Normalisation by evaluation

Sam Lindley

Laboratory for Foundations of Computer Science
The University of Edinburgh

Sam.Lindley@ed.ac.uk

August 11th, 2016

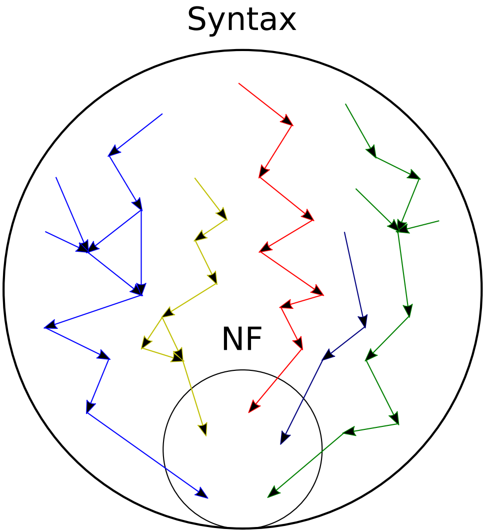
Normalisation and embedded domain specific languages

Why normalisation for embedded DSLs (QDSLs)

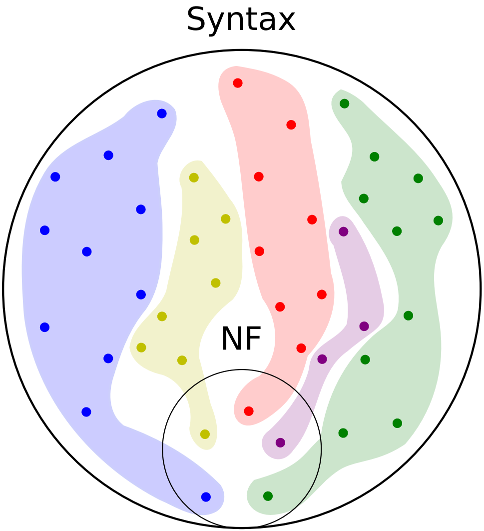
What is normalisation by evaluation (NBE)

How to use NBE for embedded DSLs (EBN)

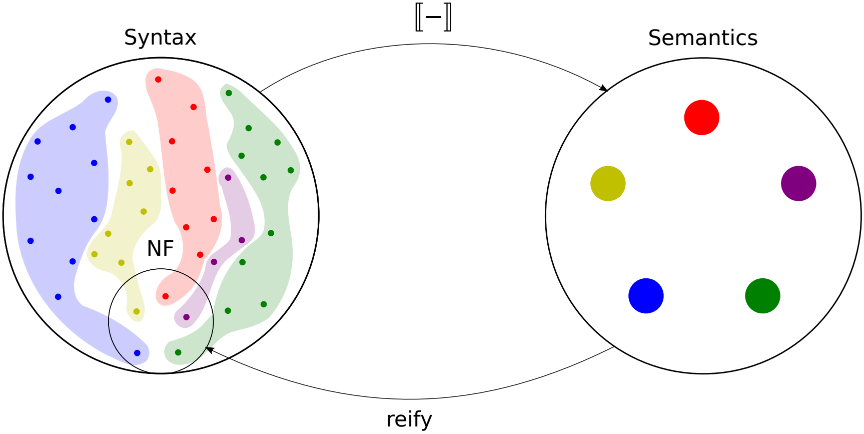
Reduction-based normalisation



Equational normalisation



NBE



$$\boxed{\mathsf{norm} = \mathsf{reify} \circ [\![-]\!]}$$

Why NBE?

- Embedding DSLs (Shayan's lecture tomorrow)
- Partial evaluation (Oleg's finally-tagless optimisations)
- Semantics
- Proof theory
- Type theory
- Efficiency

Simply typed lambda calculus

Typing rules

$$\textsc{Var}\ \frac{}{\Gamma, x : A, \Delta \vdash \mathsf{Var}\ x : A}$$

$$\to\text{-I}\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \mathsf{Lam}\ x\, M : A \to B}$$

$$\to\text{-E}\ \frac{\Gamma \vdash L : A \to B \qquad \Gamma \vdash M : A}{\Gamma \vdash \mathsf{App}\ L\, M : B}$$

Simply typed lambda calculus

Conversions

$$(\mathsf{Lam}\ x\ M)\ N \simeq_\beta M[N/x]$$
$$M \simeq_\eta \mathsf{Lam}\ x\ (M\ x)$$

β-normal form (intensional)

$$\begin{array}{ll} (\mathsf{NF}) & M ::= N \mid \mathsf{Lam}\ x\ M \\ (\mathsf{NE}) & N ::= \mathsf{Var}\ x \mid \mathsf{App}\ N\ M \end{array}$$

$\beta\eta$-long normal form (extensional)

$$\begin{array}{lrl} (\mathsf{NF}) & M_\iota & ::= N_\iota \\ & M_{A\to B} & ::= \mathsf{Lam}\ x\ M_B \\ (\mathsf{NE}) & N_B & ::= \mathsf{Var}\ x \mid \mathsf{App}\ N_{A\to B}\ M_A \end{array}$$

NF = normal form
NE = neutral

Simply typed lambda calculus

Semantics

$$
\begin{aligned}
[\![\iota]\!] &= S \\
[\![A \to B]\!] &= [\![A]\!] \to [\![B]\!]
\end{aligned}
\qquad
\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= [\![M]\!]\rho\ [\![N]\!]\rho
\end{aligned}
$$

S can be any countably infinite set.

Theorem (Soundness)

If $\Gamma \vdash M \simeq N : A$, *then* $[\![M]\!] = [\![N]\!]$.

Theorem (Completeness)

If $[\![M]\!] = [\![N]\!]$, *then* $\Gamma \vdash M \simeq N : A$.

What is reification?

Reification extracts a term from a semantic object by "poking it".

Example 1: $f \in [\![\iota \to \iota]\!]$
Structure of normal forms / parametricity $\Longrightarrow$

$$\mathsf{reify}\ f = \mathsf{Lam}\ x\ (\mathsf{Var}\ x)$$

But this *is not* reification.

Example 2: $g \in [\![\iota \to \iota \to \iota]\!]$
Two possible closed normal forms of this type:

$$\mathsf{Lam}\ x\ (\mathsf{Lam}\ y\ (\mathsf{Var}\ x)) \qquad \text{and} \qquad \mathsf{Lam}\ x\ (\mathsf{Lam}\ y\ (\mathsf{Var}\ y))$$

Pick a suitably *syntactic* interpretation for ι and run g.

