Normalization by Evaluation for Call-by-Push-Value

Andreas Abel* and Christian Sattler

Department of Computer Science and Engineering, Gothenburg University

Normalization by evaluation (NbE) [Berger and Schwichtenberg, 1991] is the interpretation of an (open) term of type A as value in a suitable model $[\![A]\!]$, followed by *reification* of the value to a normal form of type A. Functions f in $[\![A \Rightarrow B]\!]$ are reified as λ-abstractions whose bodies are obtained by *reflecting* a fresh variable of type A as value a in $[\![A]\!]$ and reifying the application $f\,a$ at type B. A suitable model that supports fresh variable generation are *presheaves* over the category of typing contexts Γ and order-preserving embeddings $\Gamma \subseteq \Gamma'$, where a base type o is interpreted as the presheaf $\mathsf{Ne}\ o$ of neutral normal forms of type o, and function types by the presheaf exponential aka Kripke function space [Coquand, 1993, Altenkirch et al., 1995].

NbE for sum types requires a refinement of the model, since reflection of a variable of type $A + B$ as a value in $[\![A + B]\!]$ requires case distinction in the model. One such refinement are *sheaves* [Altenkirch et al., 2001]; another is the use of a monad $\mathcal{C}$ [Filinski, 2001, Barral, 2008] in the category of presheaves for the interpretation of sum types: $[\![A + B]\!] = \mathcal{C}([\![A]\!] + [\![B]\!])$. The smallest such *"cover" monad* $\mathcal{C}$ are binary trees where leaves are the monadic unit aka return, and the nodes case distinctions over neutrals $\mathsf{Ne}\,(A_1 + A_2)$ of sum type. When leaves are normal forms, the whole tree represents a normal form, thus, $\mathsf{runNf} : \mathcal{C}(\mathsf{Nf}\,A) \to \mathsf{Nf}\,A$ is trivial. This *running of the monad* on normal forms represents the algorithmic part of the sheaf condition on $\mathsf{Nf}\,A$ and extends as $\mathsf{run} : \mathcal{C}[\![A]\!] \to [\![A]\!]$ to all semantic types.

The given interpretation of sum types $[\![A + B]\!] = \mathcal{C}([\![A]\!] + [\![B]\!])$ corresponds to the call-by-name (CBN) lambda calculus with lazy constructors. NbE can also be performed in call-by-value (CBV) style, then the monad is placed in the codomain of function types: $[\![A \Rightarrow B]\!] = [\![A]\!] \Rightarrow \mathcal{C}[\![B]\!]$ [Danvy, 1996]. A systematic semantic analysis of CBN and CBV lambda-calculi has been pioneered by Moggi [1991] through translation into his computational lambda calculus; Filinski [2001] studied NbE for the latter calculus using the continuation monad. Moggi's work was continued and refined by Levy [2006] who subsumed CBV and CBN under his monadic call-by-push-value (CBPV) calculus. In this work, we study NbE for CBPV.

CBPV was designed to study lambda-calculus with effects. It separates types into *value* types P and *computation* types N, which we, in analogy to polarized lambda-calculus [Zeilberger, 2009] refer to as *positive* and *negative* types. Variables stand for values, thus, have positive types. The monad that models the effects is placed at the transition from values to computations $\mathit{Comp}\,P$, and computations can be embedded into values by *thunking* ($\mathit{Thunk}\,N$).

$$\begin{array}{llllll} \mathsf{Ty}^+ & \ni & P & ::= & o \mid P_1 + P_2 \mid \mathit{Thunk}\,N & \text{positive type / value type} \\ \mathsf{Ty}^- & \ni & N & ::= & P \Rightarrow N \mid \mathit{Comp}\,P & \text{negative type / computation type} \end{array}$$

We restrict to a fragment of *pure* CBPV with a single positive connective, sum types $P_1 + P_2$, and a single negative connective, call-by-value function types $P \Rightarrow N$. While we have no proper effects, the evaluation of open terms requires the effect of case distinction over neutrals, modeled by a cover monad $\mathcal{C}$. In the following, we give inductive definitions of the presheaves of normal (Nf) and neutral normal forms (Ne) of our fragment of CBPV and a concrete, strong cover monad Cov.

$$\mathsf{var}\ \frac{\mathsf{Var}\ o\ \Gamma}{\mathsf{Nf}\ o\ \Gamma} \quad \mathsf{thunk}\ \frac{\mathsf{Nf}\ N\ \Gamma}{\mathsf{Nf}\ (\mathit{Thunk}\ N)\ \Gamma} \quad \mathsf{inj}_i\ \frac{\mathsf{Nf}\ P_i\ \Gamma}{\mathsf{Nf}\ (P_1 + P_2)\ \Gamma} \quad \mathsf{ret}\ \frac{\mathsf{Cov}\ (\mathsf{Nf}\ P)\ \Gamma}{\mathsf{Nf}\ (\mathit{Comp}\ P)\ \Gamma} \quad \mathsf{abs}\ \frac{\mathsf{Nf}\ N\ (\Gamma.P)}{\mathsf{Nf}\ (P \Rightarrow N)\ \Gamma}$$

*Supported by VR grant 621-2014-4864 and EU Cost Action CA15123.

