Beating the Productivity Checker Using Embedded Languages

Nils Anders Danielsson (Nottingham)

PAR 2010, Edinburgh, 2010-07-15

Introduction

Guarded corecursion provides a simple principle for defining productive values:

$$
\begin{aligned}
&\mathit{iterate} \;:\; (A \to A) \to A \to \mathit{Stream}\ A \\
&\mathit{iterate}\ f\ x \;=\; x :: \sharp\ \mathit{iterate}\ f\ (f\ x)
\end{aligned}
$$

Introduction

Many productive, corecursive definitions fail to be guarded:

$$\begin{aligned} \mathit{fib} &: \mathit{Stream}\ \mathbb{N} \\ \mathit{fib} &= 0 :: \sharp\ \mathit{zipWith}\ \_+\_\ \mathit{fib}\ (1 :: \sharp\ \mathit{fib}) \end{aligned}$$

Introduction

Many productive, corecursive definitions fail to be guarded:

$$\mathit{fib}\ :\ \mathit{Stream}\ \mathbb{N}$$
$$\mathit{fib}\ =\ \mathit{0} :: \sharp\ \mathit{zipWith}\ \_+\_\ \mathit{fib}\ (\mathit{1} :: \sharp\ \mathit{fib})$$

Introduction

Many productive, corecursive definitions fail to be guarded:

$$\begin{aligned} \mathit{fib} &: \mathit{Stream}\ \mathbb{N} \\ \mathit{fib} &= \mathit{0} :: \sharp\ \mathit{zipWith}\ \_+\_\ \mathit{fib}\ (\mathit{1} :: \sharp\ \mathit{fib}) \end{aligned}$$

This talk: An *ad-hoc*, *manual* (but useful) method for making *productive* definitions guarded.

Introduction

Many productive, corecursive definitions fail to be guarded:

$$\mathit{fib} \;:\; \mathit{Stream}\;\mathbb{N}$$
$$\mathit{fib} \;=\; 0 :: \sharp\; \mathit{zipWith}\; \_+\_\; \mathit{fib}\; (1 :: \sharp\; \mathit{fib})$$

Simple observation: If *zipWith* were a constructor, then the definition would be accepted.

Fibonacci sequence

Streams:

$$\textbf{data}\ \mathit{Stream}\ (A : \mathit{Set}) : \mathit{Set}\ \textbf{where}$$
$$\_{::}\_ : A \to \infty\,(\mathit{Stream}\ A) \to \mathit{Stream}\ A$$

Stream programs:

$$\textbf{data}\ \mathit{Stream}_{\mathrm{P}} : \mathit{Set} \to \mathit{Set}_1\ \textbf{where}$$
$$\_{::}\_ : A \to \infty\,(\mathit{Stream}_{\mathrm{P}}\ A) \to \mathit{Stream}_{\mathrm{P}}\ A$$
$$\mathsf{zipWith} : (A \to B \to C) \to \mathit{Stream}_{\mathrm{P}}\ A \to \mathit{Stream}_{\mathrm{P}}\ B \to \mathit{Stream}_{\mathrm{P}}\ C$$

Fibonacci sequence

Stream programs:

$$
\begin{array}{lll}
\textbf{data}\ \mathit{Stream}_{\mathsf{P}} & : & \mathit{Set} \to \mathit{Set}_1\ \textbf{where} \\
\quad \_{::}\_ & : & A \to \infty\,(\mathit{Stream}_{\mathsf{P}}\ A) \to \mathit{Stream}_{\mathsf{P}}\ A \\
\quad \mathsf{zipWith} & : & (A \to B \to C) \to \\
 & & \mathit{Stream}_{\mathsf{P}}\ A \to \mathit{Stream}_{\mathsf{P}}\ B \to \mathit{Stream}_{\mathsf{P}}\ C
\end{array}
$$

Weak head normal forms:

$$
\begin{array}{l}
\textbf{data}\ \mathit{Stream}_{\mathsf{W}}\ :\ \mathit{Set} \to \mathit{Set}_1\ \textbf{where} \\
\quad \_{::}\_\ :\ A \to \mathit{Stream}_{\mathsf{P}}\ A \to \mathit{Stream}_{\mathsf{W}}\ A
\end{array}
$$

