are constructors

Guardedness by constructors: [Coquand 1993]

A corecursive equation has a unique solution if all recursive calls occur only directly under constructor applications.

Example of guarded definition

Puzzle: 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : 4 : 9 : 2 : 10 : 5 : 11 : 1 : 12 : 6 : 13 : 3 : 14 : 7 : 15

Definitions are accepted if they satisfy the guardedness condition [Gimènez 1998]

$$
\begin{aligned}
&\mathsf{fguard} : \mathbb{N} \to \mathbb{S} \\
&\mathsf{fguard}\, n = \mathbf{case}\, \mathsf{mod}(n, 3) \\
&\begin{cases}
0 \mapsto \mathsf{nat} \\
1 \mapsto n :: (n-1) :: \mathsf{fguard}\,(n+1) \\
2 \mapsto n :: \mathsf{map}\,(2 \cdot -)\,(\mathsf{fguard}\,(2 \cdot n))
\end{cases}
\end{aligned}
$$

No recursive calls.
Recursive calls under two constructors.
Map *filters* the constructors.

Some equations don't satisfy guardedness
but they still have a unique solution.

Some equations don't satisfy guardedness
but they still have a unique solution.
More powerful methods:

Some equations don't satisfy guardedness
but they still have a unique solution.
More powerful methods:

- Metrics (fixpoints of contractions) [Di Giannantonio/Miculan 2002]

Methods to solve corecursive equations

Puzzle: 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : 4 : 9 : 2 : 10 : 5 : 11 : 1 : 12 : 6 : 13 : 3 : 14 : 7 : 15

Some equations don't satisfy guardedness
but they still have a unique solution.
More powerful methods:

- Metrics (fixpoints of contractions) [Di Giannantonio/Miculan 2002]
- Pebbleflow Networks [Endrullis/Grabmayer/Hendriks/Isihara/Klop 2008]

Methods to solve corecursive equations

Puzzle: 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : 4 : 9 : 2 : 10 : 5 : 11 : 1 : 12 : 6 : 13 : 3 : 14 : 7 : 15

Some equations don't satisfy guardedness
but they still have a unique solution.
More powerful methods:

- Metrics (fixpoints of contractions) [Di Giannantonio/Miculan 2002]
- Pebbleflow Networks [Endrullis/Grabmayer/Hendriks/Isihara/Klop 2008]
- Circular Coinduction (CIRC) [Roşu/Lucanu 2009]

Methods to solve corecursive equations

Puzzle: 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : 4 : 9 : 2 : 10 : 5 : 11 : 1 : 12 : 6 : 13 : 3 : 14 : 7 : 15

Some equations don't satisfy guardedness
but they still have a unique solution.
More powerful methods:

- Metrics (fixpoints of contractions) [Di Giannantonio/Miculan 2002]
- Pebbleflow Networks [Endrullis/Grabmayer/Hendriks/Isihara/Klop 2008]
- Circular Coinduction (CIRC) [Roșu/Lucanu 2009]
- Termination of Rewriting Systems [Zantema 2009]

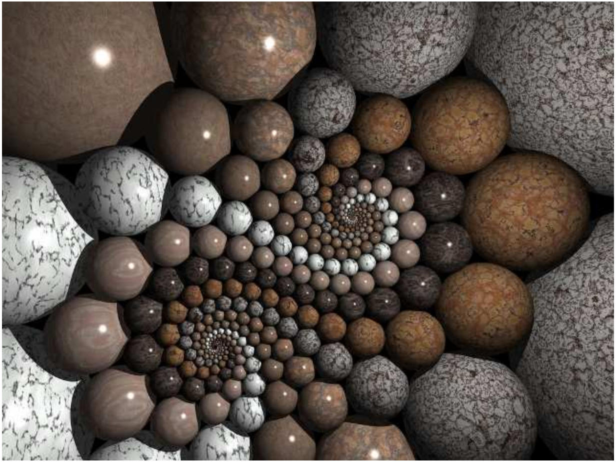
Methods to solve corecursive equations

Puzzle: 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : 4 : 9 : 2 : 10 : 5 : 11 : 1 : 12 : 6 : 13 : 3 : 14 : 7 : 15

Some equations don't satisfy guardedness
but they still have a unique solution.
More powerful methods:

- Metrics (fixpoints of contractions) [Di Giannantonio/Miculan 2002]
- Pebbleflow Networks [Endrullis/Grabmayer/Hendriks/Isihara/Klop 2008]
- Circular Coinduction (CIRC) [Roșu/Lucanu 2009]
- Termination of Rewriting Systems [Zantema 2009]
- Unicity by Bisimulation [VC 2010]

Bisimulations



Bisimulations and the Coinduction Principle

Definition of bisimulation. [Park 1981, Milner 1989]

Bisimulations and the Coinduction Principle

Definition of bisimulation. [Park 1981, Milner 1989]

A relation $\sim$ on a coinductive type is a *bisimulation* if

$$x_1 \sim x_2 \Rightarrow \begin{cases} \text{same top constructor} \\ \text{same non-recursive arguments} \\ \text{recursive arguments related by } \sim \end{cases}$$

Bisimulations and the Coinduction Principle

Definition of bisimulation. [Park 1981, Milner 1989]

