A Computational Understanding of Classical (Co)Recursion

Paul Downen and Zena M. Ariola

PPDP 2020, September 8–10

Topic

Topic

- Both programs and proofs with loops

Topic

- Both programs and proofs with loops
- (Co)Recursion and (Co)Induction

Topic

- Both programs and proofs with loops
- (Co)Recursion and (Co)Induction
- “Terminating” or “Productive”

Topic

- Both programs and proofs with loops
- (Co)Recursion and (Co)Induction
- “Terminating” or “Productive”
- Extend to non-termination, effects

Methodology

Methodology

- Duality

Methodology

- Duality
- Computational

Methodology

- Duality
- Computational
- Curry-Howard

Methodology

- Duality
- Computational
- Curry-Howard
    - sequent calculus as abstract machines

Methodology

- Duality
- Computational
- Curry-Howard
  - sequent calculus as abstract machines
- Classical

Recursive Programs

Recursion on Natural Numbers

In System T

Recursion on Natural Numbers

In System T

- Simply-typed λ-calculus plus inductive $Nat = Zero \mid Succ\ Nat$

Recursion on Natural Numbers

In System T

- Simply-typed λ-calculus plus inductive $Nat = Zero \mid Succ\ Nat$

$$\mathbf{rec}^{A}_{Nat} : Nat \to A \to (Nat \to A \to A) \to A$$

Recursion on Natural Numbers

In System T

- Simply-typed λ-calculus plus inductive $Nat = Zero \mid Succ\ Nat$

$$\mathbf{rec}^{A}_{Nat} : Nat \to A \to (Nat \to A \to A) \to A$$

$\mathbf{rec}\ M\ \mathbf{as}\ \{Zero \to N \mid Succ\ x \to y\,.\,P\}$ where $M, x : Nat$; N and $P, y : A$

Recursion on Natural Numbers

In System T

- Simply-typed λ-calculus plus inductive $Nat = Zero \mid Succ\ Nat$

$$\mathbf{rec}^{A}_{Nat} : Nat \to A \to (Nat \to A \to A) \to A$$

$$\mathbf{rec}\ M\ \mathbf{as}\ \{Zero \to N \mid Succ\ x \to y\,.\,P\} \qquad \text{where } M, x : Nat; N \text{ and } P, y : A$$

$$\begin{array}{l} \mathbf{case}\ M\ \mathbf{of}\ Zero \to N \\ \qquad\qquad\qquad Succ\ x \to P \end{array} := \begin{array}{l} \mathbf{rec}\ M\ \mathbf{of}\ Zero \to M \\ \qquad\qquad\quad Succ\ x \to \_\,.\,P \end{array}$$

Recursion on Natural Numbers

In System T

- Simply-typed λ-calculus plus inductive $Nat = Zero \mid Succ\ Nat$

$$\mathbf{rec}^{A}_{Nat} : Nat \to A \to (Nat \to A \to A) \to A$$

$$\mathbf{rec}\ M\ \mathbf{as}\ \{Zero \to N \mid Succ\ x \to y\,.\,P\} \qquad \text{where } M, x : Nat; N \text{ and } P, y : A$$

$$\begin{aligned}\mathbf{case}\ M\ \mathbf{of}\ Zero &\to N \\ Succ\ x &\to P\end{aligned} \;:=\; \begin{aligned}\mathbf{rec}\ M\ \mathbf{of}\ Zero &\to M \\ Succ\ x &\to \_\,.\,P\end{aligned}$$

$$\begin{aligned}\mathbf{iter}\ M\ \mathbf{as}\ Zero &\to N \\ Succ &\to y\,.\,P\end{aligned} \;:=\; \begin{aligned}\mathbf{rec}\ M\ \mathbf{as}\ Zero &\to N \\ Succ\ \_ &\to y\,.\,P\end{aligned}$$

Examples of Recursion

In System T

Examples of Recursion

In System T

$$
\begin{aligned}
plus\ Zero\ y &= y \\
plus\ (Succ\ x')\ y &= Succ\ (plus\ x'\ y)
\end{aligned}
$$

Examples of Recursion

In System T

$$
\begin{aligned}
\mathit{plus}\ \mathit{Zero}\ y &= y \\
\mathit{plus}\ (\mathit{Succ}\ x')\ y &= \mathit{Succ}\ (\mathit{plus}\ x'\ y)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{plus} = \lambda x\,.\,\lambda y\,.\,\mathbf{iter}\ x\ \mathbf{as}& \\
\mathit{Zero} &\to y \\
\mathit{Succ} &\to z\,.\,\mathit{Succ}\ z
\end{aligned}
$$

Examples of Recursion

In System T

$$\begin{aligned} plus\ Zero\ y &= y \\ plus\ (Succ\ x')\ y &= Succ\ (plus\ x'\ y) \end{aligned}$$

$$\begin{aligned} plus = \lambda x\,.\,\lambda y\,.\,\mathbf{iter}\ x\ \mathbf{as} \\ Zero &\to y \\ Succ &\to z\,.\,Succ\ z \end{aligned}$$

$$\begin{aligned} pred\ Zero &= Zero \\ pred\ (Succ\ x') &= x' \end{aligned}$$

Examples of Recursion

In System T

$$\begin{aligned} plus\ Zero\ y &= y \\ plus\ (Succ\ x')\ y &= Succ\ (plus\ x'\ y) \end{aligned}$$

$$\begin{aligned} plus = \lambda x\,.\,\lambda y\,.\,&\mathbf{iter}\ x\ \mathbf{as} \\ &\quad Zero \to y \\ &\quad Succ \to z\,.\,Succ\ z \end{aligned}$$

$$\begin{aligned} pred\ Zero &= Zero \\ pred\ (Succ\ x') &= x' \end{aligned}$$

$$\begin{aligned} pred = \lambda x\,.\,&\mathbf{case}\ x\ \mathbf{of} \\ &\quad Zero \to Zero \\ &\quad Succ\ x' \to x' \end{aligned}$$

Examples of Recursion

In System T

$$\begin{aligned} plus\ Zero\ y &= y \\ plus\ (Succ\ x')\ y &= Succ\ (plus\ x'\ y) \end{aligned}$$

$$\begin{aligned} plus = \lambda x\,.\,\lambda y\,.\,&\textbf{iter}\ x\ \textbf{as} \\ &Zero \to y \\ &Succ \to z\,.\,Succ\ z \end{aligned}$$

$$\begin{aligned} pred\ Zero &= Zero \\ pred\ (Succ\ x') &= x' \end{aligned}$$

$$\begin{aligned} pred = \lambda x\,.\,&\textbf{case}\ x\ \textbf{of} \\ &Zero \to Zero \\ &Succ\ x' \to x' \end{aligned}$$

$$\begin{aligned} minus\ x\ Zero &= x \\ minus\ x\ (Succ\ y') &= pred\ (minus\ x\ y') \end{aligned}$$

Examples of Recursion

In System T

$$plus\ Zero\ y = y$$
$$plus\ (Succ\ x')\ y = Succ\ (plus\ x'\ y)$$

$$plus = \lambda x\,.\,\lambda y\,.\,\mathbf{iter}\ x\ \mathbf{as}$$
$$Zero \to y$$
$$Succ \to z\,.\,Succ\ z$$

$$pred\ Zero = Zero$$
$$pred\ (Succ\ x') = x'$$

$$pred = \lambda x\,.\,\mathbf{case}\ x\ \mathbf{of}$$
$$Zero \to Zero$$
$$Succ\ x' \to x'$$

$$minus\ x\ Zero = x$$
$$minus\ x\ (Succ\ y') = pred\ (minus\ x\ y')$$

$$minus = \lambda x\,.\,\lambda y\,.\,\mathbf{iter}\ y\ \mathbf{as}$$
$$Zero \to x$$
$$Succ \to z\,.\,pred\ z$$