Fibonacci sequence

Stream programs:

$$\textbf{data}\ \mathit{Stream}_{\mathrm{P}} : \mathit{Set} \to \mathit{Set}_1\ \textbf{where}$$
$$\_{::}\_ : A \to \infty\,(\mathit{Stream}_{\mathrm{P}}\ A) \to \mathit{Stream}_{\mathrm{P}}\ A$$
$$\mathrm{zipWith} : (A \to B \to C) \to \mathit{Stream}_{\mathrm{P}}\ A \to \mathit{Stream}_{\mathrm{P}}\ B \to \mathit{Stream}_{\mathrm{P}}\ C$$

Weak head normal forms:

$$\textbf{data}\ \mathit{Stream}_{\mathrm{W}} : \mathit{Set} \to \mathit{Set}_1\ \textbf{where}$$
$$\_{::}\_ : A \to \mathit{Stream}_{\mathrm{P}}\ A \to \mathit{Stream}_{\mathrm{W}}\ A$$

Fibonacci sequence

Turning programs into WHNFs:

$$
\begin{aligned}
&\mathit{whnf} \;:\; \mathit{Stream}_{\mathsf{P}}\; A \to \mathit{Stream}_{\mathsf{W}}\; A \\
&\mathit{whnf}\; (x :: \mathit{xs}) \qquad\qquad\quad = \; x :: \flat\; \mathit{xs} \\
&\mathit{whnf}\; (\mathsf{zipWith}\; f\; \mathit{xs}\; \mathit{ys}) \; = \\
&\quad \mathit{zipWith}_{\mathsf{W}}\; f\; (\mathit{whnf}\; \mathit{xs})\; (\mathit{whnf}\; \mathit{ys})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{zipWith}_{\mathsf{W}} \;:\; (A \to B \to C) \to \\
&\qquad\qquad\quad \mathit{Stream}_{\mathsf{W}}\; A \to \mathit{Stream}_{\mathsf{W}}\; B \to \mathit{Stream}_{\mathsf{W}}\; C \\
&\mathit{zipWith}_{\mathsf{W}}\; f\; (x :: \mathit{xs})\; (y :: \mathit{ys}) \; = \\
&\quad f\; x\; y :: \mathsf{zipWith}\; f\; \mathit{xs}\; \mathit{ys}
\end{aligned}
$$

Fibonacci sequence

Turning programs into streams:

$$\llbracket \_ \rrbracket_{\mathsf{W}} \;:\; \mathit{Stream}_{\mathsf{W}}\ A \rightarrow \mathit{Stream}\ A$$
$$\llbracket\ x :: xs\ \rrbracket_{\mathsf{W}} \;=\; x :: \sharp\ \llbracket\ \mathit{whnf}\ \ xs\ \rrbracket_{\mathsf{W}}$$

Fibonacci sequence

Turning programs into streams:

mutual

$$[\![\_]\!]_{\mathsf{W}} \ : \ \mathit{Stream}_{\mathsf{W}}\ A \to \mathit{Stream}\ A$$
$$[\![\ x :: xs\ ]\!]_{\mathsf{W}} \ = \ x :: \sharp\ [\![\ xs\ ]\!]_{\mathsf{P}}$$

$$[\![\_]\!]_{\mathsf{P}} \ : \ \mathit{Stream}_{\mathsf{P}}\ A \to \mathit{Stream}\ A$$
$$[\![\ xs\ ]\!]_{\mathsf{P}} \ = \ [\![\ \mathit{whnf}\ \ xs\ ]\!]_{\mathsf{W}}$$

Fibonacci sequence

The sequence itself:

$$
\begin{aligned}
&\mathit{fib}_{\mathsf{P}} \;:\; \mathit{Stream}_{\mathsf{P}}\ \mathbb{N} \\
&\mathit{fib}_{\mathsf{P}} \;=\; \mathit{0} :: \sharp\ \mathsf{zipWith}\ \_+\_\ \mathit{fib}_{\mathsf{P}}\ (\mathit{1} :: \sharp\ \mathit{fib}_{\mathsf{P}})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{fib} \;:\; \mathit{Stream}\ \mathbb{N} \\
&\mathit{fib} \;=\; [\![\ \mathit{fib}_{\mathsf{P}}\ ]\!]_{\mathsf{P}}
\end{aligned}
$$