A relation $\sim$ on a coinductive type is a *bisimulation* if

$$x_1 \sim x_2 \Rightarrow \begin{cases} \text{same top constructor} \\ \text{same non-recursive arguments} \\ \text{recursive arguments related by } \sim \end{cases}$$

On Streams: $s_1 \sim s_2 \Rightarrow \mathfrak{h} s_1 = \mathfrak{h} s_2 \wedge \mathfrak{t} s_1 \sim \mathfrak{t} s_2$

Bisimulations and the Coinduction Principle

Definition of bisimulation. [Park 1981, Milner 1989]

A relation $\sim$ on a coinductive type is a *bisimulation* if

$$x_1 \sim x_2 \Rightarrow \begin{cases} \text{same top constructor} \\ \text{same non-recursive arguments} \\ \text{recursive arguments related by } \sim \end{cases}$$

On Streams: $s_1 \sim s_2 \Rightarrow {}^{\mathfrak{h}}s_1 = {}^{\mathfrak{h}}s_2 \wedge {}^{\mathfrak{t}}s_1 \sim {}^{\mathfrak{t}}s_2$

On Trees:

$$t_1 \sim t_2 \Rightarrow \begin{cases} t_1 = \mathsf{leaf}\ b = t_2 \quad \vee \\ t_1 = \mathsf{node}\ f_1 \wedge t_2 = \mathsf{node}\ f_2 \\ \quad \wedge \forall a. f_1\ a \sim f_2\ a \end{cases}$$

Bisimulations and the Coinduction Principle

Definition of bisimulation. [Park 1981, Milner 1989]

A relation $\sim$ on a coinductive type is a *bisimulation* if

$$x_1 \sim x_2 \Rightarrow \begin{cases} \text{same top constructor} \\ \text{same non-recursive arguments} \\ \text{recursive arguments related by } \sim \end{cases}$$

On Streams: $s_1 \sim s_2 \Rightarrow \mathfrak{h} s_1 = \mathfrak{h} s_2 \wedge \mathfrak{t} s_1 \sim \mathfrak{t} s_2$

On Trees:

$$t_1 \sim t_2 \Rightarrow \begin{cases} t_1 = \mathsf{leaf}\ b = t_2 \quad \vee \\ t_1 = \mathsf{node}\ f_1 \wedge t_2 = \mathsf{node}\ f_2 \\ \quad \wedge \forall a. f_1\ a \sim f_2\ a \end{cases}$$

The Coinduction principle:

$$x_1 \sim x_2 \Rightarrow x_1 = x_2.$$

Bisimulation as a coinductive relation

The Coinduction principle doesn't hold in Type Theory:
Equality is intentional: Equality of normal forms

Bisimulation as a coinductive relation

The Coinduction principle doesn't hold in Type Theory:
Equality is intentional: Equality of normal forms
Instead: Bisimilarity is defined as a coinductive relation:

$$\begin{array}{l} \textbf{codata}\,(\approx) : \mathbb{S} \to \mathbb{S} \to \mathrm{Prop} \\ \mathrm{conssim} : (x : D)(s_1, s_2 : \mathbb{S}) s_1 \approx s_2 \to (x :: s_1) \approx (x :: x_2) \end{array}$$

Bisimulation as a coinductive relation

The Coinduction principle doesn't hold in Type Theory:
Equality is intentional: Equality of normal forms
Instead: Bisimilarity is defined as a coinductive relation:

$$\textbf{codata}\,(\approx) : \mathbb{S} \to \mathbb{S} \to \mathsf{Prop}$$
$$\mathsf{conssim} : (x : D)(s_1, s_2 : \mathbb{S}) s_1 \approx s_2 \to (x :: s_1) \approx (x :: x_2)$$

$$\textbf{codata}\,(\approx) : \mathbb{T} \to \mathbb{T} \to \mathsf{Prop}$$
$$\mathsf{leafsim} : (b : B)\mathsf{leaf}\ b \approx \mathsf{leaf}\ b$$
$$\mathsf{nodesim} : (f_1, f_2 : A \to \mathbb{T})(\forall a. f_1\ a \approx f_2\ a) \to (\mathsf{node}\ f_1) \approx (\mathsf{node}\ f_2)$$

Bisimulation and Unicity of Solutions

Unicity of solutions for the equation:

$$\chi : \mathbb{S} \to \mathbb{S}$$
$$\chi\ s = {}^{\mathfrak{h}}s :: {}^{\mathfrak{t}}s \ltimes {}^{\mathfrak{t}}(\chi\ {}^{\mathfrak{t}}s)$$

Bisimulation and Unicity of Solutions

Coalgebras in Type Theory

Venanzio Capretta

Unicity of solutions for the equation:

$$\chi : \mathbb{S} \to \mathbb{S}$$
$$\chi\ s = {}^{\mathfrak{h}}s :: {}^{\mathfrak{t}}s \ltimes {}^{\mathfrak{t}}(\chi\ {}^{\mathfrak{t}}s)$$

Suppose χ_1 and χ_2 are solutions.