Recursion vs Iteration

Expressiveness vs Cost

Recursion vs Iteration

Expressiveness vs Cost

$$\begin{array}{l}\textbf{iter}\ M\ \textbf{as}\ Zero \to N \\ \qquad\qquad\quad Succ \to y\,.\,P\end{array} := \begin{array}{l}\textbf{rec}\ M\ \textbf{as}\ Zero \to N \\ \qquad\qquad\quad Succ\ \_ \to y\,.\,P\end{array}$$

Recursion vs Iteration

Expressiveness vs Cost

$$\begin{aligned}\textbf{iter}\ M\ \textbf{as}\ &Zero \to N \\ &Succ \to y\,.\,P\end{aligned} := \begin{aligned}\textbf{rec}\ M\ \textbf{as}\ &Zero \to N \\ &Succ\ \_ \to y\,.\,P\end{aligned}$$

$$\begin{aligned}\textbf{rec}\ M\ \textbf{as}\ &Zero \to N \\ &Succ\ x \to y\,.\,P\end{aligned} := \begin{aligned}snd(\textbf{iter}\ M\ \textbf{as}\ &Zero \to (Zero, N) \\ &Succ \to (x, y)\,.\,(Succ\ x, P))\end{aligned}$$

Recursion vs Iteration

Expressiveness vs Cost

$$\begin{aligned}\textbf{iter}\ M\ \textbf{as}\ \mathit{Zero} &\rightarrow N \\ \mathit{Succ} &\rightarrow y\,.\,P\end{aligned} \;:=\; \begin{aligned}\textbf{rec}\ M\ \textbf{as}\ \mathit{Zero} &\rightarrow N \\ \mathit{Succ}\ \_ &\rightarrow y\,.\,P\end{aligned}$$

$$\begin{aligned}\textbf{rec}\ M\ \textbf{as}\ \mathit{Zero} &\rightarrow N \\ \mathit{Succ}\ x &\rightarrow y\,.\,P\end{aligned} \;:=\; \begin{aligned}\mathit{snd}(\textbf{iter}\ M\ \textbf{as}\ \mathit{Zero} &\rightarrow (\mathit{Zero}, N) \\ \mathit{Succ} &\rightarrow (x, y)\,.\,(\mathit{Succ}\ x, P))\end{aligned}$$

- $\mathit{pred}\ (\mathit{Succ}^n\ \mathit{Zero})$ goes from $O(1)$ to $O(n)$ time

Recursion vs Iteration

Expressiveness vs Cost

$$\begin{array}{r}\textbf{iter}\ M\ \textbf{as}\ Zero \to N \\ Succ \to y\,.\,P\end{array} := \begin{array}{r}\textbf{rec}\ M\ \textbf{as}\ Zero \to N \\ Succ\ \_ \to y\,.\,P\end{array}$$

$$\begin{array}{r}\textbf{rec}\ M\ \textbf{as}\ Zero \to N \\ Succ\ x \to y\,.\,P\end{array} := \begin{array}{r}snd(\textbf{iter}\ M\ \textbf{as}\ Zero \to (Zero, N) \\ Succ \to (x, y)\,.\,(Succ\ x, P))\end{array}$$

- $pred\ (Succ^n\ Zero)$ goes from $O(1)$ to $O(n)$ time
- $minus\ (Succ^n\ Zero)\ (Succ^m\ Zero)$ goes from $O(n)$ to $O(n^2 + nm)$

Recursion vs Iteration

Expressiveness vs Cost

$$\begin{array}{r}\textbf{iter}\ M\ \textbf{as}\ Zero \to N \\ Succ \to y\,.\,P\end{array} := \begin{array}{r}\textbf{rec}\ M\ \textbf{as}\ Zero \to N \\ Succ\ \_ \to y\,.\,P\end{array}$$

$$\begin{array}{r}\textbf{rec}\ M\ \textbf{as}\ Zero \to N \\ Succ\ x \to y\,.\,P\end{array} := \begin{array}{r}snd(\textbf{iter}\ M\ \textbf{as}\ Zero \to (Zero, N) \\ Succ \to (x, y)\,.\,(Succ\ x, P))\end{array}$$

- $pred\ (Succ^n\ Zero)$ goes from $O(1)$ to $O(n)$ time
- $minus\ (Succ^n\ Zero)\ (Succ^m\ Zero)$ goes from $O(n)$ to $O(n^2 + nm)$
- Native **rec** has the same performance penalty as encoding in CBV

Recursion vs Iteration

Expressiveness vs Cost

$$\begin{array}{rl}\textbf{iter}\ M\ \textbf{as}\ \mathit{Zero} \to N & \\ \mathit{Succ} \to y\,.\,P\end{array} := \begin{array}{rl}\textbf{rec}\ M\ \textbf{as}\ \mathit{Zero} \to N & \\ \mathit{Succ}\ \_ \to y\,.\,P\end{array}$$

$$\begin{array}{rl}\textbf{rec}\ M\ \textbf{as}\ \mathit{Zero} \to N & \\ \mathit{Succ}\ x \to y\,.\,P\end{array} := \begin{array}{rl}\mathit{snd}(\textbf{iter}\ M\ \textbf{as}\ \mathit{Zero} \to (\mathit{Zero}, N) & \\ \mathit{Succ} \to (x, y)\,.\,(\mathit{Succ}\ x, P))\end{array}$$

- $\mathit{pred}\ (\mathit{Succ}^n\ \mathit{Zero})$ goes from $O(1)$ to $O(n)$ time
- $\mathit{minus}\ (\mathit{Succ}^n\ \mathit{Zero})\ (\mathit{Succ}^m\ \mathit{Zero})$ goes from $O(n)$ to $O(n^2 + nm)$
- Native **rec** has the same performance penalty as encoding in CBV
  - Recursive result always computed; full traversal is mandatory

Recursion in an Abstract Machine

Building the Recursive Continuation

Recursion in an Abstract Machine

Building the Recursive Continuation

$$\langle M \parallel E \rangle$$

Recursion in an Abstract Machine

Building the Recursive Continuation

$$\langle M \,\|\, E \rangle$$

$$\langle M\ N \,\|\, E \rangle \mapsto \langle M \,\|\, N \cdot E \rangle$$
$$\langle \lambda x\,.\, M \,\|\, N \cdot E \rangle \mapsto \langle M[N/x] \,\|\, E \rangle$$

Recursion in an Abstract Machine

Building the Recursive Continuation

$$\langle M \parallel E \rangle$$

$$\langle M\ N \parallel E \rangle \mapsto \langle M \parallel N \cdot E \rangle$$
$$\langle \lambda x \,.\, M \parallel N \cdot E \rangle \mapsto \langle M[N/x] \parallel E \rangle$$

$$ralt := \{\ Zero \to N \mid Succ\ x \to y \,.\, P\ \}$$

Recursion in an Abstract Machine

Building the Recursive Continuation

$$\langle M \parallel E \rangle$$

$$\langle M\ N \parallel E \rangle \mapsto \langle M \parallel N \cdot E \rangle$$
$$\langle \lambda x\,.\,M \parallel N \cdot E \rangle \mapsto \langle M[N/x] \parallel E \rangle$$

$$ralt := \{\ Zero \to N \mid Succ\ x \to y\,.\,P\ \}$$

$$\langle \mathbf{rec}\ M\ \mathbf{as}\ ralt \parallel E \rangle \mapsto \langle M \parallel \mathbf{rec}\ ralt\ \mathbf{with}\ E \rangle$$