Fibonacci sequence

Properties (have to be proved manually):

$$\mathit{Fib\text{-}like} \;:\; \mathit{Stream}\ \mathbb{N} \to \mathit{Set}$$
$$\mathit{Fib\text{-}like}\ \mathit{ns} \;=\; \mathit{ns} \approx 0 :: \sharp\ \mathit{zipWith}\ \_{+}\_\ \mathit{ns}\ (1 :: \sharp\ \mathit{ns})$$

$$\mathit{Fib\text{-}like}\ \mathit{fib}$$

$$\mathit{Fib\text{-}like}\ \mathit{ms} \to \mathit{Fib\text{-}like}\ \mathit{ns} \to \mathit{ms} \approx \mathit{ns}$$

$$[\![\ \mathsf{zipWith}\ f\ \mathit{xs}\ \mathit{ys}\ ]\!]_{\mathsf{P}} \approx \mathit{zipWith}\ f\ [\![\ \mathit{xs}\ ]\!]_{\mathsf{P}}\ [\![\ \mathit{ys}\ ]\!]_{\mathsf{P}}$$

The method

1. Construct language including offending functions as constructors.
2. Define WHNF type.
3. Write *whnf* function.
4. Write interpreter: $[\![\_]\!]$.
5. Write programs in language and interpret them.
6. (Optional.) Prove properties about programs.

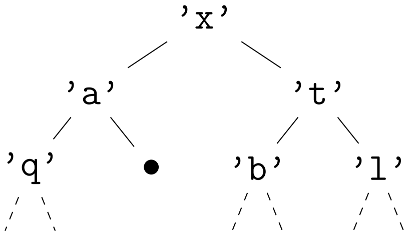
Breadth-first labelling

Breadth-first labelling

Potentially infinite trees:

$$
\begin{array}{l}
\mathbf{data}\ \mathit{Tree}\ (A\ :\ \mathit{Set})\ :\ \mathit{Set}\ \mathbf{where} \\
\quad \mathsf{leaf}\ \ :\ \mathit{Tree}\ A \\
\quad \mathsf{node}\ :\ \infty\,(\mathit{Tree}\ A) \to A \to \infty\,(\mathit{Tree}\ A) \to \mathit{Tree}\ A
\end{array}
$$