Bisimulation and Unicity of Solutions

Coalgebras in Type Theory

Venanzio Capretta

Corecursive Equations

CoInductive Types

Bisimulations

Constructive Infinity

Tabulations

General Recursion

Non-Standard Type Theory

Mixing Induction and Coinduction

Unicity of solutions for the equation:

$$\chi : \mathbb{S} \to \mathbb{S}$$
$$\chi\ s = {}^{\mathfrak{h}}s :: {}^{\mathfrak{t}}s \ltimes {}^{\mathfrak{t}}(\chi\ {}^{\mathfrak{t}}s)$$

Suppose χ_1 and χ_2 are solutions.
Ad hoc bisimulation, inductively defined by:

Bisimulation and Unicity of Solutions

Unicity of solutions for the equation:

$$\chi : \mathbb{S} \to \mathbb{S}$$
$$\chi\ s = {}^{\mathfrak{h}}s :: {}^{\mathfrak{t}}s \ltimes {}^{\mathfrak{t}}(\chi\ {}^{\mathfrak{t}}s)$$

Suppose χ_1 and χ_2 are solutions.
Ad hoc bisimulation, inductively defined by:

$$\frac{s : \mathbb{S}}{\chi_1\, s \sim \chi_2\, s}(R0)$$

Bisimulation and Unicity of Solutions

Unicity of solutions for the equation:

$$\begin{aligned} &\chi : \mathbb{S} \to \mathbb{S} \\ &\chi\ s = \mathfrak{h}s :: \mathfrak{t}s \ltimes \mathfrak{t}(\chi\ \mathfrak{t}s) \end{aligned}$$

Suppose χ_1 and χ_2 are solutions.
Ad hoc bisimulation, inductively defined by:

$$\frac{s : \mathbb{S}}{\chi_1\ s \sim \chi_2\ s}(R0) \qquad \frac{s, x_1, x_2 : \mathbb{S} \quad x_1 \sim x_2}{s \ltimes \mathfrak{t}x_1 \sim s \ltimes \mathfrak{t}x_2}(R1)$$

Bisimulation and Unicity of Solutions

Unicity of solutions for the equation:

$$\begin{array}{l} \chi : \mathbb{S} \to \mathbb{S} \\ \chi\ s = {}^{\mathfrak{h}}s :: {}^{\mathfrak{t}}s \ltimes {}^{\mathfrak{t}}(\chi\ {}^{\mathfrak{t}}s) \end{array}$$

Suppose χ_1 and χ_2 are solutions.
Ad hoc bisimulation, inductively defined by:

$$\frac{s : \mathbb{S}}{\chi_1\ s \sim \chi_2\ s}(R0) \qquad \frac{s, x_1, x_2 : \mathbb{S} \quad x_1 \sim x_2}{s \ltimes {}^{\mathfrak{t}}x_1 \sim s \ltimes {}^{\mathfrak{t}}x_2}(R1)$$

$$\frac{s, x_1, x_2 : \mathbb{S} \quad x_1 \sim x_2}{x_1 \ltimes s \sim x_2 \ltimes s}(R2).$$

Bisimulation and Unicity of Solutions

Unicity of solutions for the equation:

$$
\begin{array}{l}
\chi : \mathbb{S} \to \mathbb{S} \\
\chi\ s = {}^{\mathfrak{h}}s :: {}^{\mathfrak{t}}s \ltimes {}^{\mathfrak{t}}(\chi\ {}^{\mathfrak{t}}s)
\end{array}
$$

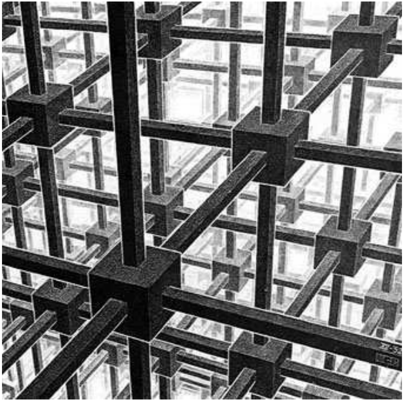
Suppose χ_1 and χ_2 are solutions.
Ad hoc bisimulation, inductively defined by:

$$
\frac{s : \mathbb{S}}{\chi_1\ s \sim \chi_2\ s}(R0) \qquad \frac{s, x_1, x_2 : \mathbb{S} \quad x_1 \sim x_2}{s \ltimes {}^{\mathfrak{t}}x_1 \sim s \ltimes {}^{\mathfrak{t}}x_2}(R1)
$$

$$
\frac{s, x_1, x_2 : \mathbb{S} \quad x_1 \sim x_2}{x_1 \ltimes s \sim x_2 \ltimes s}(R2).
$$

By the coinduction principle and R0, $\chi_1 = \chi_2$.

Constructive Infinity



Brouwer's Continuity Principle

Brouwer's Continuity (In Functional Programming terms):
Given a function $f : \mathbb{S}_{\mathbb{N}} \to \mathbb{N}$,
for every $s : \mathbb{S}_{\mathbb{N}}$, there exists $n : \mathbb{N}$ such that
for every $s' : \mathbb{S}_{\mathbb{N}}$, if $\mathsf{take}\, n\, s' = \mathsf{take}\, n\, s$, then $f\, s' = f\, s$.

Brouwer's Continuity Principle

Brouwer's Continuity (In Functional Programming terms):
Given a function $f : \mathbb{S}_{\mathbb{N}} \to \mathbb{N}$,
for every $s : \mathbb{S}_{\mathbb{N}}$, there exists $n : \mathbb{N}$ such that
for every $s' : \mathbb{S}_{\mathbb{N}}$, if $\mathsf{take}\, n\, s' = \mathsf{take}\, n\, s$, then $f\, s' = f\, s$.

