Normalization by Evaluation with Free Extensions

Nathan Corbyn[1], Ohad Kammar[2], Sam Lindley[2], Nachi Valliappan[3], Jeremy Yallop[4]

[1] Oxford University, [2] University of Edinburgh, [3] Chalmers University of Technology, [4] Cambridge University

TyDe, 11 Sep ’22, Ljubljana

RESEARCH-ARTICLE •

Practical normalization by evaluation for EDSLs

Authors: Nachiappan Valliappan,  Alejandro Russo, Sam Lindley Authors Info & Claims

Haskell 2021: Proceedings of the 14th ACM SIGPLAN International Symposium on Haskell • August 2021

Practical Normalization by Evaluation for EDSLs

Nachiappan Valliappan
Chalmers University of Technology
Gothenburg, Sweden

Alejandro Russo
Chalmers University of Technology
Gothenburg, Sweden

Sam Lindley
The University of Edinburgh
Edinburgh, United Kingdom

with a rich set of features such as sums, arrays, exceptions, and state—and in particular—a detailed and extensible account of their interaction.
- Practical extensions of standard NbE techniques to implement a richer set of domain-specific equations, and variations that control unnecessary code expansion.
- Examples showing that NbE provides a principled alternative to ad hoc techniques that combine deep and shallow embedding to implement fusion for functions, loops and arrays in an eDSL.

The complete Haskell source code and examples in this paper can be found in the accompanying material available at https://github.com/nachivpn/nbe-edsl.

2 Normalizing EDSL Programs

This section showcases our implementation with examples of normalizing eDSL programs using NbE. We begin with standard examples of normalizing the exponentiation function

```
power :: Exp (Int → Int → Int)
power = lam $ λn → lam $ λx → rec n (f x) 1
  where f x = lam $ λ_ → lam $ λacc → (x ∗ acc)
```

This implementation corresponds to the $power_7$ variant, and is implemented using expression combinators: $lam :: (Exp\ a \to Exp\ b) \to Exp\ (a \to b)$ is a lambda expression combinator and $rec :: Exp\ Int \to Exp\ (Int \to a \to a) \to Exp\ a \to Exp\ a$ is a primitive recursion combinator such that $rec\ n\ g\ a$ is equivalent to $g\ 1\ (g\ 2\ (...(g\ n\ a)))$. Although possible, note that the type of *rec* is not entirely wrapped under *Exp* as $Exp\ (Int \to (Int \to a \to a) \to a \to a)$. This choice prevents unnecessary clutter caused by explicit function application in the expression language, and trades some specialization power (i.e., the subsumption of some stage separations) for a more convenient interface. We make this choice for all primitive combinators that require multiple arguments.

An expression of a function type can be applied using the combinator $app :: Exp\ (a \to b) \to Exp\ a \to Exp\ b$ as *app* (*power* 3), where the argument is a numeral expression

Practical Normalization by Evaluation for EDSLs

Nachiappan Valliappan
Chalmers University of Technology
Gothenburg, Sweden

Alejandro Russo
Chalmers University of Technology
Gothenburg, Sweden

Sam Lindley
The University of Edinburgh
Edinburgh, United Kingdom

"Examples showing that NbE provides a principled alternative to ad hoc techniques that combine deep and shallow embedding to implement fusion for functions, loops and arrays in an eDSL"

Practical Normalization by Evaluation for EDSLs

Nachiappan Valliappan
Chalmers University of Technology
Gothenburg, Sweden

Alejandro Russo
Chalmers University of Technology
Gothenburg, Sweden

Sam Lindley
The University of Edinburgh
Edinburgh, United Kingdom

"Examples showing that NbE provides a **principled alternative** to ad hoc techniques that combine deep and shallow embedding to implement fusion for functions, loops and arrays in an eDSL"

Practical Normalization by Evaluation for EDSLs

Nachiappan Valliappan
Chalmers University of Technology
Gothenburg, Sweden

Alejandro Russo
Chalmers University of Technology
Gothenburg, Sweden

Sam Lindley
The University of Edinburgh
Edinburgh, United Kingdom

"**Examples** showing that NbE provides a **principled alternative** to ad hoc techniques that combine deep and shallow embedding to implement fusion for functions, loops and arrays in an eDSL"

Reviewer 2

Reviewer 2 strikes again!

Reviewer 2 strikes again!

“just a huge long list of examples”

Reviewer 2 strikes again!