Breadth-first labelling

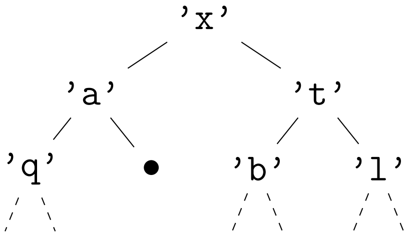


Breadth-first labelling

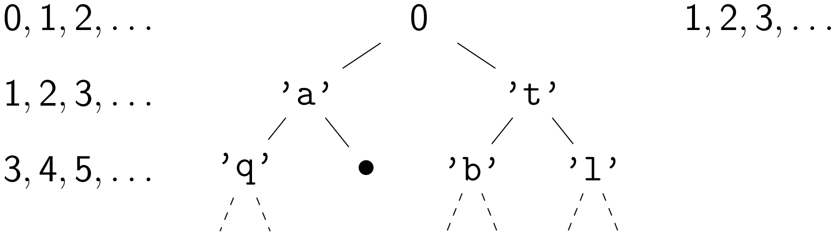
$0, 1, 2, \ldots$

$1, 2, 3, \ldots$

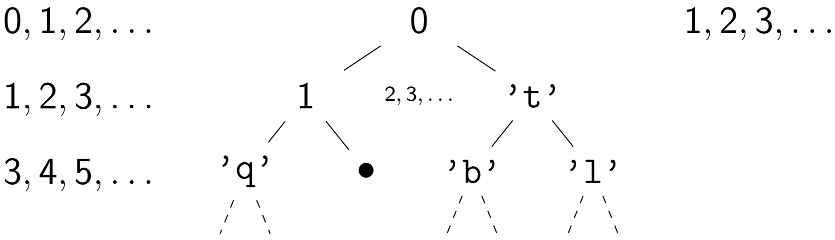
$3, 4, 5, \ldots$



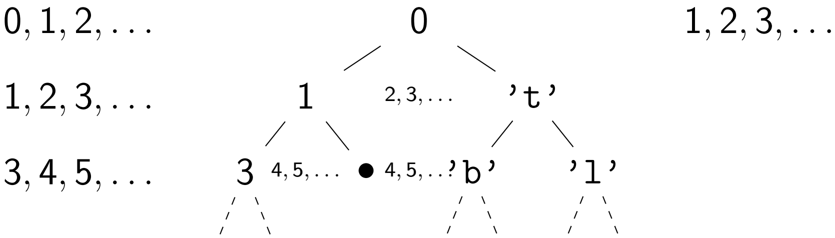
Breadth-first labelling



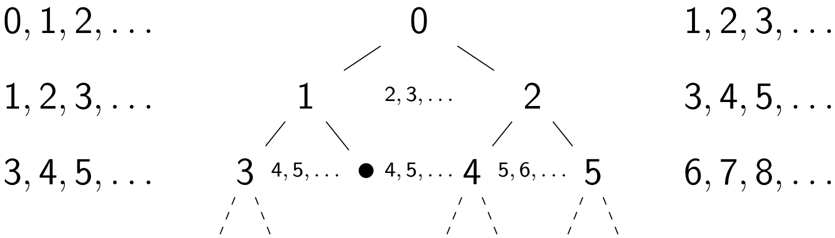
Breadth-first labelling



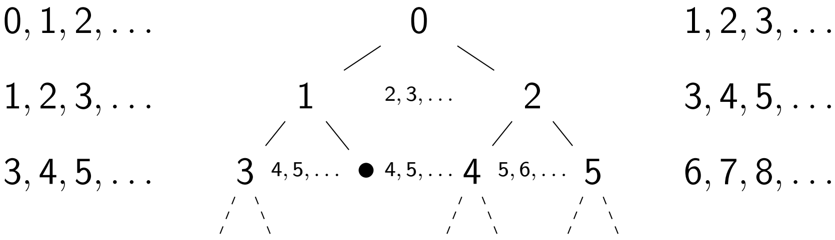
Breadth-first labelling



Breadth-first labelling

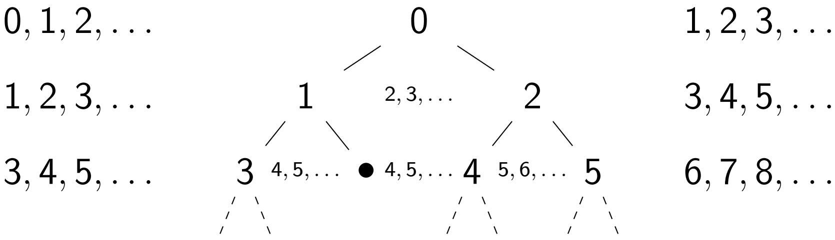


Breadth-first labelling



$$\mathit{lab} : \mathit{Tree}\ A \rightarrow \mathit{Stream}\ (\mathit{Stream}\ B) \rightarrow$$
$$\mathit{Tree}\ B \times \mathit{Stream}\ (\mathit{Stream}\ B)$$

Breadth-first labelling



$$\begin{aligned}
\mathit{lab} \ &: \ \mathit{Tree}\ A \to \mathit{Stream}\ (\mathit{Stream}\ B) \to \\
&\quad\ \mathit{Tree}\ B \times \mathit{Stream}\ (\mathit{Stream}\ B)
\end{aligned}$$

$$\begin{aligned}
&\mathit{label} \ : \ \mathit{Tree}\ A \to \mathit{Stream}\ B \to \mathit{Tree}\ B \\
&\mathit{label}\ t\ \mathit{bs} \ = \ t' \\
&\quad \textbf{where}\ (t', \mathit{bss}) \ = \ \mathit{lab}\ t\ (\mathit{bs} :: \sharp\ \mathit{bss})
\end{aligned}$$

Breadth-first labelling

A small universe:

$$
\begin{aligned}
&\textbf{data } U : Set_1 \textbf{ where}\\
&\quad \mathsf{tree} : U \to U\\
&\quad \mathsf{stream} : U \to U\\
&\quad \_\otimes\_ : U \to U \to U\\
&\quad \lceil\_\rceil : Set \to U
\end{aligned}
$$

$$
\begin{aligned}
&El : U \to Set\\
&El\ (\mathsf{tree}\ a) = Tree\ (El\ a)\\
&El\ (\mathsf{stream}\ a) = Stream\ (El\ a)\\
&El\ (a \otimes b) = El\ a \times El\ b\\
&El\ \lceil A \rceil = A
\end{aligned}
$$