Apparently impossible functional program: [Martin Escardo]

$$\mathsf{allb} : (\mathbb{S}_{\mathbb{B}} \to \mathbb{B}) \to \mathbb{B}$$

Brouwer's Continuity Principle

Brouwer's Continuity (In Functional Programming terms):
Given a function $f : \mathbb{S}_{\mathbb{N}} \to \mathbb{N}$,
for every $s : \mathbb{S}_{\mathbb{N}}$, there exists $n : \mathbb{N}$ such that
for every $s' : \mathbb{S}_{\mathbb{N}}$, if $\mathsf{take}\, n\, s' = \mathsf{take}\, n\, s$, then $f\, s' = f\, s$.

Apparently impossible functional program: [Martin Escardo]

$$\begin{aligned} &\mathsf{allb} : (\mathbb{S}_{\mathbb{B}} \to \mathbb{B}) \to \mathbb{B} \\ &\mathsf{allb}\, f = f\,(\mathsf{counterexample}\, f) \end{aligned}$$

Brouwer's Continuity Principle

Brouwer's Continuity (In Functional Programming terms):
Given a function $f : \mathbb{S}_{\mathbb{N}} \to \mathbb{N}$,
for every $s : \mathbb{S}_{\mathbb{N}}$, there exists $n : \mathbb{N}$ such that
for every $s' : \mathbb{S}_{\mathbb{N}}$, if $\mathsf{take}\, n\, s' = \mathsf{take}\, n\, s$, then $f\, s' = f\, s$.

Apparently impossible functional program: [Martin Escardo]

$$
\begin{aligned}
&\mathsf{allb} : (\mathbb{S}_{\mathbb{B}} \to \mathbb{B}) \to \mathbb{B} \\
&\mathsf{allb}\, f = f\,(\mathsf{counterexample}\, f)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{counterexample} : (\mathbb{S}_{\mathbb{B}} \to \mathbb{B}) \to \mathbb{S}_{\mathbb{B}} \\
&\mathsf{counterexample}\, f = \ \ \mathbf{if}\ (\mathsf{allb}\, f_t) \\
&\qquad\qquad\qquad\qquad\quad \mathbf{then}\ (\mathsf{false} :: \mathsf{counterexample}\, f_f) \\
&\qquad\qquad\qquad\qquad\quad \mathbf{else}\ (\mathsf{true} :: \mathsf{counterexample}\, f_t) \\
&\qquad \mathbf{where}\ \ f_t = \lambda s. f\,(\mathsf{true} :: s) \\
&\qquad\qquad\qquad\ \ f_f = \lambda s. f\,(\mathsf{false} :: s)
\end{aligned}
$$

Tabulation of functions on inductive types

Integers in binary representation: $[\mathbb{B}]$.

Tabulation of functions on inductive types

Integers in binary representation: $[\mathbb{B}]$.
A function $f : [\mathbb{B}] \to A$ can be represented by a tree:

$$\textbf{codata } \mathbb{T}_A : \mathsf{Set}$$
$$\mathsf{node} : A \to \mathbb{T}_A \to \mathbb{T}_A \to \mathbb{T}_A$$

Tabulation of functions on inductive types

Integers in binary representation: $[\mathbb{B}]$.
A function $f : [\mathbb{B}] \to A$ can be represented by a tree:

$$\begin{aligned}&\textbf{codata}\ \mathbb{T}_A : \mathrm{Set}\\ &\quad \mathrm{node} : A \to \mathbb{T}_A \to \mathbb{T}_A \to \mathbb{T}_A\end{aligned}$$

Tabulation of the function:

$$\begin{aligned}&\mathrm{tabulate} : ([\mathbb{B}] \to A) \to \mathbb{T}_A\\ &\mathrm{tabulate}\, f = \mathrm{node}\,(f\,[\,])\,(\mathrm{tabulate}\, f_t)\,(\mathrm{tabulate}\, f_f)\\ &\quad \textbf{where}\quad f_t = \lambda s. f\,(\mathrm{true} :: s)\\ &\qquad\qquad\quad\;\; f_f = \lambda s. f\,(\mathrm{false} :: s)\end{aligned}$$

Tabulation of functions on inductive types

Integers in binary representation: $[\mathbb{B}]$.
A function $f : [\mathbb{B}] \to A$ can be represented by a tree:

$$\begin{array}{l} \textbf{codata } \mathbb{T}_A : \mathsf{Set} \\ \quad \mathsf{node} : A \to \mathbb{T}_A \to \mathbb{T}_A \to \mathbb{T}_A \end{array}$$

Tabulation of the function:

$$\begin{array}{l} \mathsf{tabulate} : ([\mathbb{B}] \to A) \to \mathbb{T}_A \\ \mathsf{tabulate}\, f = \mathsf{node}\,(f\,[])\,(\mathsf{tabulate}\, f_t)\,(\mathsf{tabulate}\, f_f) \\ \quad \textbf{where} \quad f_t = \lambda s. f\,(\mathsf{true} :: s) \\ \qquad\qquad\quad f_f = \lambda s. f\,(\mathsf{false} :: s) \end{array}$$