Recursion in an Abstract Machine

Building the Recursive Continuation

$$\langle M \,\|\, E \rangle$$

$$\langle M\ N \,\|\, E \rangle \mapsto \langle M \,\|\, N \cdot E \rangle$$
$$\langle \lambda x \,.\, M \,\|\, N \cdot E \rangle \mapsto \langle M[N/x] \,\|\, E \rangle$$

$$\mathit{ralt} := \{\ \mathit{Zero} \to N \mid \mathit{Succ}\ x \to y \,.\, P\ \}$$

$$\langle \mathbf{rec}\ M\ \mathbf{as}\ \mathit{ralt} \,\|\, E \rangle \mapsto \langle M \,\|\, \mathbf{rec}\ \mathit{ralt}\ \mathbf{with}\ E \rangle$$
$$\langle \mathit{Zero} \,\|\, \mathbf{rec}\ \mathit{ralt}\ \mathbf{with}\ E \rangle \mapsto \langle N \,\|\, E \rangle$$

Recursion in an Abstract Machine

Building the Recursive Continuation

$$\langle M \,\|\, E \rangle$$

$$\langle M\ N \,\|\, E \rangle \mapsto \langle M \,\|\, N \cdot E \rangle$$

$$\langle \lambda x \,.\, M \,\|\, N \cdot E \rangle \mapsto \langle M[N/x] \,\|\, E \rangle$$

$$ralt := \{\ Zero \to N \mid Succ\ x \to y \,.\, P\ \}$$

$$\langle \mathbf{rec}\ M\ \mathbf{as}\ ralt \,\|\, E \rangle \mapsto \langle M \,\|\, \mathbf{rec}\ ralt\ \mathbf{with}\ E \rangle$$

$$\langle Zero \,\|\, \mathbf{rec}\ ralt\ \mathbf{with}\ E \rangle \mapsto \langle N \,\|\, E \rangle$$

$$\langle Succ\ M \,\|\, \mathbf{rec}\ ralt\ \mathbf{with}\ E \rangle \mapsto \langle P[M/x, \mathbf{rec}\ M\ \mathbf{as}\ ralt/y] \,\|\, E \rangle$$

Corecursive Programs

What's the Dual of Natural Numbers?

$Nat^{\perp}$

What's the Dual of Natural Numbers?

$Nat^{\perp}$

- $Zero : 1 \to Nat$ is dual to $Run : Nat^{\perp} \to \perp$

What's the Dual of Natural Numbers?

$Nat^{\perp}$

- $Zero : 1 \to Nat$ is dual to $Run : Nat^{\perp} \to \perp$
- $Succ : Nat \to Nat$ is dual to $Tail : Nat^{\perp} \to Nat^{\perp}$

What's the Dual of Natural Numbers?

$Nat^{\perp}$

- $Zero : 1 \to Nat$ is dual to $Run : Nat^{\perp} \to \perp$
- $Succ : Nat \to Nat$ is dual to $Tail : Nat^{\perp} \to Nat^{\perp}$
- $Nat^{\perp}$ is an infinite stream of computations

What's the Dual of Natural Numbers?

$Nat^{\perp}$

- $Zero : 1 \to Nat$ is dual to $Run : Nat^{\perp} \to \perp$
- $Succ : Nat \to Nat$ is dual to $Tail : Nat^{\perp} \to Nat^{\perp}$
- $Nat^{\perp}$ is an infinite stream of computations
  - You can run them, but they don't return

What's the Dual of Natural Numbers?

$Nat^{\perp}$

- $Zero : 1 \to Nat$ is dual to $Run : Nat^{\perp} \to \perp$
- $Succ : Nat \to Nat$ is dual to $Tail : Nat^{\perp} \to Nat^{\perp}$
- $Nat^{\perp}$ is an infinite stream of computations
  - You can run them, but they don't return

Nat Values: $Zero$ $Succ\ V$

Nat Continuation: $\mathbf{rec}\ \{Zero \to N \mid Succ\ x \to y\,.\,P\}\ \mathbf{with}\ E$

What's the Dual of Natural Numbers?

$Nat^{\perp}$

- $Zero : 1 \to Nat$ is dual to $Run : Nat^{\perp} \to \perp$
- $Succ : Nat \to Nat$ is dual to $Tail : Nat^{\perp} \to Nat^{\perp}$
- $Nat^{\perp}$ is an infinite stream of computations
  - You can run them, but they don't return

Nat Values: $Zero$ $Succ\ V$

Nat Continuation: $\mathbf{rec}\ \{Zero \to N \mid Succ\ x \to y\,.\,P\}\ \mathbf{with}\ E$

$Nat^{\perp}$ Value: $\mathbf{corec}\ \{Run \to E \mid Tail\ \alpha \to \beta\,.\,F\}\ \mathbf{with}\ V$

$Nat^{\perp}$ Continuations: Run $Tail\ E$

Corecursion on Streams

In an Abstract Machine

Corecursion on Streams

In an Abstract Machine

- Generalize $Nat^{\perp}$ to $Stream\ A$

Corecursion on Streams

In an Abstract Machine

- Generalize $Nat^{\perp}$ to $Stream\ A$
  - Infinite stream of computations that return an A

Corecursion on Streams

In an Abstract Machine

- Generalize $Nat^{\perp}$ to $Stream\ A$
  - Infinite stream of computations that return an A
  - $Head : Stream\ A \to A$ and $Tail : Stream\ A \to Stream\ A$

Corecursion on Streams

In an Abstract Machine

- Generalize $Nat^{\perp}$ to $\mathit{Stream}\ A$
  - Infinite stream of computations that return an A
  - $\mathit{Head} : \mathit{Stream}\ A \to A$ and $\mathit{Tail} : \mathit{Stream}\ A \to \mathit{Stream}\ A$

$Nat^{\perp}$ Value: $\mathbf{corec}\ \{\mathit{Run} \to E \mid \mathit{Tail}\ \beta \to \gamma\,.\,F\}\ \mathbf{with}\ V$

$Nat^{\perp}$ Conts.: Run $\quad$ $\mathit{Tail}\ E$

Corecursion on Streams

In an Abstract Machine

- Generalize $Nat^{\perp}$ to $Stream\ A$
  - Infinite stream of computations that return an A
  - $Head : Stream\ A \to A$ and $Tail : Stream\ A \to Stream\ A$

$Nat^{\perp}$ Value: $\mathbf{corec}\ \{Run \to E \mid Tail\ \beta \to \gamma\,.\,F\}\ \mathbf{with}\ V$

$Nat^{\perp}$ Conts.: Run $Tail\ E$

$Stream\ A$ Value: $\mathbf{corec}\ \{Head\ \alpha \to E \mid Tail\ \beta \to \gamma\,.\,F\}\ \mathbf{with}\ V$

$Stream\ A$ Conts.: $Head\ E$ $Tail\ E$

Corecursion on Streams

In the λμ-Calculus

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don’t mention continuations directly; implicit “evaluation contexts”

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don't mention continuations directly; implicit "evaluation contexts"
  - Contexts named by $\mu\alpha \, . \, J$; invoked by jumps $\langle M \parallel \alpha \rangle$

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don't mention continuations directly; implicit "evaluation contexts"
  - Contexts named by $\mu\alpha\,.\,J$; invoked by jumps $\langle M \,\|\, \alpha \rangle$