Breadth-first labelling

Programs and WHNFs:

mutual

$$
\begin{array}{l}
\textbf{data}\ El_{\mathrm{P}}\ :\ U \to Set_1\ \textbf{where} \\
\quad \downarrow \quad : El_{\mathrm{W}}\ a \to El_{\mathrm{P}}\ a \\
\quad \mathsf{fst} \quad : El_{\mathrm{P}}\ (a \otimes b) \to El_{\mathrm{P}}\ a \\
\quad \mathsf{snd} \quad : El_{\mathrm{P}}\ (a \otimes b) \to El_{\mathrm{P}}\ b \\
\quad \mathsf{lab} \quad : Tree\ A \to El_{\mathrm{P}}\ (\mathsf{stream}\ \lceil\ Stream\ B\ \rceil) \to \\
\qquad\qquad El_{\mathrm{P}}\ (\mathsf{tree}\ \lceil\ B\ \rceil \otimes \mathsf{stream}\ \lceil\ Stream\ B\ \rceil) \\
\textbf{data}\ El_{\mathrm{W}}\ :\ U \to Set_1\ \textbf{where} \\
\quad \ldots
\end{array}
$$

Breadth-first labelling

Programs and WHNFs:

$$
\begin{array}{l}
\textbf{mutual} \\
\quad \textbf{data}\ El_{\mathrm{P}}\ :\ U \rightarrow Set_1\ \textbf{where} \\
\qquad \ldots \\
\quad \textbf{data}\ El_{\mathrm{W}}\ :\ U \rightarrow Set_1\ \textbf{where} \\
\qquad \begin{array}{lll}
\mathsf{leaf} & : & El_{\mathrm{W}}\ (\mathsf{tree}\ a) \\
\mathsf{node} & : & \infty\ (El_{\mathrm{P}}\ (\mathsf{tree}\ a)) \rightarrow El_{\mathrm{W}}\ a \rightarrow \\
 & & \infty\ (El_{\mathrm{P}}\ (\mathsf{tree}\ a)) \rightarrow El_{\mathrm{W}}\ (\mathsf{tree}\ a) \\
\_{::}\_ & : & El_{\mathrm{W}}\ a \rightarrow \infty\ (El_{\mathrm{P}}\ (\mathsf{stream}\ a)) \rightarrow \\
 & & El_{\mathrm{W}}\ (\mathsf{stream}\ a) \\
\_{,}\_ & : & El_{\mathrm{W}}\ a \rightarrow El_{\mathrm{W}}\ b \rightarrow El_{\mathrm{W}}\ (a \otimes b) \\
\lceil\_\rceil & : & A \rightarrow El_{\mathrm{W}}\ \lceil A \rceil
\end{array}
\end{array}
$$

Breadth-first labelling

Turning programs into WHNFs:

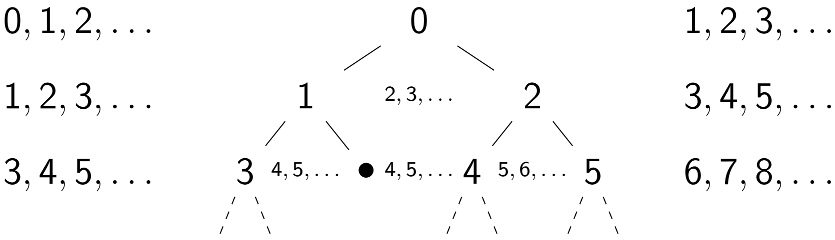
$$
\begin{aligned}
&\mathit{whnf}\ :\ \mathit{El}_{\mathsf{P}}\ a \to \mathit{El}_{\mathsf{W}}\ a\\
&\mathit{whnf}\ (\downarrow w) &&= w\\
&\mathit{whnf}\ (\mathsf{fst}\ p) &&= \mathit{fst}_{\mathsf{W}}\ (\mathit{whnf}\ p)\\
&\mathit{whnf}\ (\mathsf{snd}\ p) &&= \mathit{snd}_{\mathsf{W}}\ (\mathit{whnf}\ p)\\
&\mathit{whnf}\ (\mathsf{lab}\ t\ \mathit{bss}) &&= \mathit{lab}_{\mathsf{W}}\ t\ (\mathit{whnf}\ \mathit{bss})
\end{aligned}
$$