Application of a tabulation:

$$\begin{array}{l} \mathsf{apply} : \mathbb{T}_A \to [\mathbb{B}] \to A \\ \mathsf{apply}\,(\mathsf{node}\, a\, t_1\, t_2)\,[] = a \\ \mathsf{apply}\,(\mathsf{node}\, a\, t_1\, t_2)\,(\mathsf{true} :: l) = \mathsf{apply}\, t_1\, l \\ \mathsf{apply}\,(\mathsf{node}\, a\, t_1\, t_2)\,(\mathsf{false} :: l) = \mathsf{apply}\, t_2\, l \end{array}$$

Tabulation of functions on coinductive types

A function $f : \mathbb{S}_A \rightarrow B$ can (?) be represented by

Tabulation of functions on coinductive types

A function $f : \mathbb{S}_A \to B$ can (?) be represented by a well-founded tree: [Ghani/Hancock/Pattinson:2006]

$$
\begin{array}{l}
\textbf{data } \mathbb{T}_{A,B} : \mathsf{Set} \\
\quad \mathsf{leaf} : B \to \mathbb{T}_{A,B} \\
\quad \mathsf{node} : (A \to \mathbb{T}_{A,B}) \to \mathbb{T}_{A,B}
\end{array}
$$

Tabulation of functions on coinductive types

A function $f : \mathbb{S}_A \to B$ can (?) be represented by a well-founded tree: [Ghani/Hancock/Pattinson:2006]

$$\begin{array}{l} \textbf{data } \mathbb{T}_{A,B} : \mathsf{Set} \\ \quad \mathsf{leaf} : B \to \mathbb{T}_{A,B} \\ \quad \mathsf{node} : (A \to \mathbb{T}_{A,B}) \to \mathbb{T}_{A,B} \end{array}$$

Application of a tabulation:

$$\begin{array}{l} \mathsf{apply} : \mathbb{T}_{A,B} \to \mathbb{S}_A \to B \\ \mathsf{apply}\,(\mathsf{leaf}\; b)\; s = b \\ \mathsf{apply}\,(\mathsf{node}\; g)\,(a :: s) = \mathsf{apply}\,(g\; a)\; s \end{array}$$

Tabulation of functions on coinductive types

A function $f : \mathbb{S}_A \rightarrow B$ can (?) be represented by a well-founded tree: [Ghani/Hancock/Pattinson:2006]

$$\begin{array}{l} \textbf{data}\ \mathbb{T}_{A,B} : \mathsf{Set} \\ \quad \mathsf{leaf} : B \rightarrow \mathbb{T}_{A,B} \\ \quad \mathsf{node} : (A \rightarrow \mathbb{T}_{A,B}) \rightarrow \mathbb{T}_{A,B} \end{array}$$

Application of a tabulation:

$$\begin{array}{l} \mathsf{apply} : \mathbb{T}_{A,B} \rightarrow \mathbb{S}_A \rightarrow B \\ \mathsf{apply}\,(\mathsf{leaf}\ b)\ s = b \\ \mathsf{apply}\,(\mathsf{node}\ g)\,(a :: s) = \mathsf{apply}\,(g\ a)\ s \end{array}$$

Is there an inverse transformation/tabulation?

$$\mathsf{tabulate} : (\mathbb{S}_A \rightarrow B) \rightarrow \mathbb{T}_{A,B}$$

Surely B must be a discrete/inductive type.

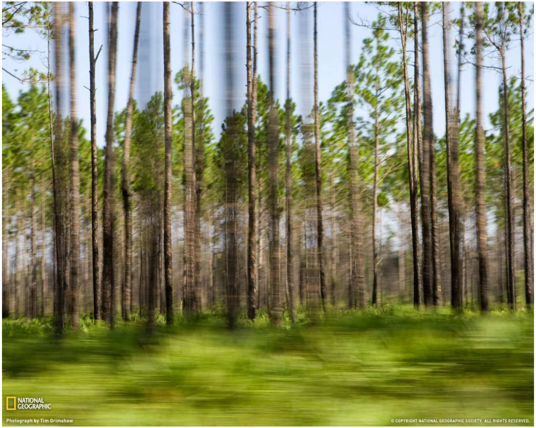
Tabulation Duality?

What is the relation between the two kinds of tabulations?

- Function on Inductive Types : Coinductive Tabulations.
- Function on CoInductive Types : Inductive Tabulations.

How to build a tabulation in the second case?
We need strong intentionality.

General Recursion



Partial Recursion in Type Theory

In Type Theory all functions are total.
How do we represent partial recursive functions?

Partial Recursion in Type Theory

In Type Theory all functions are total.
How do we represent partial recursive functions?
Partiality Monad: [VC 2005]

$$\begin{array}{l}\textbf{codata}\; B^\nu : \mathrm{Set}\\ \quad \lceil - \rceil : B \to B^\nu\\ \quad \lhd : B^\nu \to B^\nu\end{array}$$

Partial Recursion in Type Theory

In Type Theory all functions are total.
How do we represent partial recursive functions?
Partiality Monad: [VC 2005]

$$\begin{array}{c} \textbf{codata}\, B^{\nu} : \mathrm{Set} \\ \lceil - \rceil : B \to B^{\nu} \\ \lhd : B^{\nu} \to B^{\nu} \end{array}$$

A partial function is represented as $f : A \to B^{\nu}$.

