A(b)normal Environments
Beniamino Accattoli\(^1\), Andrea Condoluci\(^3\), Giulio Guerrieri\(^2\), and Claudio Sacerdoti Coen\(^3\)

\(^1\) INRIA & École Polytechnique
\(^2\) Department of Computer Science - University of Bath
\(^3\) Department of Computer Science and Engineering - University of Bologna

The λ-calculus is a minimalistic computational model, having only three constructs, variables, abstractions, and applications, and one evaluation rule, β-reduction. It is natural to wonder whether it is indeed minimal—is there a sub-calculus that is able to represent the whole of the λ-calculus?

**Continuation-Passing Style.** It is well-known that the answer is yes: there is a family of continuation-passing style (CPS) transformations that identifies such an expressive sub-calculus. The key structural concept here is the notion of value, given by variables and abstractions, which is used to restrict the shape of applications. CPS translations may look convoluted and unnatural (in typed settings, in particular, they modify the type of the term). They have instead a number of nice properties and natural readings—notably, the change of types is the Curry-Howard mirror of double negation translations in proof theory. Reynolds historical survey is a nice introduction [9]—we refrain from even attempting an essential literature about continuations. In particular, they were extensively used in compilers for functional languages.

**Administrative Normal Form.** One of the most interesting studies about CPS is properly named The Essence of Compiling with Continuations, by Flanagan, Sabry, Duba, and Felleisen, appeared in 1993 [8] and based on Reasoning about Programs in Continuation-Passing Style by Sabry and Felleisen [10]. This line of research revealed that, for what concerns compilers and abstract machines, CPS translations can be replaced by a simpler transformation to administrative normal form (or A-normal form, or ANF), providing the advantages of the CPS without the convoluted transformation, in particular without changing the type. Morally, there is an ANF sub-sub-calculus at the heart of the CPS sub-calculus, or simply an alternative, simpler sub-calculus of the λ-calculus. The discovery had an immediate practical impact, but very limited theoretical impact—after 25 years, the literature on ANF is almost inexistent, while that on CPS still keeps growing.

Our work is devoted to the fine understanding of the role played by the ANF transformation with respect to recent developments in the theory of abstract machines and of call-by-value (CbV for short) evaluation. In particular, we are interested in analysing the impact of the ANF on the design and efficiency of abstract machines, and on its scalability to open terms.

**The New Wave of Abstract Machines.** The authors, and some of their coauthors (Barrenbaum, Barras, Mazza), have recently started a new wave in the theory of abstract machines [2, 3, 6, 1, 5, 4], where machines are analysed quantitatively, building on a new understanding of machine transitions rooted in linear logic. The new methodology amounts to bound the asymptotic overhead of machines as a function of the size of the initial term and the number of β-steps, essentially studying the efficiency of abstract machines from a complexity point of view. For as strange as it may sound, this is an aspect largely neglected by the literature—before 2014 we are aware of only two independent works on this topic, Blelloch and Greiner’s
A(b)normal Environments

Abnormal Environments. Environments store the delayed substitutions of the previously (partially) reduced beta-redexes. Such a delay is a form of sharing: the non-substituted argument is stored and thus shared among the non-substituted occurrences of the abstracted variable. And sharing is essential in order to avoid size explosion, the degeneracy typical of λ-calculi for which the size of terms can grow exponentially with the number of evaluation steps, and thus obtain efficient implementations. All implementations of functional programming languages rely on some form of environment.

In a recent work [4], Accattoli and Barras compare various kinds of environments, namely, global, local, and split, from implementative and complexity points of view. The present work complements Accattoli and Barras’ by studying the special role that environments play when terms are put in ANF, or—as we prefer to say—when terms are in abnormal form¹. Abnormal forms roughly are obtained by (recursively) decomposing iterated applications by introducing a sharing point in between any two of them as an environment entry. For instance, the abnormal representation of the term \(((\lambda x.x)y)((\lambda z.z)y)y\) is

\[
(w''y)[w''\leftarrow w'[w'[\leftarroww(\lambda x.x)y)]\leftarrow((\lambda z.z)y)]
\]

where e is the (global) environment. The key point about abnormal forms is that abstract machines no longer need data structures such as the applicative stack or the dump, used to store the evaluation context, that is, to search for the redex in the ordinary applicative structure of λ-terms. These data structures disappear because they are encoded in the sequential structure of the environment via the abnormal transformation, that decomposes iterated applications in environment entries. Abnormal environments then play a double role: they both store delayed substitutions and encode evaluation contexts.

The Value of this Work. In our paper Crumbling Abstract Machines submitted to ICFP 2019 we study abnormal environments in two cases, closed and open CbV evaluation. In both cases we provide detailed studies of both the correctness and the complexity of the implementation; we also provide an OCaml implementation of our machines, thus turning abstract machines into concrete ones. The moral of our study is that the abnormal transformation encodes the stack and the dump of the ordinary abstract machines into abnormal environments, making them a sort of universal data structure. Additionally, the transformation comes at no additional cost and in the open case it actually subsumes an optimisation needed in order to obtain efficient machines.

The removal of data structures may be a fact known to experts, but it is not evident in any of the papers on the subject, and its clear spelling—we believe—is a relevant contribution. Such a removal, however, is not an absolute gain. On the one hand, the form of abnormal terms is less natural, but on the other hand the lack of data structure is compensated by more code to handle the copy of subterms, with respect to the implementation of [4].

¹We prefer abnormal to A-normal because: 1) referring to normal forms is potentially confusing since terms in A-normal form are in general not normal with respect to evaluation; 2) such a special form is the result of a compilation technique and thus less natural / readable than the usual form; 3) it is catchier.
References