Breadth-first labelling

Turning programs into WHNFs:

$$\begin{aligned}
&\mathit{fst}_{\mathrm{W}} \;:\; \mathit{El}_{\mathrm{W}}\,(a \otimes b) \to \mathit{El}_{\mathrm{W}}\; a\\
&\mathit{fst}_{\mathrm{W}}\,(x,y) \;=\; x\\
&\mathit{snd}_{\mathrm{W}} \;:\; \mathit{El}_{\mathrm{W}}\,(a \otimes b) \to \mathit{El}_{\mathrm{W}}\; b\\
&\mathit{snd}_{\mathrm{W}}\,(x,y) \;=\; y
\end{aligned}$$

Breadth-first labelling



$$
\begin{aligned}
&\mathit{lab}_{\mathsf{W}} \;:\; \mathit{Tree}\ A \to \mathit{El}_{\mathsf{W}}\ (\mathsf{stream}\ \lceil\ \mathit{Stream}\ B\ \rceil) \to \\
&\qquad \mathit{El}_{\mathsf{W}}\ (\mathsf{tree}\ \lceil\ B\ \rceil \otimes \mathsf{stream}\ \lceil\ \mathit{Stream}\ B\ \rceil) \\
&\mathit{lab}_{\mathsf{W}}\ \mathsf{leaf}\ \mathit{bss} = (\mathsf{leaf}, \mathit{bss}) \\
&\mathit{lab}_{\mathsf{W}}\ (\mathsf{node}\ l\ \_\ r)\ (\lceil\ b :: \mathit{bs}\ \rceil :: \mathit{bss}) = \\
&\quad (\mathsf{node}\ (\sharp\ \mathsf{fst}\ x)\ \lceil\ b\ \rceil\ (\sharp\ \mathsf{fst}\ y), \lceil\ \flat\ \mathit{bs}\ \rceil :: \sharp\ \mathsf{snd}\ y) \\
&\quad \textbf{where}\ x = \mathsf{lab}\ (\flat\ l)\ (\flat\ \mathit{bss});\ y = \mathsf{lab}\ (\flat\ r)\ (\mathsf{snd}\ x)
\end{aligned}
$$

Breadth-first labelling

Interpreting programs:

mutual

$$
\begin{array}{lcl}
[\![\_]\!]_W & : & El_W\ a \to El\ a \\
[\![\ \mathsf{leaf}\ ]\!]_W & = & \mathsf{leaf} \\
[\![\ \mathsf{node}\ l\ x\ r\ ]\!]_W & = & \mathsf{node}\ (\sharp\ [\![\ \flat\ l\ ]\!]_P)\ [\![\ x\ ]\!]_W\ (\sharp\ [\![\ \flat\ r\ ]\!]_P) \\
[\![\ x :: xs\ ]\!]_W & = & [\![\ x\ ]\!]_W :: \sharp\ [\![\ \flat\ xs\ ]\!]_P \\
[\![\ (x,y)\ ]\!]_W & = & ([\![\ x\ ]\!]_W, [\![\ y\ ]\!]_W) \\
[\![\ \lceil x \rceil\ ]\!]_W & = & x
\end{array}
$$

$$
\begin{array}{lcl}
[\![\_]\!]_P & : & El_P\ a \to El\ a \\
[\![\ p\ ]\!]_P & = & [\![\ whnf\ p\ ]\!]_W
\end{array}
$$

Breadth-first labelling

$$
\begin{aligned}
\mathit{label}' \;&:\; \mathit{Tree}\ A \to \mathit{Stream}\ B \to \\
&\quad \mathit{El}_{\mathsf{P}}\ (\mathsf{tree}\ \lceil\ B\ \rceil \otimes \mathsf{stream}\ \lceil\ \mathit{Stream}\ B\ \rceil) \\
\mathit{label}'\ t\ \mathit{bs} \;&=\; \mathsf{lab}\ t\ (\downarrow (\lceil\ \mathit{bs}\ \rceil :: \sharp\ \mathsf{snd}\ (\mathit{label}'\ t\ \mathit{bs}))) \\
\mathit{label} \;&:\; \mathit{Tree}\ A \to \mathit{Stream}\ B \to \mathit{Tree}\ B \\
\mathit{label}\ t\ \mathit{bs} \;&=\; [\![\ \mathsf{fst}\ (\mathit{label}'\ t\ \mathit{bs})\ ]\!]_{\mathsf{P}}
\end{aligned}
$$

Problems

Problems

- Large interpretive overhead: loss of sharing.
- Properties not proved automatically.
- Less of a problem if the method is used to make *proofs* guarded.