- $g\ `x'\ `y' = `x' \Longrightarrow \mathsf{reify}\ g = \mathsf{Lam}\ x\ (\mathsf{Lam}\ y\ (\mathsf{Var}\ x))$
- $g\ `x'\ `y' = `y' \Longrightarrow \mathsf{reify}\ g = \mathsf{Lam}\ x\ (\mathsf{Lam}\ y\ (\mathsf{Var}\ y))$

This *is* reification.

Simply typed lambda calculus

Residualising semantics

$$
\begin{aligned}
[\![\iota]\!] &= \mathsf{NE}_\iota \\
[\![A \to B]\!] &= [\![A]\!] \to [\![B]\!]
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= [\![M]\!]\rho\ [\![N]\!]\rho
\end{aligned}
$$

Simply typed lambda calculus

Extensional NBE

$$
\begin{aligned}
\mathsf{reify}_A &: [\![A]\!] \to \mathsf{NF}_A \\
\mathsf{reify}_\iota\, N &= N \\
\mathsf{reify}_{A\to B}\, f &= \mathsf{Lam}\ x\ (\mathsf{reify}_B\ (f\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x)))), \quad \text{x fresh}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{reflect}_A &: \mathsf{NE}_A \to [\![A]\!] \\
\mathsf{reflect}_\iota\, N &= N \\
\mathsf{reflect}_{A\to B}\, N &= \lambda v.\mathsf{reflect}_B\ (\mathsf{App}\ N\ (\mathsf{reify}_A\ v))
\end{aligned}
$$

$$
\mathsf{norm}_A\, M = \mathsf{reify}_A\ ([\![M]\!]\varnothing)
$$

Type-directed partial evaluation (TDPE)

TDPE is an implementation of NBE in which the object language is a subset of the host language and the residualising semantics coincides with the semantics of that subset of the host-language.

[Danvy, POPL 1996]

Correctness properties for NBE

Theorem (Soundness)

If $M \simeq N$ *then* $[\![M]\!] = [\![N]\!]$.

Theorem (Consistency)

$\mathsf{reify}\ [\![M]\!] \simeq M$

soundness $\wedge$ consistency $\implies$ completeness of the semantics

Proving consistency

- existence of normal forms $\wedge$ soundness $\wedge$
  preservation of normal forms ($\forall M \in \mathsf{NF}. \mathsf{reify}\ [\![M]\!] = M$)
  $\implies$ consistency
- otherwise, consistency is typically proved with logical relations

Simply typed lambda calculus

Intensional residualising semantics

$$
\begin{aligned}
[\![\iota]\!] &= \mathsf{NE}_\iota \\
[\![A \to B]\!] &= ([\![A]\!] \to [\![B]\!]) + \mathsf{NE}_{A \to B}
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= \mathsf{app}\ [\![M]\!]\rho\ [\![N]\!]\rho
\end{aligned}
$$

Intensional NBE

$$
\begin{aligned}
\mathsf{app} &: [\![A \to B]\!] \to [\![A]\!] \to [\![B]\!] \\
\mathsf{app}\ f\ v &= f\ v \\
\mathsf{app}\ N\ v &= \mathsf{App}\ N\ (\mathsf{reify}_A\ v)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{reify}_A &: [\![A]\!] \to \mathsf{NF}_A \\
\mathsf{reify}_A\ N &= N \\
\mathsf{reify}_{A \to B}\ f &= \mathsf{Lam}\ x\ (\mathsf{reify}_B\ (\mathsf{app}\ f\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x)))), \quad \text{x fresh}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{reflect}_A &: \mathsf{NE}_A \to [\![A]\!] \\
\mathsf{reflect}_A\ N &= N
\end{aligned}
$$

$$
\mathsf{norm}_A\ M = \mathsf{reify}_A\ ([\![M]\!]\varnothing)
$$

Simply typed lambda calculus

Intensional residualising semantics

$$[\![\iota]\!] = \mathsf{NE}_\iota$$
$$[\![A \to B]\!] = ([\![A]\!] \to [\![B]\!]) + \mathsf{NE}_{A\to B}$$

$$[\![\mathsf{Var}\ x]\!]\rho = \rho\ x$$
$$[\![\mathsf{Lam}\ x\ M]\!]\rho = \lambda v.[\![M]\!]\rho[x \mapsto v]$$
$$[\![\mathsf{App}\ M\ N]\!]\rho = \mathsf{app}\ [\![M]\!]\rho\ [\![N]\!]\rho$$

Intensional NBE

$$\mathsf{app} : [\![A \to B]\!] \to [\![A]\!] \to [\![B]\!]$$
$$\mathsf{app}\ f\ v = f\ v$$
$$\mathsf{app}\ N\ v = \mathsf{App}\ N\ (\mathsf{reify}\ v)$$

$$\mathsf{reify} : [\![A]\!] \to \mathsf{NF}_A$$
$$\mathsf{reify}\ N = N$$
$$\mathsf{reify}\ f = \mathsf{Lam}\ x\ (\mathsf{reify}\ (\mathsf{app}\ f\ (\mathsf{Var}\ x))), \quad \text{x fresh}$$

$$\mathsf{norm}\ M = \mathsf{reify}\ ([\![M]\!]\varnothing)$$

Untyped lambda calculus

Intensional residualising semantics

$$\boxed{[\![\Lambda]\!] = \mathsf{NE} + ([\![\Lambda]\!] \to [\![\Lambda]\!])}$$

$$\begin{aligned} [\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\ [\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\ [\![\mathsf{App}\ M\ N]\!]\rho &= \mathsf{app}\ [\![M]\!]\rho\ [\![N]\!]\rho \end{aligned}$$

Intensional NBE

$$\begin{aligned} \mathsf{app} &: \boxed{[\![\Lambda]\!] \to [\![\Lambda]\!] \to [\![\Lambda]\!]} \\ \mathsf{app}\ f\ v &= f\ v \\ \mathsf{app}\ N\ v &= \mathsf{App}\ N\ (\mathsf{reify}\ v) \end{aligned}$$

$$\begin{aligned} \mathsf{reify} &: \boxed{[\![\Lambda]\!] \to \mathsf{NF}} \\ \mathsf{reify}\ N &= N \\ \mathsf{reify}\ f &= \mathsf{Lam}\ x\ (\mathsf{reify}\ (\mathsf{app}\ f\ (\mathsf{Var}\ x))), \quad \text{x fresh} \end{aligned}$$

$$\mathsf{norm}\ M = \mathsf{reify}\ ([\![M]\!]\varnothing)$$

Products

Typing rules

$$\frac{\Gamma \vdash M : A_1 \qquad \Gamma \vdash M : A_2}{\Gamma \vdash \mathsf{Pair}\ M\ N : A_1 \times A_2}\ \times\text{-I}$$

$$\frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \mathsf{Proj}_i\ M : A_i}\ \times\text{-E}$$