Partial Recursion in Type Theory

In Type Theory all functions are total.
How do we represent partial recursive functions?
Partiality Monad: [VC 2005]

$$
\begin{aligned}
&\textbf{codata}\, B^\nu : \mathrm{Set} \\
&\quad \lceil - \rceil : B \to B^\nu \\
&\quad \lhd : B^\nu \to B^\nu
\end{aligned}
$$

A partial function is represented as $f : A \to B^\nu$.
f defined on a: $f\,a = \lhd\lhd\lhd \cdots \lhd\lceil b \rceil$.

Partial Recursion in Type Theory

In Type Theory all functions are total.
How do we represent partial recursive functions?
Partiality Monad: [VC 2005]

$$\begin{array}{c} \textbf{codata}\, B^\nu : \mathrm{Set} \\ \lceil - \rceil : B \rightarrow B^\nu \\ \lhd : B^\nu \rightarrow B^\nu \end{array}$$

A partial function is represented as $f : A \rightarrow B^\nu$.
f defined on a: $f\,a = \lhd\lhd\lhd\cdots\lhd\lceil b\rceil$.
f undefined on a: $f\,a = \lhd\lhd\lhd\lhd\cdots$.

Bove/Capretta method

Different approach to partial recursive functions.[Bove/VC 2001]

Bove/Capretta method

Different approach to partial recursive functions.[Bove/VC 2001]
A partial function $f : A \rightharpoonup B$ is represented by
A domain predicate and a function on the domain:

$$\begin{aligned} &\mathsf{Dom} : A \to \mathsf{Prop} \\ &f : (a : A)\mathsf{Dom}\, a \to B \end{aligned}$$

Bove/Capretta method

Different approach to partial recursive functions.[Bove/VC 2001]
A partial function $f : A \rightharpoonup B$ is represented by
A domain predicate and a function on the domain:

$$\mathsf{Dom} : A \to \mathsf{Prop}$$
$$f : (a : A)\mathsf{Dom}\, a \to B$$

Example, function that seeks a 0 in a stream:

$$\mathsf{seek} : \mathbb{S}_{\mathbb{N}} \rightharpoonup \mathbb{N}$$

Bove/Capretta method

Different approach to partial recursive functions.[Bove/VC 2001]
A partial function $f : A \rightharpoonup B$ is represented by
A domain predicate and a function on the domain:

$$\mathrm{Dom} : A \to \mathrm{Prop}$$
$$f : (a : A)\mathrm{Dom}\, a \to B$$

Example, function that seeks a 0 in a stream:

$$\mathrm{seek} : \mathbb{S}_{\mathbb{N}} \rightharpoonup \mathbb{N}$$

$$\textbf{data}\ \mathrm{Dom}_{\mathrm{seek}} : \mathbb{S}_{\mathbb{N}} \to \mathrm{Prop}$$
$$\mathrm{seek} : (s : \mathbb{S}_{\mathbb{N}})\mathrm{Dom}_{\mathrm{seek}}\, s \to \mathbb{N}$$

Bove/Capretta method

Different approach to partial recursive functions.[Bove/VC 2001]
A partial function $f : A \rightharpoonup B$ is represented by
A domain predicate and a function on the domain:

$$\mathsf{Dom} : A \to \mathsf{Prop}$$
$$f : (a : A)\mathsf{Dom}\, a \to B$$

Example, function that seeks a 0 in a stream:

$$\mathsf{seek} : \mathbb{S}_{\mathbb{N}} \rightharpoonup \mathbb{N}$$

$\textbf{data}\ \mathsf{Dom}_{\mathsf{seek}} : \mathbb{S}_{\mathbb{N}} \to \mathsf{Prop}$
$\quad \mathsf{found} : (s : \mathbb{S}_{\mathbb{N}})\mathsf{Dom}_{\mathsf{seek}}\,(0 :: s)$
$\quad \mathsf{notfound} : (n : \mathbb{N})(s : \mathbb{S}_{\mathbb{N}})\mathsf{Dom}_{\mathsf{seek}}\, s \to \mathsf{Dom}_{\mathsf{seek}}\,(\mathsf{S}\, n :: s)$

$\mathsf{seek} : (s : \mathbb{S}_{\mathbb{N}})\mathsf{Dom}_{\mathsf{seek}}\, s \to \mathbb{N}$
$\mathsf{seek}\,(0 :: s)\,(\mathsf{found}\, z) = 0$
$\mathsf{seek}\,(\mathsf{S}\, n :: s)\,(\mathsf{notfound}\, n\, s\, h) = \mathsf{S}\,(\mathsf{seek}\, s\, h)$

Computation by Prophecy

Coinductive version of the domain:
trace of computation. [Bove/VC 2007]

Computation by Prophecy

Coinductive version of the domain:
trace of computation. [Bove/VC 2007]

$$
\begin{array}{l}
\textbf{codata}\ \mathsf{Trace}_{\mathsf{seek}} : \mathbb{S}_{\mathbb{N}} \to \mathsf{Prop} \\
\quad \mathsf{found} : (s : \mathbb{S}_{\mathbb{N}})\mathsf{Trace}_{\mathsf{seek}}\,(0 :: s) \\
\quad \mathsf{notfound} : (n : \mathbb{N})(s : \mathbb{S}_{\mathbb{N}})\mathsf{Trace}_{\mathsf{seek}}\,s \to \mathsf{Trace}_{\mathsf{seek}}\,(\mathsf{S}\,n :: s)
\end{array}
$$

$$
\mathsf{seek} : (s : \mathbb{S}_{\mathbb{N}})\mathsf{Trace}_{\mathsf{seek}}\,s \to \mathbb{N}^{\nu}
$$