Destructors: $\mathit{Head}\ M : A \qquad \mathit{Tail}\ M : \mathit{Stream}\ A$ when $M : \mathit{Stream}\ A$

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don't mention continuations directly; implicit "evaluation contexts"
  - Contexts named by $\mu\alpha\,.\,J$; invoked by jumps $\langle M \,\|\, \alpha \rangle$

Destructors: $\mathit{Head}\ M : A \qquad \mathit{Tail}\ M : \mathit{Stream}\ A$ when $M : \mathit{Stream}\ A$

Generator: $\mathbf{corec}\ \{\mathit{Head} \to x\,.\,N \mid \mathit{Tail}\ \beta \to y\,.\,P\}\ \mathbf{with}\ M$

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don't mention continuations directly; implicit "evaluation contexts"
  - Contexts named by $\mu\alpha\,.\,J$; invoked by jumps $\langle M \| \alpha \rangle$

Destructors: $\mathit{Head}\ M : A \qquad \mathit{Tail}\ M : \mathit{Stream}\ A$ when $M : \mathit{Stream}\ A$

Generator: $\mathbf{corec}\ \{\mathit{Head} \to x\,.\,N \mid \mathit{Tail}\ \beta \to y\,.\,P\}\ \mathbf{with}\ M$

- Accumulator M, named x and y in the branches

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don't mention continuations directly; implicit "evaluation contexts"
  - Contexts named by $\mu\alpha\,.\,J$; invoked by jumps $\langle M \,\|\, \alpha \rangle$

Destructors: $\mathit{Head}\ M : A \qquad \mathit{Tail}\ M : \mathit{Stream}\ A \quad \text{when } M : \mathit{Stream}\ A$

Generator: $\mathbf{corec}\ \{\mathit{Head} \to x\,.\,N \mid \mathit{Tail}\ \beta \to y\,.\,P\}\ \mathbf{with}\ M$

- Accumulator M, named x and y in the branches
- Head branch: N computes first element from current accumulator x

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don't mention continuations directly; implicit "evaluation contexts"
  - Contexts named by $\mu\alpha\,.\,J$; invoked by jumps $\langle M \,\|\, \alpha\rangle$

Destructors: $\quad \mathit{Head}\ M : A \qquad \mathit{Tail}\ M : \mathit{Stream}\ A \quad$ when $M : \mathit{Stream}\ A$

Generator: $\quad \mathbf{corec}\ \{\mathit{Head} \to x\,.\,N \mid \mathit{Tail}\ \beta \to y\,.\,P\}\ \mathbf{with}\ M$

- Accumulator M, named x and y in the branches
- Head branch: N computes first element from current accumulator x
- Tail branch: P computes one of two options

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don't mention continuations directly; implicit "evaluation contexts"
  - Contexts named by $\mu\alpha \,.\, J$; invoked by jumps $\langle M \| \alpha \rangle$

Destructors: $Head\ M : A \qquad Tail\ M : Stream\ A$ when $M : Stream\ A$

Generator: $\mathbf{corec}\ \{Head \to x \,.\, N \mid Tail\ \beta \to y \,.\, P\}\ \mathbf{with}\ M$

- Accumulator M, named x and y in the branches
- Head branch: N computes first element from current accumulator x
- Tail branch: P computes one of two options
  - Continue: return a new accumulator value from current y used for next corecursive loop

Corecursion on Streams

In the λμ-Calculus

- Functional, direct-style
  - Don't mention continuations directly; implicit "evaluation contexts"
  - Contexts named by $\mu\alpha\,.\,J$; invoked by jumps $\langle M \,\|\, \alpha \rangle$

Destructors: $\mathit{Head}\ M : A \qquad \mathit{Tail}\ M : \mathit{Stream}\ A \quad \text{when } M : \mathit{Stream}\ A$

Generator: $\mathbf{corec}\ \{\mathit{Head} \to x\,.\,N \mid \mathit{Tail}\ \beta \to y\,.\,P\}\ \mathbf{with}\ M$

- Accumulator M, named x and y in the branches
- Head branch: N computes first element from current accumulator x
- Tail branch: P computes one of two options
  - Continue: return a new accumulator value from current y used for next corecursive loop
  - End: send a fully-formed stream to context β; this corecursive loop is finished

Examples of Corecursion

In an Abstract Machine

Examples of Corecursion

In an Abstract Machine

$$count\ x = x, x + 1, x + 2, x + 3\ldots$$

Examples of Corecursion

In an Abstract Machine

$$count\ x = x, x+1, x+2, x+3\ldots$$

$$count = \lambda x\,.\,\textbf{corec}\left\{Head \to y\,.\,y \mid Tail\ \_ \to z\,.\,Succ\ z\right\}\ \textbf{with}\ x$$

Examples of Corecursion

In an Abstract Machine

$$count\ x = x,\ x+1,\ x+2,\ x+3\ldots$$

$$count = \lambda x\,.\,\mathbf{corec}\left\{Head \to y\,.\,y \mid Tail\ \_ \to z\,.\,Succ\ z\right\}\ \mathbf{with}\ x$$

$$scons\ x\ (y_0, y_1, y_2\ldots) = x,\ y_0,\ y_1,\ y_2\ldots$$

Examples of Corecursion

In an Abstract Machine

$$count\ x = x,\ x+1,\ x+2,\ x+3\ldots$$

$$count = \lambda x\,.\,\mathbf{corec}\left\{Head \to y\,.\,y \mid Tail\ \_ \to z\,.\,Succ\ z\right\}\ \mathbf{with}\ x$$

$$scons\ x\ (y_0, y_1, y_2\ldots) = x,\ y_0,\ y_1,\ y_2\ldots$$

$$scons = \lambda x\,.\,\lambda ys\,.\,\mathbf{corec}\left\{Head \to \_\,.\,x \mid Tail\ \alpha \to \_\,.\,\mu\delta\,.\,\langle ys \,\|\, \alpha\rangle\right\}\ \mathbf{with}\ \_$$

Examples of Corecursion

In an Abstract Machine

$$count\ x = x,\ x+1,\ x+2,\ x+3\ldots$$

$$count = \lambda x\,.\,\mathbf{corec}\left\{Head \rightarrow y\,.\,y \mid Tail\ \_ \rightarrow z\,.\,Succ\ z\right\}\ \mathbf{with}\ x$$

$$scons\ x\ (y_0, y_1, y_2\ldots) = x,\ y_0,\ y_1,\ y_2\ldots$$

$$scons = \lambda x\,.\,\lambda ys\,.\,\mathbf{corec}\left\{Head \rightarrow \_\,.\,x \mid Tail\ \alpha \rightarrow \_\,.\,\mu\delta\,.\,\langle ys \,\|\, \alpha\rangle\right\}\ \mathbf{with}\ \_$$

$$app\ [x_0, x_1, \ldots, x_n]\ (y_0, y_1, y_2\ldots) = x_0,\ x_1,\ \ldots,\ x_n,\ y_0,\ y_1,\ y_2\ldots$$