Proofs

Iterate fusion

$$map : (A \to B) \to Stream\ A \to Stream\ B$$
$$map\ f\ (x :: xs) = f\ x :: \sharp\ map\ f\ (\flat\ xs)$$

$$iterate : (A \to A) \to A \to Stream\ A$$
$$iterate\ f\ x = x :: \sharp\ iterate\ f\ (f\ x)$$

$$\begin{aligned} fusion : {} & (\forall\ x \to h\ (f_1\ x) \equiv f_2\ (h\ x)) \to \\ & \forall\ x \to map\ h\ (iterate\ f_1\ x) \approx iterate\ f_2\ (h\ x) \end{aligned}$$

Iterate fusion

Proof programs:

$$
\begin{aligned}
&\textbf{data}\ \_{\approx_P}\_ \ :\ \mathit{Stream}\ A \to \mathit{Stream}\ A \to \mathit{Set}\ \textbf{where}\\
&\quad \_::\_ \qquad : \forall\ x \to \infty\ (\flat\ xs\ \approx_P\ \flat\ ys) \to\\
&\qquad\qquad\qquad x :: xs\ \approx_P\ x :: ys\\
&\quad \_\approx\langle\_\rangle\_ \ : \forall\ xs \to\\
&\qquad\qquad\qquad xs\ \approx_P\ ys \to ys\ \approx_P\ zs \to xs\ \approx_P\ zs\\
&\quad \_\Box \qquad\ \ : \forall\ xs \to xs\ \approx_P\ xs
\end{aligned}
$$

Soundness:

$$\mathit{sound}_P\ :\ xs\ \approx_P\ ys \to xs \approx ys$$

Iterate fusion

$$
\begin{array}{l}
\mathit{fusion} \;:\; (\forall\, x \to h\,(f_1\;x) \equiv f_2\,(h\,x)) \to \\
\quad \forall\, x \to \mathit{map}\; h\,(\mathit{iterate}\; f_1\; x) \;\approx_{\mathsf{P}}\; \mathit{iterate}\; f_2\,(h\,x) \\
\mathit{fusion}\; \mathit{hyp}\; x \;= \\
\quad \mathit{map}\; h\,(\mathit{iterate}\; f_1\; x) \\
\qquad \approx\langle\; \mathit{by\ definition}\; \rangle \\
\quad h\,x :: \sharp\; \mathit{map}\; h\,(\mathit{iterate}\; f_1\,(f_1\; x)) \\
\qquad \approx\langle\; h\,x :: \sharp\; \mathit{fusion}\; \mathit{hyp}\,(f_1\; x)\; \rangle \\
\quad h\,x :: \sharp\; \mathit{iterate}\; f_2\,(h\,(f_1\; x)) \\
\qquad \approx\langle\; h\,x :: \sharp\; \mathit{iterate\text{-}cong}\; f_2\,(\mathit{hyp}\; x)\; \rangle \\
\quad h\,x :: \sharp\; \mathit{iterate}\; f_2\,(f_2\,(h\,x)) \\
\qquad \approx\langle\; \mathit{by\ definition}\; \rangle \\
\quad \mathit{iterate}\; f_2\,(h\,x) \\
\qquad \square
\end{array}
$$

Wrapping up

Other examples

- Nested applications
  $(\varphi\ (x :: xs)\ =\ x :: \sharp\ \varphi\ (\varphi\ xs))$.
- Destructors (*tail*).
- Non-uniform moduli of production
  (Thue-Morse sequence).

Conclusions

- Ad-hoc.
- Manual.
- Inefficient.
- Useful.

Conclusions

- Ad-hoc.
- Manual.
- Inefficient.
- Useful.

?