Computation by Prophecy

Coinductive version of the domain:
trace of computation. [Bove/VC 2007]

$$
\begin{array}{l}
\textbf{codata}\ \mathsf{Trace}_{\mathsf{seek}} : \mathbb{S}_{\mathbb{N}} \to \mathsf{Prop} \\
\quad \mathsf{found} : (s : \mathbb{S}_{\mathbb{N}})\mathsf{Trace}_{\mathsf{seek}}\,(0 :: s) \\
\quad \mathsf{notfound} : (n : \mathbb{N})(s : \mathbb{S}_{\mathbb{N}})\mathsf{Trace}_{\mathsf{seek}}\,s \to \mathsf{Trace}_{\mathsf{seek}}\,(\mathsf{S}\,n :: s)
\end{array}
$$

$$\mathsf{seek} : (s : \mathbb{S}_{\mathbb{N}})\mathsf{Trace}_{\mathsf{seek}}\,s \to \mathbb{N}^{\nu}$$

In all these representations:
How do we effectively compute the function?
Coinductive objects don't automatically unfold.
Domain predicate must be proved.

Computation by Judgement Rewriting

To compute $\mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)$
Assume the domain predicate and type-check the result:

$$h : \mathsf{Dom}_{\mathsf{seek}}\,(7 :: 2 :: 5 :: \cdots) \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\; h : \mathbb{N}$$

Computation by Judgement Rewriting

To compute $\mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)$
Assume the domain predicate and type-check the result:

$$h : \mathsf{Dom}_{\mathsf{seek}}\,(7 :: 2 :: 5 :: \cdots) \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\,h : \mathbb{N}$$

Inversion: h must have the form $(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\,h_1)$.
Rewrite the judgement:

Computation by Judgement Rewriting

To compute $\mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)$
Assume the domain predicate and type-check the result:

$$h : \mathsf{Dom}_{\mathsf{seek}}\,(7 :: 2 :: 5 :: \cdots) \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\,h : \mathbb{N}$$

Inversion: h must have the form $(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\,h_1)$.
Rewrite the judgement:

$$\mapsto \quad h_1 : \mathsf{Dom}_{\mathsf{seek}}\,(2 :: 5 :: \cdots) \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\,(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\,h_1) : \mathbb{N}$$

Computation by Judgement Rewriting

To compute $\mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)$
Assume the domain predicate and type-check the result:

$$h : \mathsf{Dom}_{\mathsf{seek}}\,(7 :: 2 :: 5 :: \cdots) \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\; h : \mathbb{N}$$

Inversion: h must have the form $(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\; h_1)$.
Rewrite the judgement:

$$\begin{array}{ll} \mapsto & h_1 : \mathsf{Dom}_{\mathsf{seek}}\,(2 :: 5 :: \cdots) \\ & \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\,(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\; h_1) : \mathbb{N} \\ \rightsquigarrow & h_1 : \mathsf{Dom}_{\mathsf{seek}}\,(2 :: 5 :: \cdots) \vdash \mathsf{S}\,\mathsf{seek}\,(2 :: 5 :: \cdots)\;\; h_1 : \mathbb{N} \end{array}$$

Computation by Judgement Rewriting

To compute $\mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)$
Assume the domain predicate and type-check the result:

$$h : \mathsf{Dom}_{\mathsf{seek}}\,(7 :: 2 :: 5 :: \cdots) \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\,h : \mathbb{N}$$

Inversion: h must have the form $(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\,h_1)$.
Rewrite the judgement:

$$\begin{array}{ll}
\mapsto & h_1 : \mathsf{Dom}_{\mathsf{seek}}\,(2 :: 5 :: \cdots) \\
& \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\,(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\,h_1) : \mathbb{N} \\
\rightsquigarrow & h_1 : \mathsf{Dom}_{\mathsf{seek}}\,(2 :: 5 :: \cdots) \vdash \mathsf{S}\,\mathsf{seek}\,(2 :: 5 :: \cdots)\;h_1 : \mathbb{N} \\
\mapsto & h_2 : \mathsf{Dom}_{\mathsf{seek}}\,(5 :: \cdots) \vdash \mathsf{S}\,\mathsf{S}\,\mathsf{seek}\,(5 :: \cdots)\;h_2 : \mathbb{N}
\end{array}$$

Computation by Judgement Rewriting

To compute $\mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)$
Assume the domain predicate and type-check the result:

$$h : \mathsf{Dom}_{\mathsf{seek}}\,(7 :: 2 :: 5 :: \cdots) \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\,h : \mathbb{N}$$