Examples of Corecursion

In an Abstract Machine

$$count\ x = x,\ x+1,\ x+2,\ x+3\ldots$$

$$count = \lambda x\,.\,\mathbf{corec}\left\{Head \to y\,.\,y \mid Tail\ \_ \to z\,.\,Succ\ z\right\}\ \mathbf{with}\ x$$

$$scons\ x\ (y_0, y_1, y_2\ldots) = x,\ y_0,\ y_1,\ y_2\ldots$$

$$scons = \lambda x\,.\,\lambda ys\,.\,\mathbf{corec}\left\{Head \to \_\,.\,x \mid Tail\ \alpha \to \_\,.\,\mu\delta\,.\,\langle ys \,\|\, \alpha\rangle\right\}\ \mathbf{with}\ \_$$

$$app\ [x_0, x_1, \ldots, x_n]\ (y_0, y_1, y_2\ldots) = x_0,\ x_1,\ \ldots,\ x_n,\ y_0,\ y_1,\ y_2\ldots$$

$$app = \lambda xs\,.\,\lambda ys\,.\,\mathbf{corec}\left\{\begin{array}{l} Head \to Cons\ x\ xs\,.\,x \\ Tail\ \_ \to Cons\ x\ xs\,.\,xs \\ Head \to Nil\,.\,Head\ ys \\ Tail\ \alpha \to Nil\,.\,\mu\delta\,.\,\langle Tail\ ys \,\|\, \alpha\rangle \end{array}\right\}\ \mathbf{with}\ xs$$

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

$$\mathbf{coiter} \begin{Bmatrix} \mathit{Head}\ \alpha \rightarrow E \\ \mathit{Tail} \rightarrow \gamma\,.\,F \end{Bmatrix} \mathbf{with}\ V := \mathbf{corec} \begin{Bmatrix} \mathit{Head}\ \alpha \rightarrow E \\ \mathit{Tail}\ \_ \rightarrow \gamma\,.\,F \end{Bmatrix} \mathbf{with}\ V$$

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

$$\mathbf{coiter}\begin{Bmatrix} \mathit{Head}\ \alpha \to E \\ \mathit{Tail} \to \gamma\,.\,F \end{Bmatrix} \mathbf{with}\ V := \mathbf{corec}\begin{Bmatrix} \mathit{Head}\ \alpha \to E \\ \mathit{Tail}\ \_ \to \gamma\,.\,F \end{Bmatrix} \mathbf{with}\ V$$

$$\langle \mathit{Left}\ V \| [E, F] \rangle \mapsto \langle V \| E \rangle \qquad \langle \mathit{Right}\ V \| [E, F] \rangle \mapsto \langle V \| F \rangle$$

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

$$\mathbf{coiter}\left\{\begin{array}{r} \mathit{Head}\ \alpha \to E \\ \mathit{Tail} \to \gamma\,.\,F \end{array}\right\}\ \mathbf{with}\ V := \mathbf{corec}\left\{\begin{array}{r} \mathit{Head}\ \alpha \to E \\ \mathit{Tail}\ \_ \to \gamma\,.\,F \end{array}\right\}\ \mathbf{with}\ V$$

$$\langle \mathit{Left}\ V \,\|\, [E, F]\rangle \mapsto \langle V \,\|\, E\rangle \qquad \langle \mathit{Right}\ V \,\|\, [E, F]\rangle \mapsto \langle V \,\|\, F\rangle$$

$$\mathbf{corec}\left\{\begin{array}{r} \mathit{Head}\ \alpha \to E \\ \mathit{Tail}\ \beta \to \gamma\,.\,F \end{array}\right\}\ \mathbf{with}\ V := \mathbf{coiter}\left\{\begin{array}{r} \mathit{Head}\ \alpha \to [\mathit{Head}\ \alpha,\ E] \\ \mathit{Tail} \to [\beta, \gamma]\,.\,[\mathit{Tail}\ \beta, F] \end{array}\right\}\ \mathbf{with}\ \mathit{Right}\ V$$

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

$$\mathbf{coiter}\begin{Bmatrix} Head\ \alpha \to E \\ Tail \to \gamma\,.\,F \end{Bmatrix} \mathbf{with}\ V := \mathbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \_ \to \gamma\,.\,F \end{Bmatrix} \mathbf{with}\ V$$

$$\langle Left\ V \| [E, F] \rangle \mapsto \langle V \| E \rangle \qquad \langle Right\ V \| [E, F] \rangle \mapsto \langle V \| F \rangle$$

$$\mathbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \beta \to \gamma\,.\,F \end{Bmatrix} \mathbf{with}\ V := \mathbf{coiter}\begin{Bmatrix} Head\ \alpha \to [Head\ \alpha,\ E] \\ Tail \to [\beta, \gamma]\,.\,[Tail\ \beta, F] \end{Bmatrix} \mathbf{with}\ Right\ V$$

- (Amortized) overhead cost; consider *scons x ys*:

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

$$\textbf{coiter}\begin{Bmatrix} Head\ \alpha \to E \\ Tail \to \gamma\,.\,F \end{Bmatrix} \textbf{with}\ V := \textbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \_ \to \gamma\,.\,F \end{Bmatrix} \textbf{with}\ V$$

$$\langle Left\ V \| [E, F] \rangle \mapsto \langle V \| E \rangle \qquad \langle Right\ V \| [E, F] \rangle \mapsto \langle V \| F \rangle$$

$$\textbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \beta \to \gamma\,.\,F \end{Bmatrix} \textbf{with}\ V := \textbf{coiter}\begin{Bmatrix} Head\ \alpha \to [Head\ \alpha,\ E] \\ Tail \to [\beta, \gamma]\,.\,[Tail\ \beta, F] \end{Bmatrix} \textbf{with}\ Right\ V$$

- (Amortized) overhead cost; consider $scons\ x\ ys$:
  - Native **corec**: $Head(Tail^{n+1}(scons\ x\ ys))$ adds $O(1)$ overhead to cost of $Head(Tail^{n}\ ys)$

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

$$\mathbf{coiter}\begin{Bmatrix} Head\ \alpha \to E \\ Tail \to \gamma\,.\,F \end{Bmatrix}\ \mathbf{with}\ V := \mathbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \_ \to \gamma\,.\,F \end{Bmatrix}\ \mathbf{with}\ V$$

$$\langle Left\ V \| [E, F] \rangle \mapsto \langle V \| E \rangle \qquad \langle Right\ V \| [E, F] \rangle \mapsto \langle V \| F \rangle$$

$$\mathbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \beta \to \gamma\,.\,F \end{Bmatrix}\ \mathbf{with}\ V := \mathbf{coiter}\begin{Bmatrix} Head\ \alpha \to [Head\ \alpha,\ E] \\ Tail \to [\beta, \gamma]\,.\,[Tail\ \beta, F] \end{Bmatrix}\ \mathbf{with}\ Right\ V$$

- (Amortized) overhead cost; consider $scons\ x\ ys$:
  - Native **corec**: $Head(Tail^{n+1}(scons\ x\ ys))$ adds $O(1)$ overhead to cost of $Head(Tail^n\ ys)$
  - Encoded **corec**: $Head(Tail^{n+1}(scons\ x\ ys))$ adds $O(n)$ overhead to cost of $Head(Tail^n\ ys)$

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

$$\textbf{coiter}\begin{Bmatrix} Head\ \alpha \to E \\ Tail \to \gamma\,.\,F \end{Bmatrix} \textbf{with } V := \textbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \_ \to \gamma\,.\,F \end{Bmatrix} \textbf{with } V$$

$$\langle Left\ V \| [E, F] \rangle \mapsto \langle V \| E \rangle \qquad \langle Right\ V \| [E, F] \rangle \mapsto \langle V \| F \rangle$$

$$\textbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \beta \to \gamma\,.\,F \end{Bmatrix} \textbf{with } V := \textbf{coiter}\begin{Bmatrix} Head\ \alpha \to [Head\ \alpha,\ E] \\ Tail \to [\beta, \gamma]\,.\,[Tail\ \beta, F] \end{Bmatrix} \textbf{with } Right\ V$$