Products

Conversions

$$\begin{aligned}
\mathsf{App}\ (\mathsf{Lam}\ x\ M)\ N &\simeq_\beta M[N/x] \\
\mathsf{Proj}_i\ (\mathsf{Pair}\ M_1\ M_2) &\simeq_\beta M_i \\
M &\simeq_\eta \mathsf{Lam}\ x\ (\mathsf{App}\ M\ x) \\
M &\simeq_\eta \mathsf{Pair}\ (\mathsf{Proj}_1\ M)\ (\mathsf{Proj}_2\ M)
\end{aligned}$$

$\beta\eta$-long normal form

$$\begin{aligned}
&(\mathsf{NF}) & M_\iota &::= N_\iota \\
& & M_{A\to B} &::= \mathsf{Lam}\ x\ M_B \\
& & M_{A\times B} &::= \mathsf{Pair}\ M_A\ M_B \\
&(\mathsf{NE}) & N_B &::= \mathsf{Var}\ x \mid \mathsf{App}\ N_{A\to B}\ M_A \mid \mathsf{Proj}_1\ N_{B\times A} \mid \mathsf{Proj}_2\ N_{A\times B}
\end{aligned}$$

Products

Semantics

$$
\begin{aligned}
[\![\iota]\!] &= S \\
[\![A \to B]\!] &= [\![A]\!] \to [\![B]\!] \\
\boxed{[\![A \times B]\!] = [\![A]\!] \times [\![B]\!]}
\end{aligned}
\qquad
\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= [\![M]\!]\rho\ [\![N]\!]\rho \\
[\![\mathsf{Pair}\ M\ N]\!]\rho &= ([\![M]\!]\rho, [\![N]\!]\rho) \\
[\![\mathsf{Proj}_i\ M]\!]\rho &= ([\![M]\!]\rho).i
\end{aligned}
$$

S can be any countably infinite set.

Products

Residualising semantics

$$
\begin{aligned}
[\![\iota]\!] &= \mathsf{NE}_\iota \\
[\![A \to B]\!] &= [\![A]\!] \to [\![B]\!] \\
[\![A \times B]\!] &= [\![A]\!] \times [\![B]\!]
\end{aligned}
\qquad
\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= [\![M]\!]\rho\ [\![N]\!]\rho \\
[\![\mathsf{Pair}\ M\ N]\!]\rho &= ([\![M]\!]\rho, [\![N]\!]\rho) \\
[\![\mathsf{Proj}_i\ M]\!]\rho &= ([\![M]\!]\rho).i
\end{aligned}
$$

Products

Extensional NBE

$$\begin{aligned}
\mathsf{reify}_A &: [\![A]\!] \to \mathsf{NF}_A \\
\mathsf{reify}_\iota\ N &= N \\
\mathsf{reify}_{A\to B}\ f &= \mathsf{Lam}\ x\ (\mathsf{reify}_B\ (f\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x)))), \quad \text{x fresh} \\
\mathsf{reify}_{A\times B}\ p &= \mathsf{Pair}\ (\mathsf{reify}_A\ p.1)\ (\mathsf{reify}_B\ p.2)
\end{aligned}$$

$$\begin{aligned}
\mathsf{reflect}_A &: \mathsf{NE}_A \to [\![A]\!] \\
\mathsf{reflect}_\iota\ N &= N \\
\mathsf{reflect}_{A\to B}\ N &= \lambda v.\mathsf{reflect}_B\ (\mathsf{App}\ N\ (\mathsf{reify}_A\ v)) \\
\mathsf{reflect}_{A\times B}\ N &= (\mathsf{reflect}_A\ (\mathsf{Proj}_1\ N), \mathsf{reflect}_B\ (\mathsf{Proj}_2\ N))
\end{aligned}$$

$$\mathsf{norm}_A\ M = \mathsf{reify}_A\ ([\![M]\!]\varnothing)$$

Sums

Typing rules

+-I

$$\frac{\Gamma \vdash M : A_i}{\Gamma \vdash \mathsf{Inj}_i\ M : A_1 + A_2}$$

+-E

$$\frac{\Gamma \vdash M : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash N_1 : C \qquad \Gamma, x_2 : A_2 \vdash N_2 : C}{\Gamma \vdash \mathsf{Case}\ M\ x_1\ N_1\ x_2\ N_2 : C}$$

Sums

Conversions

$$
\begin{aligned}
\mathsf{App}\ (\mathsf{Lam}\ x\ M)\ N &\simeq_\beta M[N/x] \\
\mathsf{Proj}_i\ (\mathsf{Pair}\ M_1\ M_2) &\simeq_\beta M_i \\
\mathsf{Case}\ (\mathsf{Inj}_i\ M)\ x_1\ N_1\ x_2\ N_2 &\simeq_\beta N_i[M/x_i] \\
M &\simeq_\eta \mathsf{Lam}\ x\ (\mathsf{App}\ M\ x) \\
M &\simeq_\eta \mathsf{Pair}\ (\mathsf{Proj}_1\ M)\ (\mathsf{Proj}_2\ M) \\
N[M/z] &\simeq_\eta \mathsf{Case}\ M\ x_1\ N[\mathsf{Inj}_1\ x_1/z]\ x_2\ N[\mathsf{Inj}_2\ x_2/z]
\end{aligned}
$$

Extensional normalisation with sums is hard due to the unruly η-rule.
[Ghani, TLCA 1995; Altenkirch et al., LICS 2001;
Balat et al., POPL 2004; Lindley, TLCA 2007; Scherer, TLCA 2015]