“feels like there's an underlying technique trying to get out, but I'm not sure what it is. Without seeing the technique it's also hard to criticise the technique”

...did get published in Haskell ’21.

Types $a, b ::= a \to b \mid \mathrm{Nat}$

Terms $t, u ::= x \mid \lambda x.\, t \mid t\, u \mid \underline{k} \mid t * u$

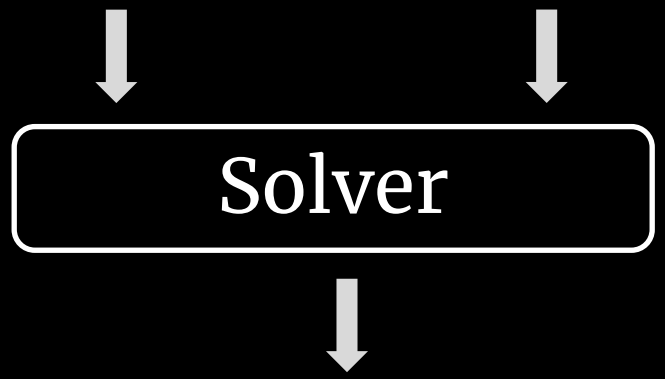
$$\lambda x.(\underline{2} * \underline{3}) * x \qquad \lambda x.(\lambda y.\underline{2} * y)(x * \underline{3})$$

$$\lambda x.\underline{6} * x$$

Types $a, b ::= a \to b \mid \mathrm{Nat}$

Terms $t, u ::= x \mid \lambda x.\, t \mid t\, u \mid \underline{k} \mid t * u$

$$\lambda x.(\underline{2} * \underline{3}) * x \qquad \lambda x.(\lambda y.\underline{2} * y)(x * \underline{3})$$



$$\lambda x.\underline{6} * x$$

Types $a, b ::= a \to b \mid \mathrm{Nat}$

Terms $t, u ::= x \mid \lambda x.\, t \mid t\, u \mid \underline{k} \mid t * u$

$$\mathcal{N} = (\mathbb{N}, *, 1)$$

$$\lambda x.(\underline{2} * \underline{3}) * x \qquad \lambda x.(\lambda y.\underline{2} * y)(x * \underline{3})$$

$$\downarrow \qquad \downarrow$$

$$\textsc{Frex}(\mathcal{N}, \mathrm{Var})$$

$$\downarrow$$

$$\lambda x.\underline{6} * x$$

Types $a, b ::= a \to b \mid \mathrm{Nat}$

Terms $t, u ::= x \mid \lambda x.\, t \mid t\, u \mid \underline{k} \mid t * u$

$$\mathcal{N} = (\mathbb{N}, *, 1)$$

$$\lambda x.(\underline{2} * \underline{3}) * x \qquad \lambda x.(\lambda y.\underline{2} * y)(x * \underline{3})$$

$$\textsc{Frex}(\mathcal{N}, ???)$$

$$\lambda x.\underline{6} * x$$

Types $a, b ::= a \to b \mid \mathrm{Nat}$

Terms $t, u ::= x \mid \lambda x.\, t \mid t\, u \mid \underline{k} \mid t * u$

$$\mathcal{N} = (\mathbb{N}, *, 1)$$

$$\lambda x.(\underline{2} * \underline{3}) * x \qquad \lambda x.(\lambda y.\underline{2} * y)(x * \underline{3})$$

$$\downarrow \qquad \downarrow$$

$$\textsc{Frex}(\mathcal{N}, \mathrm{Ne\ Nat})$$

Key idea!

$$\downarrow$$

$$\lambda x.\underline{6} * x$$

Neutrals and normal forms

$$\text{Neutrals } n ::= x \mid n\, m$$

$$\text{Normal forms } m ::= \lambda x.m \mid \underline{k} * n_1 * \ldots * n_j \quad (n_i \leq n_{i+1})$$

Requires some ingenuity

$\lambda x.(\underline{2} * \underline{3}) * x$ isn't normal, but $\lambda x.\underline{6} * x$ is.

