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 $\lambda$-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 $\Gamma$ 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 $C$ [Filinski, 2001, Barral, 2008] in the category of presheaves for the interpretation of sum types: $[A + B] = C([A] + [B])$. The smallest such “cover” monad $C$ 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, $\mathsf{runNf} : C(\mathsf{Nf} A) \rightarrow \mathsf{Nf} A$ 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 $\mathsf{run} : C[A] \rightarrow [A]$ to all semantic types.

The given interpretation of sum types $[A + B] = C([A] + [B])$ 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: $[A \Rightarrow B] = [A] \Rightarrow C[B]$ [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 $\mathsf{Comp} P$, and computations can be embedded into values by thunking ($\mathsf{Thunk} N$).

$\mathsf{Ty}^+ \ni \ P :: o \mid P_1 + P_2 \mid \mathsf{Thunk} N \quad \text{positive type / value type}$

$\mathsf{Ty}^- \ni \ N :: P \Rightarrow N \mid \mathsf{Comp} P \quad \text{negative type / computation type}$

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 $\mathsf{Cov}$. In the following, we give inductive definitions of the presheaves of normal ($\mathsf{Nf}$) and neutral normal forms ($\mathsf{Ne}$) of our fragment of CBPV and a concrete, strong cover monad $\mathsf{Cov}$.

var $\var o \Gamma$ thunk $\mathsf{Nf} N \Gamma$ inj $\mathsf{Nf} P, \Gamma$ ret $\mathsf{Cov} (\mathsf{Nf} P) \Gamma$ abs $\mathsf{Nf} (N \Gamma P)$

|$\mathsf{Var} o \Gamma$|$\mathsf{Thunk} N \Gamma$|$\mathsf{Nf} (\mathsf{Thunk} N) \Gamma$|$\mathsf{Nf} (P_1 + P_2) \Gamma$|$\mathsf{Nf} (\mathsf{Comp} P) \Gamma$|$\mathsf{Nf} (P \Rightarrow N) \Gamma$

*Supported by VR grant 621-2014-4864 and EU Cost Action CA15123.
Normalization by Evaluation for Call-by-Push-Value

A. Abel and C. Sattler

\[ \begin{array}{ccl}
\text{force} & \text{Var} \left( \text{Thunk} \; N \right) \; \Gamma & \text{app} & \text{Ne} \left( P \Rightarrow N \right) \; \Gamma & \text{Ne} \; P \; \Gamma & \text{bind} & \text{Ne} \left( \text{Comp} \; P \right) \; \Gamma & \text{Cov} \; J \; (\Gamma \cdot P) \\
\quad & \text{Ne} \; N \; \Gamma & & \text{Ne} \; N \; \Gamma & & \text{Ne} \; N \; \Gamma & & \text{Cov} \; J \; \Gamma \\
\text{return} & \text{Cov} \; J \; \Gamma & \text{case} & \text{Var} \left( P_1 + P_2 \right) \; \Gamma & \text{Cov} \; J \; (\Gamma \cdot P_1) & \text{Cov} \; J \; (\Gamma \cdot P_2) \\
\end{array} \]

(\( J \) stands for an arbitrary presheaf in \( \text{Cov} \; J \).
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 \( \text{ret} \) from positive to negative types (\( \text{Comp} \; P \)).
Neutrals are eliminations of variables of type \( \text{Thunk} \; N \) into a positive type \( \text{Comp} \; P \), and can then be bound to a variable of type \( P \) to be used in a computation (see \( \text{bind} \)).
Variables of sum type \( P_1 + P_2 \) can be utilized in computations through a \( \text{case} \) split.

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

\[
\begin{align*}
[P_1 + P_2] & = [P_1] \uparrow [P_2] \\
[\text{Thunk} \; N] & = [N] \\
[o] & = \text{Var} \; o
\end{align*}
\]

The evaluation of \( \text{bind} \) terms in \( \text{Tm} \; N \) relies on \( \text{run} : \text{Cov}[N] \rightarrow [N] \), 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 \( \text{case} \). It is invoked by reification of functions \( \downarrow P \Rightarrow N \) via \( \text{runNf} \).

\[
\begin{align*}
\uparrow^P & : \text{Var} \; P \rightarrow \text{Cov} \; [P] \\
\downarrow^P & : [P] \rightarrow \text{Nf} \; P \\
\uparrow^N & : \text{Ne} \; N \rightarrow [N] \\
\downarrow^N & : [N] \rightarrow \text{Nf} \; N
\end{align*}
\]

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