Sums

Semantics

$$\llbracket \mathsf{Var}\ x \rrbracket \rho = \rho\ x$$

$$\begin{aligned}
\llbracket \iota \rrbracket &= S & \llbracket \mathsf{Lam}\ x\ M \rrbracket \rho &= \lambda v. \llbracket M \rrbracket \rho[x \mapsto v] \\
\llbracket A \to B \rrbracket &= \llbracket A \rrbracket \to \llbracket B \rrbracket & \llbracket \mathsf{App}\ M\ N \rrbracket \rho &= \llbracket M \rrbracket \rho\ \llbracket N \rrbracket \rho \\
\llbracket A \times B \rrbracket &= \llbracket A \rrbracket \times \llbracket B \rrbracket & \llbracket \mathsf{Pair}\ M\ N \rrbracket \rho &= (\llbracket M \rrbracket \rho, \llbracket N \rrbracket \rho) \\
\llbracket A + B \rrbracket &= \llbracket A \rrbracket + \llbracket B \rrbracket & \llbracket \mathsf{Proj}_i\ M \rrbracket \rho &= (\llbracket M \rrbracket \rho).i
\end{aligned}$$

$$\llbracket \mathsf{Inj}_i\ M \rrbracket \rho = (i, \llbracket M \rrbracket \rho)$$

$$\llbracket \mathsf{Case}\ M\ x_1\ N_1\ x_2\ N_2 \rrbracket \rho = \begin{cases} \llbracket N_1 \rrbracket \rho[x_1 \mapsto v], & \text{if } \llbracket M \rrbracket \rho = (1, v) \\ \llbracket N_2 \rrbracket \rho[x_2 \mapsto v], & \text{if } \llbracket M \rrbracket \rho = (2, v) \end{cases}$$

S can be any countably infinite set.

Despite the unruly η-rule, the semantics for sums is straightforward.

Sums

Extensional NBE?

$$
\begin{aligned}
\mathsf{reify}_A &: [\![A]\!] \to \mathsf{NF}_A \\
\mathsf{reify}_\iota\ N &= N \\
\mathsf{reify}_{A\to B}\ f &= \mathsf{Lam}\ x\ (\mathsf{reify}_B\ (f\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x)))), \quad \text{x fresh} \\
\mathsf{reify}_{A\times B}\ p &= \mathsf{Pair}\ (\mathsf{reify}_A\ p.1)\ (\mathsf{reify}_B\ p.2) \\
\mathsf{reify}_{A_1+A_2}\ (i, v) &= \mathsf{Inj}_i\ (\mathsf{reify}_{A_i}\ v)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{reflect}_A &: \mathsf{NE}_A \to [\![A]\!] \\
\mathsf{reflect}_\iota\ N &= N \\
\mathsf{reflect}_{A\to B}\ N &= \lambda v.\mathsf{reflect}_B\ (\mathsf{App}\ N\ (\mathsf{reify}_A\ v)) \\
\mathsf{reflect}_{A\times B}\ N &= (\mathsf{reflect}_A\ (\mathsf{Proj}_1\ N), \mathsf{reflect}_B\ (\mathsf{Proj}_2\ N)) \\
\mathsf{reflect}_{A_1+A_2}\ N &= ???
\end{aligned}
$$

Computations

Typing rules

$$\textsf{T}\text{-I}\;\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{Val}\ M : \mathsf{T} A}
\qquad
\textsf{T}\text{-E}\;\frac{\Gamma \vdash M : \mathsf{T} A \qquad \Gamma, x : A \vdash N : \mathsf{T} B}{\Gamma \vdash \mathsf{Let}\ x\ M\ N : \mathsf{T} B}$$

Computations

Conversions

$$
\begin{aligned}
\mathsf{App}\ (\mathsf{Lam}\ x\ M)\ N &\simeq_\beta M[N/x] \\
\mathsf{Proj}_i\ (\mathsf{Pair}\ M_1\ M_2) &\simeq_\beta M_i \\
\mathsf{Let}\ x\ (\mathsf{Val}\ M)\ N &\simeq_\beta N[M/x] \\
\mathsf{Let}\ y\ (\mathsf{Let}\ x\ L\ M)\ N &\simeq_\gamma \mathsf{Let}\ x\ L\ (\mathsf{Let}\ y\ M\ N) \\
M &\simeq_\eta \mathsf{Lam}\ x\ (\mathsf{App}\ M\ x) \\
M &\simeq_\eta \mathsf{Pair}\ (\mathsf{Proj}_1\ M)\ (\mathsf{Proj}_2\ M) \\
M &\simeq_\eta \mathsf{Let}\ x\ M\ (\mathsf{Val}\ x)
\end{aligned}
$$

$\beta\eta$-long normal form

$$
\begin{array}{lrl}
(\mathsf{NF}) & M_\iota & ::= N_\iota \\
 & M_{A\to B} & ::= \mathsf{Lam}\ x\ M_B \\
 & M_{A\times B} & ::= \mathsf{Pair}\ M_A\ M_B \\
 & M_{\mathsf{T}B} & ::= \mathsf{Val}\ M_A \mid \mathsf{Let}\ x\ N_{\mathsf{T}A}\ M_{\mathsf{T}B} \\
(\mathsf{NE}) & N_B & ::= \mathsf{Var}\ x \mid \mathsf{App}\ N_{A\to B}\ M_A \mid \mathsf{Proj}_1\ N_{B\times A} \mid \mathsf{Proj}_2\ N_{A\times B}
\end{array}
$$

Computations

Semantics

$$
\begin{aligned}
[\![\iota]\!] &= S \\
[\![A \to B]\!] &= [\![A]\!] \to [\![B]\!] \\
[\![A \times B]\!] &= [\![A]\!] \times [\![B]\!] \\
[\![\mathsf{T}A]\!] &= \mathsf{T}[\![A]\!]
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= [\![M]\!]\rho\ [\![N]\!]\rho \\
[\![\mathsf{Pair}\ M\ N]\!]\rho &= ([\![M]\!]\rho, [\![N]\!]\rho) \\
[\![\mathsf{Proj}_i\ M]\!]\rho &= ([\![M]\!]\rho).i \\
[\![\mathsf{Val}\ M]\!]\rho &= \mathsf{return}\ [\![M]\!]\rho \\
[\![\mathsf{Let}\ x\ M\ N]\!]\rho &= [\![M]\!]\rho \gg\!= \lambda x.[\![N]\!]\rho
\end{aligned}
$$

S can be any countably infinite set.
T can be any *monad*:

$$
\begin{aligned}
\mathsf{T} &: \star \to \star \\
\mathsf{return} &: A \to \mathsf{T}A \\
(\gg\!=) &: \mathsf{T}A \to (A \to \mathsf{T}B) \to \mathsf{T}B
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{return}\ v \gg\!= f &= f\ v \\
(c \gg\!= f) \gg\!= g &= c \gg\!= (\lambda x.f\ x \gg\!= g) \\
c &= c \gg\!= (\lambda x.\mathsf{return}\ x)
\end{aligned}
$$

Digression: pronouncing the word “monad”

Is it “moanad” or “monnad”?