$$\text{force}\ \frac{\mathsf{Var}\,(\mathit{Thunk}\ N)\,\Gamma}{\mathsf{Ne}\ N\ \Gamma} \qquad \text{app}\ \frac{\mathsf{Ne}\,(P \Rightarrow N)\,\Gamma \qquad \mathsf{Nf}\ P\ \Gamma}{\mathsf{Ne}\ N\ \Gamma} \qquad \text{bind}\ \frac{\mathsf{Ne}\,(\mathit{Comp}\ P)\,\Gamma \qquad \mathsf{Cov}\ \mathcal{J}\,(\Gamma.P)}{\mathsf{Cov}\ \mathcal{J}\ \Gamma}$$

$$\text{return}\ \frac{\mathcal{J}\ \Gamma}{\mathsf{Cov}\ \mathcal{J}\ \Gamma} \qquad \text{case}\ \frac{\mathsf{Var}\,(P_1 + P_2)\,\Gamma \qquad \mathsf{Cov}\ \mathcal{J}\,(\Gamma.P_1) \qquad \mathsf{Cov}\ \mathcal{J}\,(\Gamma.P_2)}{\mathsf{Cov}\ \mathcal{J}\ \Gamma}$$

($\mathcal{J}$ stands for an arbitrary presheaf in $\mathsf{Cov}\ \mathcal{J}$.) Normal forms start from a variable of base type and continue with introductions, except that the services of the monad can be used at the transition ret from positive to negative types ($\mathit{Comp}\ P$). Neutrals are eliminations of variables of type $\mathit{Thunk}\ N$ into a positive type $\mathit{Comp}\ P$, and can then be bound to a variable of type P to be used in a computation (see bind). Variables of sum type $P_1 + P_2$ can be utilized in computations through a case split.

Terms Tm of CBPV are obtained by blurring the distinction between Ne and Nf, generalizing bind and case from $\mathsf{Cov}\ \mathcal{J}$ to computations $\mathsf{Tm}\ N$, and relaxing var to variables of arbitrary type P and force to arbitrary terms of type $\mathit{Thunk}\ N$. Terms are evaluated in the following presheaf model, which interprets *Thunk* as the identity and *Comp* as Cov.

$$\begin{array}{lcllcl} [\![P_1 + P_2]\!] & = & [\![P_1]\!] \hat{+} [\![P_2]\!] & [\![P \Rightarrow N]\!] & = & [\![P]\!] \hat{\Rightarrow} [\![N]\!] \\ [\![\mathit{Thunk}\ N]\!] & = & [\![N]\!] & [\![\mathit{Comp}\ P]\!] & = & \mathsf{Cov}[\![P]\!] \\ [\![o]\!] & = & \mathsf{Var}\ o & & & \end{array}$$

The evaluation of bind terms in $\mathsf{Tm}\ N$ relies on $\mathsf{run} : \mathsf{Cov}[\![N]\!] \to [\![N]\!]$, which makes any computation type monadic. Reflection $\uparrow$ and reification $\downarrow$ are defined mutually by induction on the type. They take the usual form, only that reflection of positive variables is monadic, to allow the complete splitting of sums via case. It is invoked by reification of functions $\downarrow^{P \Rightarrow N}$ via runNf.

$$\begin{array}{lcllcl} \uparrow^P & : & \mathsf{Var}\ P \to \mathsf{Cov}\ [\![P]\!] & \downarrow^P & : & [\![P]\!] \to \mathsf{Nf}\ P \\ \uparrow^N & : & \mathsf{Ne}\ N \to [\![N]\!] & \downarrow^N & : & [\![N]\!] \to \mathsf{Nf}\ N \end{array}$$

The details of our construction, plus extension to product types and polarized lambda calculus, can be found in the full version at https://arxiv.org/abs/1902.06097. A partial Agda formalization is available at https://github.com/andreasabel/ipl.

References

T. Altenkirch, M. Hofmann, and T. Streicher. Categorical reconstruction of a reduction free normalization proof. In *CTCS'95*, vol. 953 of *LNCS*. Springer, 1995. https://doi.org/10.1007/3-540-60164-3_27.

T. Altenkirch, P. Dybjer, M. Hofmann, and P. J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *LICS'01*. IEEE CS Press, 2001. https://doi.org/10.1109/LICS.2001.932506.

F. Barral. *Decidability for non-standard conversions in lambda-calculus*. PhD thesis, Ludwig-Maximilians-University Munich, 2008.

U. Berger and H. Schwichtenberg. An inverse to the evaluation functional for typed λ-calculus. In *LICS'91*. IEEE CS Press, 1991. https://doi.org/10.1109/LICS.1991.151645.

C. Coquand. From semantics to rules: A machine assisted analysis. In *CSL'93*, vol. 832 of *LNCS*. Springer, 1993. https://doi.org/10.1007/BFb0049326.

O. Danvy. Type-directed partial evaluation. In *POPL'96*. ACM, 1996. https://doi.org/10.1145/237721.237784.

A. Filinski. Normalization by evaluation for the computational lambda-calculus. In *TLCA'01*, vol. 2044 of *LNCS*. Springer, 2001. https://doi.org/10.1007/3-540-45413-6_15.

P. B. Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *HOSC*, 19(4), 2006. https://doi.org/10.1007/s10990-006-0480-6.

E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1), 1991. https://doi.org/10.1016/0890-5401(91)90052-4.

N. Zeilberger. *The Logical Basis of Evaluation Order and Pattern-Matching*. PhD thesis, Carnegie Mellon University, 2009. http://software.imdea.org/~noam.zeilberger/thesis.pdf.