- (Amortized) overhead cost; consider $scons\ x\ ys$:
  - Native **corec**: $Head(Tail^{n+1}(scons\ x\ ys))$ adds $O(1)$ overhead to cost of $Head(Tail^n\ ys)$
  - Encoded **corec**: $Head(Tail^{n+1}(scons\ x\ ys))$ adds $O(n)$ overhead to cost of $Head(Tail^n\ ys)$
- Native CBN **corec** has same overhead as encoding; Native CBV **corec** more efficient

Corecursion vs Coiteration

Expressiveness vs Cost; CBV vs CBN

$$\textbf{coiter}\begin{Bmatrix} Head\ \alpha \to E \\ Tail \to \gamma\,.\,F \end{Bmatrix} \textbf{with } V := \textbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \_ \to \gamma\,.\,F \end{Bmatrix} \textbf{with } V$$

$$\langle Left\ V \| [E, F]\rangle \mapsto \langle V \| E\rangle \qquad \langle Right\ V \| [E, F]\rangle \mapsto \langle V \| F\rangle$$

$$\textbf{corec}\begin{Bmatrix} Head\ \alpha \to E \\ Tail\ \beta \to \gamma\,.\,F \end{Bmatrix} \textbf{with } V := \textbf{coiter}\begin{Bmatrix} Head\ \alpha \to [Head\ \alpha,\ E] \\ Tail \to [\beta, \gamma]\,.\,[Tail\ \beta, F] \end{Bmatrix} \textbf{with } Right\ V$$

- (Amortized) overhead cost; consider $scons\ x\ ys$:
  - Native **corec**: $Head(Tail^{n+1}(scons\ x\ ys))$ adds $O(1)$ overhead to cost of $Head(Tail^{n}\ ys)$
  - Encoded **corec**: $Head(Tail^{n+1}(scons\ x\ ys))$ adds $O(n)$ overhead to cost of $Head(Tail^{n}\ ys)$
- Native CBN **corec** has same overhead as encoding; Native CBV **corec** more efficient
- Corollary by duality of **rec** and **iter**

(Co)Inductive Reasoning

Finite Induction

By Inversion on the Input

Finite Induction

By Inversion on the Input

$$\Gamma, x : Bool \vdash \Phi(x)$$

Finite Induction

By Inversion on the Input

$$\frac{}{\Gamma, x : Bool \vdash \Phi(x)}$$

Finite Induction

By Inversion on the Input

$$\frac{\Gamma \vdash \Phi(True)}{\Gamma, x : Bool \vdash \Phi(x)}$$

Finite Induction

By Inversion on the Input

$$\frac{\Gamma \vdash \Phi(\mathit{True}) \qquad \Gamma \vdash \Phi(\mathit{False})}{\Gamma, x : \mathit{Bool} \vdash \Phi(x)}$$

Finite Induction

By Inversion on the Input

$$\frac{\Gamma \vdash \Phi(\mathit{True}) \qquad \Gamma \vdash \Phi(\mathit{False})}{\Gamma, x : \mathit{Bool} \vdash \Phi(x)}$$

Infinite Induction

By Inversion on the Input

Infinite Induction

By Inversion on the Input

$$\Gamma, x : Nat \vdash \Phi(x)$$

Infinite Induction

By Inversion on the Input

$$\frac{}{\Gamma, x : Nat \vdash \Phi(x)}$$

Infinite Induction

By Inversion on the Input

$$\frac{\Gamma \vdash \Phi(0)}{\Gamma, x : Nat \vdash \Phi(x)}$$

Infinite Induction

By Inversion on the Input

$$\frac{\Gamma \vdash \Phi(0) \quad \Gamma \vdash \Phi(1)}{\Gamma, x : Nat \vdash \Phi(x)}$$

Infinite Induction

By Inversion on the Input

$$\frac{\Gamma \vdash \Phi(0) \quad \Gamma \vdash \Phi(1) \quad \Gamma \vdash \Phi(2)}{\Gamma, x : Nat \vdash \Phi(x)}$$

Infinite Induction

By Inversion on the Input

$$\frac{\Gamma \vdash \Phi(0) \quad \Gamma \vdash \Phi(1) \quad \Gamma \vdash \Phi(2) \quad \cdots}{\Gamma, x : Nat \vdash \Phi(x)}$$

Infinite Induction

By Inversion on the Input

$$\frac{\Gamma \vdash \Phi(0) \quad \Gamma \vdash \Phi(1) \quad \Gamma \vdash \Phi(2) \quad \cdots}{\Gamma, x : Nat \vdash \Phi(x)}$$

?

An Induction Principle

Based on Information Flow

An Induction Principle

Based on Information Flow

$$\frac{}{\Gamma, x : Nat \vdash \Phi(x)}$$

An Induction Principle

Based on Information Flow

$$\frac{\Gamma \vdash \Phi(Zero)}{\Gamma, x : Nat \vdash \Phi(x)}$$

An Induction Principle

Based on Information Flow

$$\frac{\Gamma \vdash \Phi(\mathit{Zero}) \qquad \Gamma, x : \mathit{Nat}, \Phi(x) \vdash \Phi(\mathit{Succ}\ x)}{\Gamma, x : \mathit{Nat} \vdash \Phi(x)}$$

An Induction Principle

Based on Information Flow

$$\frac{\Gamma \vdash \Phi(Zero) \qquad \Gamma, x : Nat, \Phi(x) \vdash \Phi(Succ\ x)}{\Gamma, x : Nat \vdash \Phi(x)}$$

An Induction Principle

Based on Information Flow

$$\frac{\Gamma \vdash \Phi(\mathit{Zero}) \qquad \Gamma, x : \mathit{Nat}, \Phi(x) \vdash \Phi(\mathit{Succ}\ x)}{\Gamma, x : \mathit{Nat} \vdash \Phi(x)}$$

$$\Phi(\mathit{Zero}) \Rightarrow (\forall x{:}\mathit{Nat}\,.\,\Phi(x) \Rightarrow \Phi(x+1))$$
$$\Rightarrow (\forall x{:}\mathit{Nat}\,.\,\Phi(x))$$

Finite Coinduction

By Inversion on the Output

Finite Coinduction

By Inversion on the Output

$$\lambda x\,.\,V\ x =_{\eta} V$$

Finite Coinduction

By Inversion on the Output

$$\Gamma \vdash V = V' : A \to B$$

$$\lambda x \,.\, V\,x =_\eta V$$

Finite Coinduction

By Inversion on the Output

$$\frac{\Gamma, x : A \vdash V\ x = V'\ x : B}{\Gamma \vdash V = V' : A \to B}$$

$$\lambda x\,.\,V\ x =_\eta V$$

Finite Coinduction

By Inversion on the Output

$$\frac{\Gamma, x : A \vdash V\ x = V'\ x : B}{\Gamma \vdash V = V' : A \to B}$$

$$\lambda x \,.\, V\ x =_\eta V$$

$$\lambda x \,.\, \mu\beta \,.\, \langle V \| x \cdot \beta \rangle =_\eta V$$

Finite Coinduction

By Inversion on the Output

$$\frac{\Gamma, x : A \vdash V\ x = V'\ x : B}{\Gamma \vdash V = V' : A \to B}$$

$$\lambda x\,.\,V\ x =_\eta V$$

$$\Gamma, \alpha \div A \to B \vdash \Phi(\alpha)$$

$$\lambda x\,.\,\mu\beta\,.\,\langle V \| x \cdot \beta \rangle =_\eta V$$

Finite Coinduction

By Inversion on the Output

$$\frac{\Gamma, x : A \vdash V\ x = V'\ x : B}{\Gamma \vdash V = V' : A \to B}$$

$$\lambda x\,.\,V\ x =_\eta V$$

$$\frac{}{\Gamma, \alpha \div A \to B \vdash \Phi(\alpha)}$$

$$\lambda x\,.\,\mu\beta\,.\,\langle V \| x \cdot \beta \rangle =_\eta V$$

Finite Coinduction

By Inversion on the Output

$$\frac{\Gamma, x : A \vdash V\ x = V'\ x : B}{\Gamma \vdash V = V' : A \to B}$$

$$\lambda x\,.\,V\ x =_\eta V$$

$$\frac{\Gamma, x : A, \beta \div B \vdash \Phi(x \cdot \beta)}{\Gamma, \alpha \div A \to B \vdash \Phi(\alpha)}$$