Does “monad” rhyme with “gonad” or does its first syllable rhyme with the first syllable of “monoid”?

A monad is a monoid in the category of endofunctors!

Computations

Extensional NBE?

$$
\begin{aligned}
\mathsf{reify}_A &: [\![A]\!] \to \mathsf{NF}_A \\
\mathsf{reify}_\iota\ N &= N \\
\mathsf{reify}_{A\to B}\ f &= \mathsf{Lam}\ x\ (\mathsf{reify}_B\ (f\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x)))), \quad \text{x fresh} \\
\mathsf{reify}_{A\times B}\ p &= \mathsf{Pair}\ (\mathsf{reify}_A\ p.1)\ (\mathsf{reify}_B\ p.2) \\
\mathsf{reify}_{\mathsf{T}A}\ c &= ??? \quad \text{(need to collect let bindings here)}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{reflect}_A &: \mathsf{NE}_A \to [\![A]\!] \\
\mathsf{reflect}_\iota\ N &= N \\
\mathsf{reflect}_{A\to B}\ N &= \lambda v.\mathsf{reflect}_B\ (N\ (\mathsf{reify}_A\ v)) \\
\mathsf{reflect}_{A\times B}\ N &= (\mathsf{reflect}_A\ (\mathsf{Proj}_1\ N), \mathsf{reflect}_B\ (\mathsf{Proj}_2\ N)) \\
\mathsf{reflect}_{\mathsf{T}A}\ N &= ??? \quad \text{(need to register a let binding for } N \text{ here)}
\end{aligned}
$$

Residualising monads

To support reification the monad T must include sufficient syntactic data in order to keep track of let bindings.

A *residualising monad* is a monad equipped with operations

$$\begin{aligned} \mathsf{bind} &: \mathsf{NE}_{\mathsf{T}A} \to \mathsf{T}\ \mathsf{V}_A && \text{(register a let binding)} \\ \mathsf{collect} &: \mathsf{T}\ \mathsf{NF}_{\mathsf{T}A} \to \mathsf{NF}_{\mathsf{T}A} && \text{(collect let bindings)} \end{aligned}$$

satisfying the equations:

$$\begin{aligned} \mathsf{collect}\ (\mathsf{return}\ M) &= M \\ \mathsf{collect}\ (\mathsf{bind}\ N \gg\!= f) &= \mathsf{Let}\ x\ N\ (\mathsf{collect}\ (f\ x)), \quad x \text{ fresh} \end{aligned}$$

where V_A is the set of object variables of type A.

Residualising monads

Continuation monad

$$\mathsf{T}A = (A \to \mathsf{NF}) \to \mathsf{NF}$$

$$\begin{aligned} \mathsf{return}\ v &= \lambda k.k\ v \\ c \gg= f &= \lambda k.c\ (\lambda x.f\ x\ k) \end{aligned}$$

$$\begin{aligned} \mathsf{bind}\ N &= \lambda k.\mathsf{Let}\ x\ N\ (k\ x), \quad x \text{ fresh} \\ \mathsf{collect}\ c &= c\ \mathsf{id} \end{aligned}$$

Free monad over a list of let bindings

$$\mathsf{T}A = \mu X.\mathsf{Val}\ A + \mathsf{Let}\ x\ \mathsf{NE}_{\mathsf{T}B}\ X$$

$$\begin{aligned} \mathsf{return}\ v &= \mathsf{Val}\ v \\ \mathsf{Val}\ v \gg= f &= f\ v \\ \mathsf{Let}\ x\ N\ c \gg= f &= \mathsf{Let}\ x\ N\ (c \gg= f) \end{aligned}$$

$$\begin{aligned} \mathsf{bind}\ N &= \mathsf{Let}\ x\ N\ (\mathsf{Val}\ x), \quad x \text{ fresh} \\ \mathsf{collect}\ (\mathsf{Val}\ M) &= M \\ \mathsf{collect}\ (\mathsf{Let}\ x\ N\ c) &= \mathsf{Let}\ x\ N\ (\mathsf{collect}\ c) \end{aligned}$$

Computations

Residualising semantics

$$
\begin{aligned}
[\![\iota]\!] &= \mathsf{NE}_\iota \\
[\![A \to B]\!] &= [\![A]\!] \to [\![B]\!] \\
[\![A \times B]\!] &= [\![A]\!] \times [\![B]\!] \\
[\![\mathsf{T} A]\!] &= \mathsf{T}[\![A]\!]
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= [\![M]\!]\rho\ [\![N]\!]\rho \\
[\![\mathsf{Pair}\ M\ N]\!]\rho &= ([\![M]\!]\rho, [\![N]\!]\rho) \\
[\![\mathsf{Proj}_i\ M]\!]\rho &= ([\![M]\!]\rho).i \\
[\![\mathsf{Val}\ M]\!]\rho &= \mathsf{return}\ [\![M]\!]\rho \\
[\![\mathsf{Let}\ x\ M\ N]\!]\rho &= [\![M]\!]\rho \gg\!= \lambda x.[\![N]\!]\rho
\end{aligned}
$$

T can be any residualising monad.

Computations

Extensional NBE

$$\begin{aligned}
\mathsf{reify}_A &: [\![A]\!] \to \mathsf{NF}_A \\
\mathsf{reify}_\iota\, N &= N \\
\mathsf{reify}_{A\to B}\, f &= \mathsf{Lam}\ x\ (\mathsf{reify}_B\ (f\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x)))), \quad \text{x fresh} \\
\mathsf{reify}_{A\times B}\, p &= \mathsf{Pair}\ (\mathsf{reify}_A\ p.1)\ (\mathsf{reify}_B\ p.2) \\
\mathsf{reify}_{\mathsf{T}A}\, c &= \mathsf{collect}\ (c \gg\!= \lambda v.\mathsf{return}\ (\mathsf{reify}_A\ v))
\end{aligned}$$

$$\begin{aligned}
\mathsf{reflect}_A &: \mathsf{NE}_A \to [\![A]\!] \\
\mathsf{reflect}_\iota\, N &= N \\
\mathsf{reflect}_{A\to B}\, N &= \lambda v.\mathsf{reflect}_B\ (N\ (\mathsf{reify}_A\ v)) \\
\mathsf{reflect}_{A\times B}\, N &= (\mathsf{reflect}_A\ (\mathsf{Proj}_1\ N), \mathsf{reflect}_B\ (\mathsf{Proj}_2\ N)) \\
\mathsf{reflect}_{\mathsf{T}A}\, N &= \mathsf{bind}\ N \gg\!= \lambda x.\mathsf{return}\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x))
\end{aligned}$$

$$\mathsf{norm}_A\, M = \mathsf{reify}_A\ ([\![M]\!]\varnothing)$$

[Filinski, TLCA 2001; my PhD thesis]