Inversion: h must have the form $(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\,h_1)$.
Rewrite the judgement:

$$\begin{array}{ll} \mapsto & h_1 : \mathsf{Dom}_{\mathsf{seek}}\,(2 :: 5 :: \cdots) \\ & \vdash \mathsf{seek}\,(7 :: 2 :: 5 :: \cdots)\,(\mathsf{notfound}\,6\,(2 :: 5 :: \cdots)\,h_1) : \mathbb{N} \\ \rightsquigarrow & h_1 : \mathsf{Dom}_{\mathsf{seek}}\,(2 :: 5 :: \cdots) \vdash \mathsf{S}\,\mathsf{seek}\,(2 :: 5 :: \cdots)\;h_1 : \mathbb{N} \\ \mapsto & h_2 : \mathsf{Dom}_{\mathsf{seek}}\,(5 :: \cdots) \vdash \mathsf{S}\,\mathsf{S}\,\mathsf{seek}\,(5 :: \cdots)\;h_2 : \mathbb{N} \end{array}$$

Connection with Non-Standard Type Theory [Martin-Löf 1988]

Mixing Induction and Coinduction

Mixed Inductive/Coinductive Types

Data structures with both: [Danielsson/Altenkirch 2009]

- Constructors that must be well-founded;
- Constructors that can be iterated infinitely

Example, equality for the partiality monad:

$$
\begin{aligned}
&\textbf{codata}\ (\simeq) : B^{\nu} \to B^{\nu} \to \mathrm{Prop} \\
&\quad \mathrm{eqstep} : (x_1, x_2 : B^{\nu}) x_1 \simeq x_2 \to \lhd x_1 \simeq \lhd x_2 \\
&\quad \mathrm{eqval} : (b : B) \lceil b \rceil \simeq \lceil b \rceil \\
&\quad \mathrm{eqmix}_1 : (x_1, x_2 : B^{\nu}) x_1 \simeq x_2 \to \lhd x_1 \simeq x_2 \\
&\quad \mathrm{eqmix}_2 : (x_1, x_2 : B^{\nu}) x_1 \simeq x_2 \to x_1 \simeq \lhd x_2
\end{aligned}
$$

Mixed Inductive/Coinductive Types

Data structures with both: [Danielsson/Altenkirch 2009]

- Constructors that must be well-founded;
- Constructors that can be iterated infinitely

Example, equality for the partiality monad:

$$\begin{array}{l}\textbf{codata}\ (\simeq) : B^\nu \to B^\nu \to \mathrm{Prop} \\ \quad \mathrm{eqstep} : (x_1, x_2 : B^\nu) x_1 \simeq x_2 \to \lhd x_1 \simeq \lhd x_2 \\ \quad \mathrm{eqval} : (x_1, x_2 : B^\nu)(b : B) x_1 \downarrow b \to x_2 \downarrow b \to x_1 \simeq x_2\end{array}$$

$x \downarrow b$ inductive convergence relation

Mixed Inductive/Coinductive Types

Data structures with both: [Danielsson/Altenkirch 2009]

- Constructors that must be well-founded;
- Constructors that can be iterated infinitely

Example, equality for the partiality monad:

$$\begin{array}{l} \textbf{data}\ (\simeq) : B^\nu \to B^\nu \to \mathsf{Prop} \\ \quad \mathsf{eqstep} : (x_1, x_2 : B^\nu) x_1 \overset{\infty}{\simeq} x_2 \to \lhd x_1 \simeq \lhd x_2 \\ \quad \mathsf{eqval} : (b : B) \lceil b \rceil \simeq \lceil b \rceil \\ \quad \mathsf{eqmix}_1 : (x_1, x_2 : B^\nu) x_1 \simeq x_2 \to \lhd x_1 \simeq x_2 \\ \quad \mathsf{eqmix}_2 : (x_1, x_2 : B^\nu) x_1 \simeq x_2 \to x_1 \simeq \lhd x_2 \end{array}$$

The ∞ marks arguments that need not be well-founded

Conclusion

Interesting topics for future research:

Conclusion

Interesting topics for future research:

- Solution of corecursive equations;

Conclusion

Interesting topics for future research:

- Solution of corecursive equations;
- Tabulations of functions on coinductive domains;

Conclusion

Interesting topics for future research:

- Solution of corecursive equations;
- Tabulations of functions on coinductive domains;
- General recursion and non-standard type theory;

Conclusion

Interesting topics for future research:

- Solution of corecursive equations;
- Tabulations of functions on coinductive domains;
- General recursion and non-standard type theory;
- Mixed inductive/coinductive definitions.

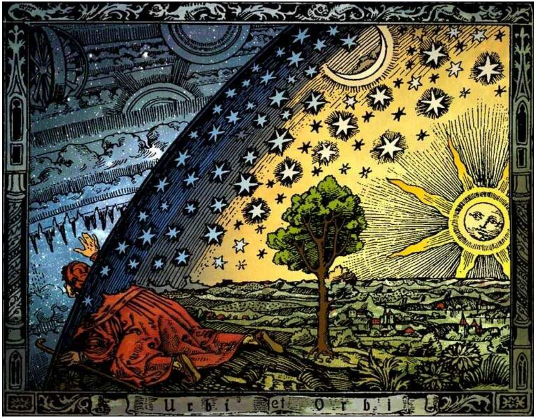
Conclusion

Interesting topics for future research:

- Solution of corecursive equations;
- Tabulations of functions on coinductive domains;
- General recursion and non-standard type theory;
- Mixed inductive/coinductive definitions.

There are many other exiting topics.

MSFP



Mathematically Structured Functional Programming

Baltimore, 25 September 2010

http://cs.ioc.ee/msfp/msfp2010/

Deadline: 9 - 16 April