From Self-Interpreters to Normalization by Evaluation

Mathieu Boespflug

FROM SELF-INTERPRETERS TO NORMALIZATION BY EVALUATION

MATHIEU BOESPFLUG

Abstract. We characterize normalization by evaluation as the composition of a self-interpreter with a self-reducer using a special representation scheme, in the sense of Mogensen (1992). We do so by deriving in a systematic way an untyped normalization by evaluation algorithm from a standard interpreter for the λ-calculus. The derived algorithm is not novel and indeed other published algorithms may be obtained in the same manner through appropriate adaptations to the representation scheme.

1. Self-interpreters and self-reducers

What is a self-interpreter? For the untyped λ-calculus, Mogensen (1992) offers the following definition, given an injective mapping $\ulcorner\cdot\urcorner$ (the *representation scheme*) that yields *representations* of arbitrary terms:

$$E\ \ulcorner M\urcorner =_\beta M.$$

That is, a self-interpreter is a term E of the λ-calculus such that when applied to the representation $\ulcorner M\urcorner$ of any term M, the result is a convertible term (modulo renaming).

The $\ulcorner\cdot\urcorner$ mapping[1] cannot of course be defined within the λ-calculus itself, but we posit its existence as a primitive operation of the calculus. The representation of a term is a piece of data, something that can be manipulated, transformed and inspected within the calculus itself. It is natural to represent data as terms in normal form, so that data may be regarded as constant with regard to term reduction. Consider the following grammar for terms and normal terms:

$$\begin{array}{rlll} & \mathsf{Var} & \ni\ x, y, z & \\ & \mathsf{Term} & \ni\ t & ::= x \mid \lambda x.t \mid t\ t \\ \mathsf{Term} \supset & \mathsf{Term}_{\mathsf{NF}} & \ni\ t_n & ::= t_a \mid \lambda x.t_n \\ \mathsf{Term} \supset & \mathsf{Term}_{\mathsf{A}} & \ni\ t_a & ::= x \mid t_a\ t_n \end{array}$$

The representation scheme can be typed as $\ulcorner\cdot\urcorner : \mathsf{Term} \to \mathsf{Term}_{\mathsf{NF}}$.

All manner of representation schema are possible, but Mogensen commits to a particularly simple representation scheme, one that enables him to implement a trivially simple self-interpreter that not only yields convertible terms from their representations, but in fact whose weak head normal form when applied to a normal term M is identical to M, up to renaming of variables. Let us call this particular self-interpreter E_α. We have that

$$E_\alpha\ \ulcorner M\urcorner \longrightarrow_{\text{whnf}} M.$$

Mogensen goes on to define a self-reducer as a transformation on representations:

$$R\ \ulcorner M\urcorner =_\beta \ulcorner NF_M\urcorner,$$

where NF_M stands for the normal form of M, if one exists. Equipped with such a contraption, we can define a special kind of self-interpreter with the additional property that all representations of terms evaluate to normal forms. For all M,

$$E_{NF}\ \ulcorner M\urcorner \hat{=} E_\alpha(R\ \ulcorner M\urcorner) \longrightarrow_{\text{whnf}} NF_M.$$

After a small detour towards concrete implementations of the above, we will show how through successive transformations we may obtain an untyped normalization by evaluation

[1] This is an analogue of the `quote` special form of Scheme.

algorithm, in fact precisely the algorithm presented in (Boespflug, 2009). This algorithm is but one of several variants of untyped normalization by evaluation algorithms, and indeed the transformations presented here can readily be adapted to derive the algorithms of Aehlig et al. (2008) and Filinski and Rohde (2004). Because the starting points of these transformations are in fact standard definitions of evaluators and normalizers, it is possible to obtain the correctness of the normalization by evaluation algorithm as a corollary of the correctness of the meaning preserving transformations we use.

2. Implementation

In the following, we assume a λ-calculus with matching constructs, for convenience. Note however, that we could always reformulate the following in the pure λ-calculus via a Church encoding.

Let us recall the representation scheme given in (Mogensen, 1992):

$$\begin{aligned} \ulcorner x \urcorner &= \mathsf{Var}\ \ x \\ \ulcorner \lambda x.t \urcorner &= \mathsf{Lam}\ (\lambda x.\ulcorner t \urcorner) \\ \ulcorner t_0\ t_1 \urcorner &= \mathsf{App}\ \ulcorner t_0 \urcorner\ \ulcorner t_1 \urcorner \end{aligned}$$

for distinguished constructors Var , Lam and App . Notice how the representation scheme uses higher order abstract syntax (HOAS). With this representation scheme, the definition of the E_α self-interpreter above can be given as

$$\begin{aligned} E_\alpha\ (\mathsf{Var}\ x) &= x \\ E_\alpha\ (\mathsf{Lam}\ t) &= t \\ E_\alpha\ (\mathsf{App}\ t_0\ t_1) &= (E_\alpha\ t_0)\ (E_\alpha\ t_1) \end{aligned}$$