Computational sums

Typing rules

+-I
$$\frac{\Gamma \vdash M : A_i}{\Gamma \vdash \mathsf{Inj}_i\, M : A_1 + A_2}$$

+-E
$$\frac{\Gamma \vdash M : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash N_1 : \mathsf{T}B \qquad \Gamma, x_2 : A_2 \vdash N_2 : \mathsf{T}B}{\Gamma \vdash \mathsf{Case}\ M\ x_1\ N_1\ x_2\ N_2 : \mathsf{T}B}$$

Computational sums

Extensional NBE?

$$
\begin{aligned}
\mathsf{reify}_A &: [\![A]\!] \to \mathsf{NF}_A \\
\mathsf{reify}_\iota\ N &= N \\
\mathsf{reify}_{A\to B}\ f &= \mathsf{Lam}\ x\ (\mathsf{reify}_B\ (f\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x)))), \quad \text{x fresh} \\
\mathsf{reify}_{A\times B}\ p &= \mathsf{Pair}\ (\mathsf{reify}_A\ p.1)\ (\mathsf{reify}_B\ p.2) \\
\mathsf{reify}_{A_1+A_2}\ (i,v) &= \mathsf{Inj}_i\ (\mathsf{reify}_{A_i}\ v) \\
\mathsf{reify}_{\mathsf{T}A}\ c &= \mathsf{collect}\ (c \gg\!= \lambda x.\mathsf{return}\ (\mathsf{reify}_A\ x))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{reflect}_A &: \mathsf{NE}_A \to [\![A]\!] \\
\mathsf{reflect}_\iota\ N &= N \\
\mathsf{reflect}_{A\to B}\ N &= \lambda v.\mathsf{reflect}_B\ (\mathsf{App}\ N\ (\mathsf{reify}_A\ v)) \\
\mathsf{reflect}_{A\times B}\ N &= (\mathsf{reflect}_A\ (\mathsf{Proj}_1\ N), \mathsf{reflect}_B\ (\mathsf{Proj}_2\ N)) \\
\mathsf{reflect}_{A_1+A_2}\ N &= \mathsf{binds}\ N \gg\!= ??? \quad \text{(need a computation type here)} \\
\mathsf{reflect}_{\mathsf{T}A}\ N &= \mathsf{bind}\ N \gg\!= \lambda x.\mathsf{return}\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x))
\end{aligned}
$$

Fix

- change the type of $\mathsf{reflect}_A$ to $\mathsf{NE}_A \to \mathsf{T}[\![A]\!]$
- restrict function types to be of the form $A \to \mathsf{T}B$

Call-by-value

Typing rules

$$\frac{\Gamma, x : A \vdash M : \mathsf{T}\, B}{\Gamma \vdash \mathsf{Lam}\; x\, M : A \to \mathsf{T}\, B}\;\to\text{-I}$$

$$\frac{\Gamma \vdash L : A \to \mathsf{T}\, B \qquad \Gamma \vdash M : A}{\Gamma \vdash \mathsf{App}\; L\, M : \mathsf{T}\, B}\;\to\text{-E}$$

Call-by-value computational sums

Conversions

$$
\begin{aligned}
\mathsf{App}\ (\mathsf{Lam}\ x\ M)\ N &\simeq_\beta M[N/x] \\
\mathsf{Proj}_i\ (\mathsf{Pair}\ M_1\ M_2) &\simeq_\beta M_i \\
\mathsf{Case}\ (\mathsf{Inj}_i\ M)\ x_1\ N_1\ x_2\ N_2 &\simeq_\beta N_i[M/x_i] \\
\mathsf{Let}\ x\ (\mathsf{Val}\ M)\ N &\simeq_\beta N[M/x] \\
\mathsf{Let}\ y\ (\mathsf{Case}\ L\ x_1\ M_1\ x_2\ M_2)\ N &\simeq_\gamma \mathsf{Case}\ L\ x_1\ (\mathsf{Let}\ y\ M_1\ N)\ x_2\ (\mathsf{Let}\ y\ M_2\ N) \\
\mathsf{Let}\ y\ (\mathsf{Let}\ x\ L\ M)\ N &\simeq_\gamma \mathsf{Let}\ x\ L\ (\mathsf{Let}\ y\ M\ N) \\
M &\simeq_\eta \mathsf{Lam}\ x\ (\mathsf{App}\ M\ x) \\
M &\simeq_\eta \mathsf{Pair}\ (\mathsf{Proj}_1\ M)\ (\mathsf{Proj}_2\ M) \\
M &\simeq_\eta \mathsf{Case}\ M\ x_1\ (\mathsf{Val}\ (\mathsf{Inj}_1\ x_1))\ x_2\ (\mathsf{Val}\ (\mathsf{Inj}_2\ x_2)) \\
M &\simeq_\eta \mathsf{Let}\ x\ M\ (\mathsf{Val}\ x)
\end{aligned}
$$

The restriction to call-by-value computational sums weakens the unruly η-rule.

Call-by-value computational sums

$\beta\eta$-long normal form

$$
\begin{array}{llcl}
(\mathsf{NF}) & M_\iota & ::= & N_\iota \\
& M_{A \to \mathsf{T}B} & ::= & \mathsf{Lam}\ x\ M_{\mathsf{T}B} \\
& M_{A \times B} & ::= & \mathsf{Pair}\ M_A\ M_B \\
& M_{A_1 + A_2} & ::= & \mathsf{Inj}_i\ M_{A_i} \\
& M_{\mathsf{T}B} & ::= & \mathsf{Val}\ M_{\overset{B}{\cancel{A}}} \mid \mathsf{Let}\ x\ N_{\mathsf{T}A}\ M_{\mathsf{T}B} \\
& & \mid & \mathsf{Case}\ N_{A_1 + A_2}\ x_1\ M_{\mathsf{T}B}\ x_2\ M'_{\mathsf{T}B} \\
(\mathsf{NE}) & N_B & ::= & \mathsf{Var}\ x \mid \mathsf{App}\ N_{A \to B}\ M_A \mid \mathsf{Proj}_1\ N_{B \times A} \mid \mathsf{Proj}_2\ N_{A \times B}
\end{array}
$$