$$\lambda x\,.\,\mu\beta\,.\,\langle V \| x \cdot \beta \rangle =_\eta V$$

Infinite Coinduction

By Inversion on the Output

Infinite Coinduction

By Inversion on the Output

$$\Gamma, \alpha \div Stream\ A \vdash \Phi(\alpha)$$

Infinite Coinduction

By Inversion on the Output

$$\frac{}{\Gamma, \alpha \div Stream\ A \vdash \Phi(\alpha)}$$

Infinite Coinduction

By Inversion on the Output

$$\frac{\Gamma, \beta \div A \vdash \Phi(Head\ \beta)}{\Gamma, \alpha \div Stream\ A \vdash \Phi(\alpha)}$$

Infinite Coinduction

By Inversion on the Output

$$\frac{\Gamma, \beta \div A \vdash \Phi(\mathit{Head}\ \beta) \qquad \Gamma, \beta \div A \vdash \Phi(\mathit{Tail}(\mathit{Head}\ \beta))}{\Gamma, \alpha \div \mathit{Stream}\ A \vdash \Phi(\alpha)}$$

Infinite Coinduction

By Inversion on the Output

$$\dfrac{\begin{array}{c}\Gamma, \beta \div A \vdash \Phi(\mathit{Head}\ \beta) \\ \Gamma, \beta \div A \vdash \Phi(\mathit{Tail}(\mathit{Head}\ \beta)) \\ \Gamma, \beta \div A \vdash \Phi(\mathit{Tail}(\mathit{Tail}(\mathit{Head}\ \beta))) \\ \vdots\end{array}}{\Gamma, \alpha \div \mathit{Stream}\ A \vdash \Phi(\alpha)}$$

Infinite Coinduction

By Inversion on the Output

$$\frac{\begin{array}{l}\Gamma, \beta \div A \vdash \Phi(\mathit{Head}\ \beta) \\ \Gamma, \beta \div A \vdash \Phi(\mathit{Tail}(\mathit{Head}\ \beta)) \\ \Gamma, \beta \div A \vdash \Phi(\mathit{Tail}(\mathit{Tail}(\mathit{Head}\ \beta))) \\ \qquad\qquad\vdots\end{array}}{\Gamma, \alpha \div \mathit{Stream}\ A \vdash \Phi(\alpha)}$$

Infinite Coinduction

By Inversion on the Output

$$
\dfrac{\begin{array}{c}\Gamma, \beta \div A \vdash \Phi(Head\ \beta) \\ \Gamma, \beta \div A \vdash \Phi(Tail(Head\ \beta)) \\ \Gamma, \beta \div A \vdash \Phi(Tail(Tail(Head\ \beta))) \\ \vdots \end{array}}{\Gamma, \alpha \div Stream\ A \vdash \Phi(\alpha)}
$$

?

A Coinduction Principle

Based on Control Flow

A Coinduction Principle

Based on Control Flow

$$\frac{}{\Gamma, \alpha \div Stream\ A \vdash \Phi(\alpha)}$$

A Coinduction Principle

Based on Control Flow

$$\frac{\Gamma, \beta \div A \vdash \Phi(\mathit{Head}\ \beta)}{\Gamma, \alpha \div \mathit{Stream}\ A \vdash \Phi(\alpha)}$$

A Coinduction Principle

Based on Control Flow

$$\frac{\Gamma, \beta \div A \vdash \Phi(\mathit{Head}\ \beta) \qquad \Gamma, \alpha \div \mathit{Stream}\ A, \Phi(\alpha) \vdash \Phi(\mathit{Tail}\ \alpha)}{\Gamma, \alpha \div \mathit{Stream}\ A \vdash \Phi(\alpha)}$$

A Coinduction Principle

Based on Control Flow

$$\frac{\Gamma, \beta \div A \vdash \Phi(\mathit{Head}\ \beta) \qquad \Gamma, \alpha \div \mathit{Stream}\ A, \Phi(\alpha) \vdash \Phi(\mathit{Tail}\ \alpha)}{\Gamma, \alpha \div \mathit{Stream}\ A \vdash \Phi(\alpha)}$$

A Coinduction Principle

Based on Control Flow

$$\frac{\Gamma, \beta \div A \vdash \Phi(\mathit{Head}\ \beta) \qquad \Gamma, \alpha \div \mathit{Stream}\ A, \Phi(\alpha) \vdash \Phi(\mathit{Tail}\ \alpha)}{\Gamma, \alpha \div \mathit{Stream}\ A \vdash \Phi(\alpha)}$$

$$\begin{aligned}
&\mathit{Bisimulation} \\
&\quad = (\forall s, s' : \mathit{Stream}\ A\,.\,\Phi(s, s') \Rightarrow \mathit{Head}\ s = \mathit{Head}\ s' : A) \\
&\quad \Rightarrow (\forall s, s' : \mathit{Stream}\ A\,.\,\Phi(s, s') \Rightarrow \Phi(\mathit{Tail}\ s, \mathit{Tail}\ s')) \\
&\quad \Rightarrow (\forall s, s' : \mathit{Stream}\ A\,.\,\Phi(s, s') \Rightarrow s = s' : \mathit{Stream}\ A)
\end{aligned}$$

A Coinduction Principle

Based on Control Flow

$$\frac{\Gamma, \beta \div A \vdash \Phi(\mathit{Head}\ \beta) \qquad \Gamma, \alpha \div \mathit{Stream}\ A, \Phi(\alpha) \vdash \Phi(\mathit{Tail}\ \alpha)}{\Gamma, \alpha \div \mathit{Stream}\ A \vdash \Phi(\alpha)}$$

$$\Downarrow$$

$$\begin{aligned}
&\mathit{Bisimulation} \\
&\quad = \ (\forall s, s' : \mathit{Stream}\ A\,.\,\Phi(s, s') \Rightarrow \mathit{Head}\ s = \mathit{Head}\ s' : A) \\
&\quad \Rightarrow (\forall s, s' : \mathit{Stream}\ A\,.\,\Phi(s, s') \Rightarrow \Phi(\mathit{Tail}\ s, \mathit{Tail}\ s')) \\
&\quad \Rightarrow (\forall s, s' : \mathit{Stream}\ A\,.\,\Phi(s, s') \Rightarrow s = s' : \mathit{Stream}\ A)
\end{aligned}$$

Proof by Coinduction

Proof by Coinduction

$repeat\ x = x, x, x\ldots \qquad alt = 0,1,0,1\ldots \qquad evens\ (x_0, x_1, x_2\ldots) = x_0, x_2, x_4\ldots$