Given a datatype for terms, and assuming the metalanguage follows a call-by-value evaluation strategy, an evaluator yielding weak head normal forms using a call-by-name strategy might typically be defined as follows:

$$\begin{aligned} \mathit{eval}\ (\mathsf{Var}\ x) &= \mathsf{Var}\ x \\ \mathit{eval}\ (\mathsf{Lam}\ t) &= \mathsf{Lam}\ t \\ \mathit{eval}\ (\mathsf{App}\ t_0\ t_1) &= \mathsf{case}\ \mathit{eval}\ t_0\ \mathsf{of} \\ &\qquad \mathsf{Lam}\ t \to \mathit{eval}\ (t\ t_1) \\ &\qquad\quad t_0' \to \mathsf{App}\ t_0'\ (\mathit{eval}\ t_1) \end{aligned}$$

Now a single change to the above gives us the definition of a normalizer:

$$\begin{aligned} \mathit{norm}\ (\mathsf{Var}\ x) &= \mathsf{Var}\ x \\ \mathit{norm}\ (\mathsf{Lam}\ t) &= \mathsf{Lam}\ (\lambda x.\mathit{norm}\ (t\ x)) \\ \mathit{norm}\ (\mathsf{App}\ t_0\ t_1) &= \mathsf{case}\ \mathit{norm}\ t_0\ \mathsf{of} \\ &\qquad \mathsf{Lam}\ t \to t\ t_1 \\ &\qquad\quad t_0' \to \mathsf{App}\ t_0'\ (\mathit{norm}\ t_1) \end{aligned}$$

Whereas the evaluator does not traverse the boundary of a binder to fetch redexes beneath it (an abstraction is a value), a normalizer on the other hand does, so we move the recursive call to *norm* inside the abstraction. There are no more recursive calls in the definition of *norm* than there are in the definition of *eval* — one of them merely changed place.

A slight modification gives a normalizer following the call-by-value strategy. Namely, we normalize the argument before applying any abstraction to it.

$$\begin{aligned} \mathit{norm}\ (\mathsf{Var}\ x) &= \mathsf{Var}\ x \\ \mathit{norm}\ (\mathsf{Lam}\ t) &= \mathsf{Lam}\ (\lambda x.\mathit{norm}\ (t\ x)) \\ \mathit{norm}\ (\mathsf{App}\ t_0\ t_1) &= \mathsf{case}\ \mathit{norm}\ t_0\ \mathsf{of} \\ &\quad \mathsf{Lam}\ t \to t\ (\mathit{norm}\ t_1) \\ &\quad\ \ t_0' \to \mathsf{App}\ t_0'\ (\mathit{norm}\ t_1) \end{aligned}$$

Notice that the call-by-value and call-by-name normalizers just defined both fit the bill as self-reducers: they take representations of terms to representations in normal form. That is, if one is willing to forgo the incompleteness of the call-by-value normalizer: there are terms for which there exists a normal form that may never be reached using a call-by-value strategy.

The need to define two separate normalizers for two different normalization strategies is rather unfortunate. But if we restrict the input terms to a certain shape, namely those terms in continuation passing style (CPS), then there is only one normalization strategy possible, because at any given step in the computation, the only possible redex outside of any abstraction is to be found at the head of the term, if any. Hence only one normalizer need be built. But more importantly, it will justify a change in the representation scheme allowing for a much more efficient self-reducer than is possible with the current representation scheme.

3. Shifting representation scheme

A CPS transformation takes terms in Term to terms in $\mathsf{Term}_{\mathsf{CPS}}$, a language generated by the following grammar.

$$\begin{array}{lllll} \mathsf{Term} \supset \mathsf{Term}_{\mathsf{V}} & \ni & v & ::= & x \mid \lambda x.t_c \\ \mathsf{Term} \supset \mathsf{Term}_{\mathsf{CPS}} & \ni & t_c & ::= & v \mid v\ v \end{array}$$

By modifying the representation scheme to return representations in continuation passing style instead, we effectively encode the reduction strategy of the normalizer into the scheme itself[2]. But on the flip side, we get a simpler definition for the normalizer:

$$\begin{aligned} \mathit{norm}\ (\mathsf{Var}\ x) &= \mathsf{Var}\ x \\ \mathit{norm}\ (\mathsf{Lam}\ t) &= \mathsf{Lam}\ (\lambda x.\mathit{norm}\ (t\ x)) \\ \mathit{norm}\ (\mathsf{App}\ (\mathsf{Lam}\ t_0)\ t_1) &= \mathit{norm}\ (t_0\ t_1) \\ \mathit{norm}\ (\mathsf{App}\ (\mathsf{Var}\ x)\ t_1) &= \mathsf{App}\ (\mathsf{Var}\ x)\ (\mathit{norm}\ t_1) \end{aligned}$$