Call-by-value computational sums

Semantics

$$
\begin{aligned}
[\![\iota]\!] &= S \\
[\![A \to \mathsf{T} B]\!] &= [\![A]\!] \to [\![\mathsf{T} B]\!] \\
[\![A \times B]\!] &= [\![A]\!] \times [\![B]\!] \\
[\![A + B]\!] &= [\![A]\!] + [\![B]\!] \\
[\![\mathsf{T} A]\!] &= \mathsf{T}[\![A]\!]
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= [\![M]\!]\rho\ [\![N]\!]\rho \\
[\![\mathsf{Pair}\ M\ N]\!]\rho &= ([\![M]\!]\rho, [\![N]\!]\rho) \\
[\![\mathsf{Proj}_i\ M]\!]\rho &= ([\![M]\!]\rho).i \\
[\![\mathsf{Inj}_i\ M]\!]\rho &= (i, [\![M]\!]\rho) \\
[\![\mathsf{Val}\ M]\!]\rho &= \mathsf{return}\ [\![M]\!]\rho \\
[\![\mathsf{Let}\ x\ M\ N]\!]\rho &= [\![M]\!]\rho \gg\!= \lambda x.[\![N]\!]\rho
\end{aligned}
$$

$$
[\![\mathsf{Case}\ M\ x_1\ N_1\ x_2\ N_2]\!]\rho = \begin{cases} [\![N_1]\!]\rho[x_1 \mapsto v], & \text{if } [\![M]\!]\rho = (1, v) \\ [\![N_2]\!]\rho[x_2 \mapsto v], & \text{if } [\![M]\!]\rho = (2, v) \end{cases}
$$

S can be any countably infinite set. T can be any monad.

Residualising sum monads

A *residualising sum monad* is a monad equipped with operations

$$
\begin{array}{rll}
\mathsf{bind} : \mathsf{NE}_{\mathsf{T}A} \to \mathsf{T}\ \mathsf{V}_A & & \text{(register let binding)} \\
\mathsf{binds} : \mathsf{NE}_{A+B} \to \mathsf{T}\ (\mathsf{V}_A + \mathsf{V}_B) & & \text{(register case binding)} \\
\mathsf{collect} : \mathsf{T}\ \mathsf{NF}_{\mathsf{T}A} \to \mathsf{NF}_{\mathsf{T}A} & & \text{(collect bindings)}
\end{array}
$$

satisfying the equations:

$$
\begin{array}{rcll}
\mathsf{collect}\ (\mathsf{return}\ M) & = & M & \\
\mathsf{collect}\ (\mathsf{bind}\ N \gg= f) & = & \mathsf{Let}\ x\ N\ (\mathsf{collect}\ (f\ x)), & x \text{ fresh} \\
\mathsf{collect}\ (\mathsf{binds}\ N \gg= f) & = & \mathsf{Case}\ N\ x_1\ (\mathsf{collect}\ (f\ (1, x_1))) & \\
 & & \qquad\qquad\ \ x_2\ (\mathsf{collect}\ (f\ (2, x_2))), & x_1, x_2 \text{ fresh}
\end{array}
$$

Residualising sum monads

Continuation monad

$$\mathsf{T}A = (A \to \mathsf{NF}) \to \mathsf{NF}$$

$$\begin{aligned}
\mathsf{return}\ v &= \lambda k.k\ v \\
c \gg\!= f &= \lambda k.c\ (\lambda x.f\ x\ k)
\end{aligned}$$

$$\begin{aligned}
\mathsf{bind}\ N &= \lambda k.\mathsf{Let}\ x\ N\ (k\ x), && x \text{ fresh} \\
\mathsf{binds}\ N &= \lambda k.\mathsf{Case}\ N\ x_1\ (k\ (1, x_1)) \\
&\qquad\qquad\qquad\quad x_2\ (k\ (2, x_2)), && x_1, x_2 \text{ fresh} \\
\mathsf{collect}\ c &= c\ \mathsf{id}
\end{aligned}$$

Residualising sum monads

Free monad over a tree of let and case bindings

$$\mathsf{T}A = \mu X.\mathsf{Val}\ A + \mathsf{Let}\ x\ \mathsf{NE}_{\mathsf{T}B}\ X + \mathsf{Case}\ \mathsf{NE}_{A_1+A_2}\ x_1\ X\ x_2\ X$$

$$\begin{aligned}
\mathsf{return}\ v &= v \\
\mathsf{Val}\ v \gg\!= f &= f\ v \\
\mathsf{Let}\ x\ N\ c \gg\!= f &= \mathsf{Let}\ x\ N\ (c \gg\!= f) \\
\mathsf{Case}\ N\ x_1\ c_1\ x_2\ c_2 \gg\!= f &= \mathsf{Case}\ N\ x_1\ (c_1 \gg\!= f)\ x_2\ (c\,2 \gg\!= f)
\end{aligned}$$

$$\begin{aligned}
\mathsf{bind}\ N &= \mathsf{Let}\ x\ N\ (\mathsf{Val}\ x), && x \text{ fresh} \\
\mathsf{binds}\ N &= \mathsf{Case}\ N\ x_1\ (\mathsf{Val}\ (1, x_1)) \\
&\qquad x_2\ (\mathsf{Val}\ (2, x_2)), && x_1, x_2 \text{ fresh} \\
\mathsf{collect}\ (\mathsf{Val}\ M) &= M \\
\mathsf{collect}\ (\mathsf{Let}\ x\ N\ c) &= \mathsf{Let}\ x\ N\ (\mathsf{collect}\ c) \\
\mathsf{collect}\ (\mathsf{Case}\ N\ x_1\ c_1\ x_2\ c_2) &= \mathsf{Case}\ N\ x_1\ (\mathsf{collect}\ c_1)\ x_2\ (\mathsf{collect}\ c_2)
\end{aligned}$$