Proof by Coinduction

$$repeat\ x = x, x, x\ldots \qquad alt = 0,1,0,1\ldots \qquad evens\ (x_0, x_1, x_2\ldots) = x_0, x_2, x_4\ldots$$

Theorem: $evens\ alt = repeat\ 0 : Stream\ A$

Proof by Coinduction

$$repeat\ x = x, x, x\ldots \qquad alt = 0{,}1{,}0{,}1\ldots \qquad evens\ (x_0, x_1, x_2\ldots) = x_0, x_2, x_4\ldots$$

Theorem: $evens\ alt = repeat\ 0 : Stream\ A$

- *S.T.S:* $\alpha \div Stream\ A \vdash \langle evens\ alt \,\|\, \alpha \rangle = \langle repeat\ 0 \,\|\, \alpha \rangle$

Proof by Coinduction

$$repeat\ x = x, x, x\ldots \qquad alt = 0,1,0,1\ldots \qquad evens\ (x_0, x_1, x_2\ldots) = x_0, x_2, x_4\ldots$$

Theorem: $evens\ alt = repeat\ 0 : Stream\ A$

- *S.T.S*: $\alpha \div Stream\ A \vdash \langle evens\ alt \| \alpha \rangle = \langle repeat\ 0 \| \alpha \rangle$

Proof: By coinduction on $\alpha \div Stream\ A$...

Proof by Coinduction

$$repeat\ x = x, x, x\dots \qquad alt = 0,1,0,1\dots \qquad evens\ (x_0, x_1, x_2\dots) = x_0, x_2, x_4\dots$$

Theorem: $evens\ alt = repeat\ 0 : Stream\ A$

- S.T.S: $\alpha \div Stream\ A \vdash \langle evens\ alt \| \alpha \rangle = \langle repeat\ 0 \| \alpha \rangle$

Proof: By coinduction on $\alpha \div Stream\ A$...

- $\alpha = Head\ \beta$: $\langle evens\ alt \| Head\ \beta \rangle = \langle 0 \| \beta \rangle = \langle repeat\ 0 \| Head\ \beta \rangle$

Proof by Coinduction

$$repeat\ x = x, x, x \ldots \qquad alt = 0{,}1{,}0{,}1 \ldots \qquad evens\ (x_0, x_1, x_2 \ldots) = x_0, x_2, x_4 \ldots$$

Theorem: $evens\ alt = repeat\ 0 : Stream\ A$

- S.T.S: $\alpha \div Stream\ A \vdash \langle evens\ alt \| \alpha \rangle = \langle repeat\ 0 \| \alpha \rangle$

Proof: By coinduction on $\alpha \div Stream\ A$...

- $\alpha = Head\ \beta$: $\langle evens\ alt \| Head\ \beta \rangle = \langle 0 \| \beta \rangle = \langle repeat\ 0 \| Head\ \beta \rangle$
- $\alpha = Tail\ \beta$: Assume CoIH $\langle evens\ alt \| \beta \rangle = \langle repeat\ 0 \| \beta \rangle$ and show $\langle evens\ alt \| Tail\ \beta \rangle = \langle repeat\ 0 \| Tail\ \beta \rangle$...

Proof by Coinduction

$$repeat\ x = x, x, x \ldots \qquad alt = 0,1,0,1 \ldots \qquad evens\ (x_0, x_1, x_2 \ldots) = x_0, x_2, x_4 \ldots$$

Theorem: $evens\ alt = repeat\ 0 : Stream\ A$

- S.T.S: $\alpha \div Stream\ A \vdash \langle evens\ alt \| \alpha \rangle = \langle repeat\ 0 \| \alpha \rangle$

Proof: By coinduction on $\alpha \div Stream\ A$...

- $\alpha = Head\ \beta$: $\langle evens\ alt \| Head\ \beta \rangle = \langle 0 \| \beta \rangle = \langle repeat\ 0 \| Head\ \beta \rangle$
- $\alpha = Tail\ \beta$: Assume CoIH $\langle evens\ alt \| \beta \rangle = \langle repeat\ 0 \| \beta \rangle$ and show $\langle evens\ alt \| Tail\ \beta \rangle = \langle repeat\ 0 \| Tail\ \beta \rangle$...

$$\begin{aligned}
\langle evens\ alt \| Tail\ \beta \rangle &= \langle evens\ (Tail(Tail\ alt)) \| \beta \rangle && (def.\ evens)\\
&= \langle evens\ alt \| \beta \rangle && (def.\ alt)\\
&= \langle repeat\ 0 \| \beta \rangle && (CoIH)\\
&= \langle repeat\ 0 \| Tail\ \beta \rangle && (def.\ repeat)
\end{aligned}$$

Weak vs Strong (Co)Induction

And Effectful Computation

Weak vs Strong (Co)Induction

And Effectful Computation

- **Strong** (co)induction proves any property Φ

Weak vs Strong (Co)Induction

And Effectful Computation

- **Strong** (co)induction proves any property Φ
  - Strong induction is ***unsound*** in CBN

Weak vs Strong (Co)Induction

And Effectful Computation

- **Strong** (co)induction proves any property Φ
  - Strong induction is ***unsound*** in CBN
  - Strong coinduction is ***unsound*** in CBV

Weak vs Strong (Co)Induction

And Effectful Computation

- **Strong** (co)induction proves any property Φ
  - Strong induction is ***unsound*** in CBN
  - Strong coinduction is ***unsound*** in CBV
- *Weak* (co)induction restricts Φ

Weak vs Strong (Co)Induction

And Effectful Computation

- **Strong** (co)induction proves any property Φ
  - Strong induction is ***unsound*** in CBN
  - Strong coinduction is ***unsound*** in CBV
- *Weak* (co)induction restricts Φ
  - Weak induction on x: must be *strict on x* like $\langle x \| E \rangle = \langle x \| E' \rangle$

Weak vs Strong (Co)Induction

And Effectful Computation

- **Strong** (co)induction proves any property Φ
  - Strong induction is ***unsound*** in CBN
  - Strong coinduction is ***unsound*** in CBV
- *Weak* (co)induction restricts Φ
  - Weak induction on x: must be *strict on x* like $\langle x \| E \rangle = \langle x \| E' \rangle$
  - Weak induction on α: must be *productive on α* like $\langle V \| \alpha \rangle = \langle V' \| \alpha \rangle$

Weak vs Strong (Co)Induction

And Effectful Computation

- **Strong** (co)induction proves any property Φ
  - Strong induction is ***unsound*** in CBN
  - Strong coinduction is ***unsound*** in CBV
- *Weak* (co)induction restricts Φ
  - Weak induction on x: must be *strict on x* like $\langle x \| E \rangle = \langle x \| E' \rangle$
  - Weak induction on α: must be *productive on α* like $\langle V \| \alpha \rangle = \langle V' \| \alpha \rangle$
- Weak (co)induction is *always sound*

Lessons Learned

Lessons Learned

- Duality — Ideas for free!

Lessons Learned

- Duality — Ideas for free!
- Impact of evaluation, computation, effects, divergence
  - CBV: strong induction and efficient **co**recursion
  - CBN: strong **co**induction and efficient recursion
  - Future work: Call-by-push-value or polarities could get best of both worlds

Lessons Learned

- Duality — Ideas for free!
- Impact of evaluation, computation, effects, divergence
  - CBV: strong induction and efficient **co**recursion
  - CBN: strong **co**induction and efficient recursion
  - Future work: Call-by-push-value or polarities could get best of both worlds
- (Co)Induction are both inversion principles
  - Induction: inversion on input, guided by information flow
  - Coinduction: inversion on output, guided by control flow