Types $a, b ::= \ldots \mid \mathrm{M}$

Terms $t, u ::= \ldots \mid \underline{k} \mid t \otimes u$

$$\mathcal{M} = (M, \otimes, \mathbb{1})$$

$$\lambda x.(\underline{a} \otimes \underline{b}) \otimes x \qquad \lambda x.(\lambda y.\underline{a} \otimes y)(x \otimes \underline{b})$$

$$\downarrow \qquad \downarrow$$

$$\textsc{Frex}(\mathcal{M}, \mathrm{Ne\ M})$$

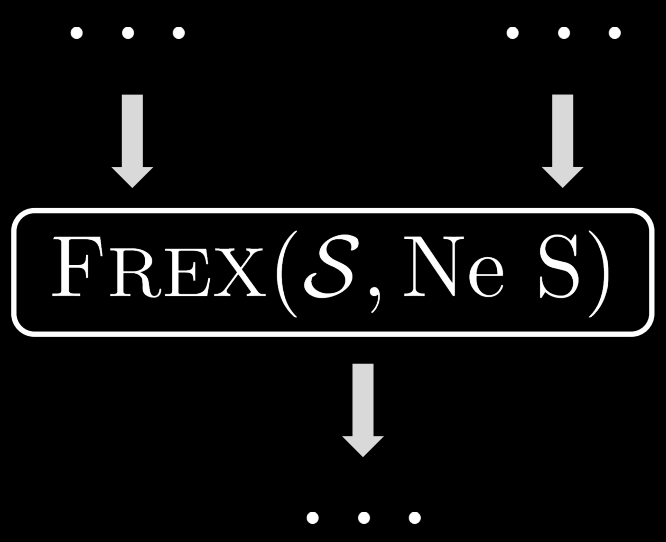
$$\downarrow$$

$$\lambda x.(\underline{a \otimes b}) \otimes x$$

Types $a, b ::= \ldots \mid \mathrm{S}$

Terms $t, u ::= \ldots \mid \underline{k} \mid t \otimes u \mid t \oplus u$

$$\mathcal{S} = (S, \otimes, \mathbb{1}, \oplus, \mathbb{0})$$



Normalization by Evaluation (NbE)

$$\mathrm{eval} : \mathrm{Tm}\ a \to [\![a]\!]$$

$$\mathrm{quote} : [\![a]\!] \to \mathrm{Nf}\ a$$

$$\mathrm{norm} : \mathrm{Tm}\ a \to \mathrm{Nf}\ a$$

$$\mathrm{norm} = \mathrm{quote} \circ \mathrm{eval}$$

NbE interpretation of types

$$
\begin{aligned}
[\![\mathrm{Nat}]\!] &= \textsc{Frex}(\mathcal{N}, \mathrm{Ne\ Nat}) \\
[\![a \to b]\!] &= [\![a]\!] \to [\![b]\!]
\end{aligned}
$$

As usual!

Frex interface

$\textsc{Frex}(\mathcal{A}, X)$ is a Σ-algebra

Requires some ingenuity

$i_{\mathcal{A}} : \mathcal{A} \to \textsc{Frex}(\mathcal{A}, X)$

$i_X : X \to |\textsc{Frex}(\mathcal{A}, X)|$

For $h : \mathcal{A} \to \mathcal{B}$, $e : X \to |\mathcal{B}|$

Pick as normal forms

$$\exists!\ match_{h,e} : \textsc{Frex}(\mathcal{A}, X) \to \mathcal{B} \text{ s.t.}$$

$$match_{h,e} \circ i_{\mathcal{A}} = h \quad |match_{h,e}| \circ i_X = e$$

NbE proofs using Frex

$$t \approx \mathrm{norm}\ t,\ \text{for}\ t : \mathrm{Tm}\ \mathrm{Nat}$$

$$\text{for FREE!}^*$$

There’s more and lots to be done

NbE with Free Extensions:

- Information-flow control extension
- WIP: Formal account of the recipe
- TODO: Extend to multi-sorted algebras

NbE *as* a Free Extension:

- Ocaml implementation
- Free Extensions of Categories, CCCs, STLC

Check out Nathan’s poster at SRC

Calling NbE fans

Normalization for Fitch-style Modal Calculi

DISTINGUISHED PAPER

Who *Nachiappan Valliappan*, *Fabian Ruch*, *Carlos Tomé Cortiñas*

Track ICFP 2022 ICFP Papers and Events

This program is tentative and subject to change.

When **Tue 13 Sep 2022 11:10 - 11:30 at Linhart** - Logic Chair(s): Ilya Sergey

In a Nutshell

Normalization by Evaluation for primitive types by leveraging their algebraic structure.

$$[\![\mathrm{A}]\!] = \textsc{Frex}(\mathcal{A}, \mathrm{Ne}\ \mathrm{A})$$

Extended abstract: nachivpn.me/nwf.pdf

EOM