Call-by-value computational sums

Residualising semantics

$$\begin{aligned}
[\![\iota]\!] &= \mathsf{NE}_\iota \\
[\![A \to \mathsf{T}B]\!] &= [\![A]\!] \to [\![\mathsf{T}B]\!] \\
[\![A \times B]\!] &= [\![A]\!] \times [\![B]\!] \\
[\![A + B]\!] &= [\![A]\!] + [\![B]\!] \\
[\![\mathsf{T}A]\!] &= \mathsf{T}[\![A]\!]
\end{aligned}$$

$$\begin{aligned}
[\![\mathsf{Var}\ x]\!]\rho &= \rho\ x \\
[\![\mathsf{Lam}\ x\ M]\!]\rho &= \lambda v.[\![M]\!]\rho[x \mapsto v] \\
[\![\mathsf{App}\ M\ N]\!]\rho &= [\![M]\!]\rho\ [\![N]\!]\rho \\
[\![\mathsf{Pair}\ M\ N]\!]\rho &= ([\![M]\!]\rho, [\![N]\!]\rho) \\
[\![\mathsf{Proj}_i\ M]\!]\rho &= ([\![M]\!]\rho).i \\
[\![\mathsf{Inj}_i\ M]\!]\rho &= (i, [\![M]\!]\rho) \\
[\![\mathsf{Val}\ M]\!]\rho &= \mathsf{return}\ [\![M]\!]\rho \\
[\![\mathsf{Let}\ x\ M\ N]\!]\rho &= [\![M]\!]\rho \gg\!= \lambda x.[\![N]\!]\rho
\end{aligned}$$

$$[\![\mathsf{Case}\ M\ x_1\ N_1\ x_2\ N_2]\!]\rho = \begin{cases} [\![N_1]\!]\rho[x_1 \mapsto v], & \text{if } [\![M]\!]\rho = (1, v) \\ [\![N_2]\!]\rho[x_2 \mapsto v], & \text{if } [\![M]\!]\rho = (2, v) \end{cases}$$

T can be any residualising sum monad.

Call-by-value computational sums

Extensional NBE

$$
\begin{aligned}
\mathsf{reify}_A &: [\![A]\!] \to \mathsf{NF}_A \\
\mathsf{reify}_\iota\ N &= N \\
\mathsf{reify}_{A \to \mathsf{T}B}\ f &= \mathsf{Lam}\ x\ (\mathsf{reify}_{\mathsf{T}B}\ (\mathsf{reflect}_A\ (\mathsf{Var}\ x) \gg\!= f)), \quad \text{x fresh} \\
\mathsf{reify}_{A \times B}\ p &= \mathsf{Pair}\ (\mathsf{reify}_A\ p.1)\ (\mathsf{reify}_B\ p.2) \\
\mathsf{reify}_{A_1 + A_2}\ (i, v) &= \mathsf{Inj}_i\ (\mathsf{reify}_{A_i}\ v) \\
\mathsf{reify}_{\mathsf{T}A}\ c &= \mathsf{collect}\ (c \gg\!= \lambda x.\mathsf{return}\ (\mathsf{reify}_A\ x))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{reflect}_A &: \mathsf{NE}_A \to \mathsf{T}[\![A]\!] \\
\mathsf{reflect}_\iota\ N &= \mathsf{return}\ N \\
\mathsf{reflect}_{A \to \mathsf{T}B}\ N &= \mathsf{return}\ (\lambda v.\mathsf{reflect}_{\mathsf{T}B}\ (\mathsf{App}\ N\ (\mathsf{reify}_A\ v)) \gg\!= \mathsf{id}) \\
\mathsf{reflect}_{A \times B}\ N &= \mathsf{reflect}_A\ (\mathsf{Proj}_1\ N) \gg\!= \lambda x. \\
&\ \mathsf{reflect}_B\ (\mathsf{Proj}_2\ N) \gg\!= \lambda y.\mathsf{return}\ (x, y) \\
\mathsf{reflect}_{A_1 + A_2}\ N &= \mathsf{binds}\ N \gg\!= \lambda(i, x_i).\mathsf{reflect}_{A_i}\ (\mathsf{Var}\ x_i) \gg\!= \lambda v.\mathsf{return}\ (i, v) \\
\mathsf{reflect}_{\mathsf{T}A}\ N &= \mathsf{return}\ (\mathsf{bind}\ N \gg\!= \lambda x.\mathsf{reflect}_A\ (\mathsf{Var}\ x))
\end{aligned}
$$

$$
\mathsf{norm}_A\ M = \mathsf{reify}_A\ ([\![M]\!]\varnothing)
$$

[Filinski, TLCA 2001; Lindley, NBE 2009]

A summary of extensional NBE for sums

- Normalising with sums is non-trivial
- Call-by-value sums can be interpreted using continuations or a suitable free monad
  [Danvy, POPL 1996; Filinski, TLCA 2001; Lindley, NBE 2009]
- Call-by-name sums require more care
  [Altenkirch et al., LICS 2001; Balat et al., POPL 2004]

From reduction-based normalisation to NBE

NBE can be derived from reduction-based normalisation by a series of standard program transformations.

Example: naive β-reduction $\longrightarrow$ intensional NBE
Input: naive normalisation algorithm (top-down traversal contracting β-redexes by substitution)

1. add an environment in place of substitution
2. factor through weak normalisation (not reducing under lambda)
3. replace lambda abstractions with closures
4. replace closures with higher-order functions

Output: intensional NBE

[my PhD thesis; Danvy, AFP 2008]

Some references

Per Martin-Löf. An intuitionistic theory of types. OUP, 1972.

Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional. LICS 1991.

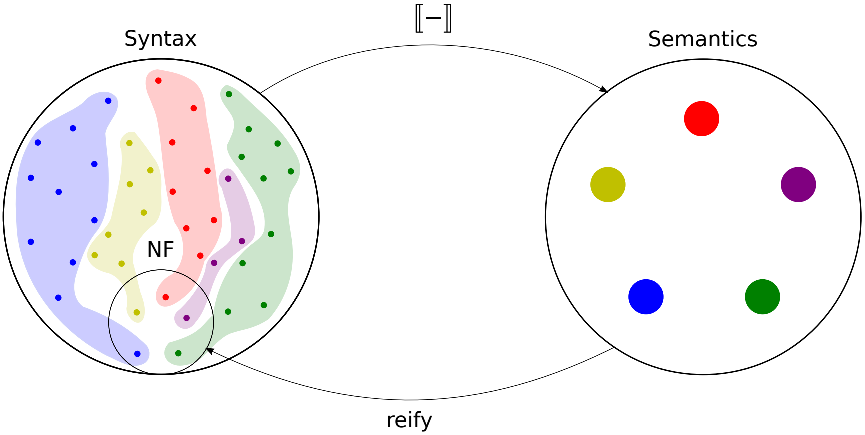
Olivier Danvy. Type-directed partial evaluation. POPL 1996.

Andrzej Filinski. Normalization by evaluation for the computational lambda calculus. TLCA 2001.

Sam Lindley. Normalisation by evaluation in the compilation of typed functional programming languages. PhD Thesis, 2005.

Sam Lindley. Accumulating bindings. NBE 2009.

NBE



$$\boxed{\mathsf{norm} = \mathsf{reify} \circ [\![-]\!]}$$