This definition is a mere specialization of the above definitions to terms in CPS, noting that such terms do not contain nested applications. We can break out the interpretation of application nodes to an auxiliary *app* function, using the fact that $\mathsf{App}\ (\mathsf{Var}\ x)\ (\mathit{norm}\ t_1) \equiv \mathit{norm}\ (\mathsf{App}\ (\mathsf{Var}\ x)\ t_1)$ for all x, t_1:

$$\begin{aligned} \mathit{norm}\ (\mathsf{Var}\ x) &= \mathsf{Var}\ x \\ \mathit{norm}\ (\mathsf{Lam}\ t) &= \mathsf{Lam}\ (\lambda x.\mathit{norm}\ (t\ x)) \\ \mathit{norm}\ (\mathsf{App}\ t_0\ t_1) &= \mathit{norm}\ (\mathit{app}\ t_0\ t_1) \\ \\ \mathit{app}\ (\mathsf{Lam}\ t_0)\ t_1 &= t_0\ t_1 \\ \mathit{app}\ (\mathsf{Var}\ x)\ t_1 &= \mathsf{App}\ (\mathsf{Var}\ x)\ t_1 \end{aligned}$$

[2]Alternatively, we could leave the representation scheme untouched and have the CPS transform as a function from representations to representations, which we intercalate between quoting and normalizing.

4. From here to there: normalization by evaluation

Normalization by evaluation corresponds to interpreting terms as their representation yet identifying entire classes of terms with unique representations. In our setting, this is achieved by inlining the third clause of *norm* in place of every occurrence of an App node in the representation of the term to normalize. As an optimization, we can inline only the call to *app*, not the recursive call to *norm*, yielding the following final definition for *norm*,

$$\begin{aligned} norm\ (\mathsf{Var}\ x) &= \mathsf{Var}\ x \\ norm\ (\mathsf{Lam}\ t) &= \mathsf{Lam}\ (\lambda x.norm\ (t\ x)) \\ norm\ (\mathsf{App}\ t_0\ t_1) &= \mathsf{App}\ t_0\ (norm\ t_1) \end{aligned}$$

where we use the fact that after inlining of *app* all App nodes that remain are applicative forms, i.e. bearing a variable as left branch. The associated interpretation function reads as follows:

$$\begin{aligned} [\![x]\!] &= \mathsf{Var}\ x \\ [\![\lambda x.t]\!] &= \mathsf{Lam}\ (\lambda x.[\![t]\!]) \\ [\![t_0\ t_1]\!] &= app\ [\![t_0]\!]\ [\![t_1]\!] \end{aligned}$$

This interpretation function can be read as a particular quote operation where terms are identified modulo weak head normal forms.

Normalization by evaluation is

$$nbe\ t = E_\alpha\ (norm\ [\![t]\!])$$

5. Conclusion

We have gone from a self-reducer to an equivalent self-reducer on representations in CPS. By inlining part of the obtained self-reducer into the representation of terms, we further obtained a preexisting normalization by evaluation algorithm (which also works for terms in direct style). In short, for terms in CPS, untyped normalization by evaluation is the composition of a self-reducer with an appropriate quote operation.

We have derived a more efficient self-reducer than the one presented in (Mogensen, 1992). This derivation also serves as a road map for an alternative and simple proof of correctness of untyped normalization by evaluation for terms in CPS.

Acknowledgements

The author wishes to dedicate this short note to Olivier Danvy. Once upon a studious afternoon, its essence fell out rather immediately from the following question when pondering an unrelated problem: “How would Olivier Danvy think?”.

References

K. Aehlig, F. Haftmann, and T. Nipkow. A Compiled Implementation of Normalization by Evaluation. *Theorem Proving in Higher Order Logics (TPHOLs 2008), Lecture Notes in Computer Science. Springer-Verlag*, 2008.

Mathieu Boespflug. Efficient normalization by evaluation. In *Informal proceedings of the 2009 Workshop on Normalization by Evaluation*, pages 29–34, August 2009.

A. Filinski and H.K. Rohde. A denotational account of untyped normalization by evaluation. *Lecture notes in computer science*, pages 167–181, 2004.

T.A. Mogensen. Efficient self-interpretation in lambda calculus. *Journal of Functional Programming*, 2(03):345–364, 1992.