Back to Direct Style for Delimited Continuations*

Dariusz Biernacki, Mateusz Pyzik, and Filip Sieczkowski

Inst. of Computer Science, University of Wrocław, Wrocław, Poland
{dabi,matp,efes}@cs.uni.wroc.pl

Abstract

We present an inverse to a CPS transformation for a functional programming language with delimited control operators shift and reset, extending an earlier line of work for simpler calculi. We also study the evaluation behaviour of corresponding programs in direct and continuation-passing styles.

1 Background and overview

Continuation Passing Style (CPS), dating back to Fischer and Plotkin [8, 10], is a format for functional programs where all intermediate results are named, all calls are tail-calls and programs are evaluation-order independent [1]. In a program that adheres to CPS, all procedures take an additional argument, a *continuation*, that represents "the rest of the computation" [2]. CPS may be used, even unknowingly, as a style of hand-written code (e.g., ubiquitous callback arguments in JavaScript) or as an intermediate language in an optimising compiler (e.g., Standard ML of New Jersey). Usage of CPS in compilers would not be possible without a mechanical *CPS transformation*. Such transformation takes an arbitrary program and produces an equivalent program in CPS.

A natural question arises: can we recover the original, *Direct Style (DS)* program from its CPS-compliant derivative? The answer, established by Danvy and extended in joint work with Lawall, is affirmative: one can indeed apply a *DS transformation* to a program in CPS and obtain its DS counterpart [2, 5] when continuations are pure or abortive. In pure CPS, the continuation parameter is called exactly once, in a tail position somewhere inside the procedure body. As a rule of a thumb, all deviations from this pattern should warn us that the control flow in the procedure is nonstandard, and that the author is trying to employ some computational effect: exceptions, Prolog-like backtracking, coroutines etc. In [5], Danvy and Lawall show that we can render some nonstandard uses of continuations (i.e. abortive) in Direct Style by using the `call-with-current-continuation` (or `call/cc` for short) *control operator* [11].

The `call/cc` operator is a prime example of an undelimited (or, *abortive*) control operator: once called, the continuation never returns to its call site – in effect, modelling a jump. In this paradigm, a continuation spans the entire future of the program's execution. However, some computational effects do not behave like jumps: one vivid example is backtracking. In this scenario, continuations model only a fragment of the future program execution, i.e., they are *delimited*, and we use them in non-tail positions, making them *composable*. In direct style, this behaviour is embodied by the control operators `shift` and `reset` whose image under the CPS translation gives rise to composable continuations [4]. The `shift` operator captures the current continuation but – in contrast to the more common `call/cc` – one that extends only up to the dynamically nearest enclosing `reset` operator. Interestingly, these operators have the capability to express *any* computational effect [7] and are closely related to the recent idea of *algebraic effects* [9]. However, in contrast to pure lambda calculus and abortive continuations,

*This work was partially supported by National Science Centre, Poland, grant no. 2014/15/B/ST6/.

the question of a direct-style transformation for a language with composable continuations has not yet been studied.

In this work we attempt to establish such a correspondence. To this end we first introduce a revised CPS transformation for a functional language with delimited control operators `shift` and `reset` that preserves all language redexes. We characterise the image of the translation by an inductive judgement, extending the approach used by Danvy, Lawall and Pfenning [3, 5, 6]. We define an effective Direct Style transformation directed by this judgement and prove that the two transformations are mutual inverses. Moreover, we study the evaluation behaviour of corresponding DS and CPS programs, following the work on Galois reflections for pure call-by-value lambda calculus by Sabry and Wadler [12]. We find that together, CPS and DS transformations establish an isomorphism between Direct Style language and the image of the CPS transformation, where CPS terms are equipped with a custom evalution relation. We are currently working on extending this result to general reduction.

2 Technical details

Direct Style language Our DS language λ_{SRT} is λ_v enriched with delimited control operators `shift`/`reset` together with somewhat artificial but later justified `throw` operator. We distinguish continuations from ordinary functions, i.e. they are not first-class. Continuations are applied using a dedicated `throw` syntax.

CPS transformation Our CPS transformation comes in equivalent second- and first-order formulations which we use interchangeably in proof developments as needed. We follow the general ideas of the *Back to direct style* papers [3, 5], devising inductive judgements which are meant to be provable if and only if a judged expression is in the range of CPS transformation. Moreover, there is at most one matching proof for each expression. To achieve such uniqueness, our CPS is sprinkled with *explicit redexes* which mark the occurences of control operators in CPS.

DS transformation The judgements are designed so that one can effectively recover the corresponding direct style term from a proof-term. In order to devise an effective and not only a theoretical inverse transformation one needs to recover the proof-term from a CPS expression. Such a procedure is neccessarily partial, as not all lambda expressions are in CPS. Most of the cases can be easily distinguished by a shallow inspection of the expression but special care needs to be taken with `shift` and `reset`. These two are hard to differentiate if not for a one peculiarity: serious expressions have a different parity of outermost λ-abstractions than continuation expressions. This subtle difference saves us from trouble and is the primary reason for keeping continuations second-class. Otherwise, distinction would disappear and it would be possible to disprove uniqueness of proof-terms. Uniqueness is crucial here – without it an inverse transformation becomes multi-valued, i.e., many different DS expression might give the same CPS expression. We do not have a decisive answer whether this problem with first-class continuations can be overcome by a different CPS transformation.

Evaluation in CPS Seemingly redundant explicit redexes appear whenever we translate a control operator. They allow us to clearly distinguish between operators before and after evaluation step. This ensures that each step of evaluation in a DS expression shall be mirrored in a CPS transform and *vice versa*. In order to establish a genuine lockstep between these,

we aggregate some β-contractions in the CPS language, making it illegal to stop evaluation in certain moments. We refine the operational semantics in the range of the CPS transformation in order to achieve monotonicity properties; they do not seem to hold when *vanilla* λ_v evaluation is considered.

We define a subrelation of ordinary λ_v reduction on the range of the CPS transformation. Usual β-reductions are allowed only in specific syntactic positions which correspond to application and control operators. This allows us to move along evaluation in CPS just as if we evaluated in Direct Style and performed CPS transformation each time. In a sense, evaluation and transformation commute, or in terms of partial orders, CPS transformation is monotone. We conjecture it should be monotone when appropriate *reduction* relations are considered but this is a topic of an ongoing work.

References

[1] D. Biernacki, O. Danvy, and C. Shan. On the static and dynamic extents of delimited continuations. *Sci. Comput. Program.*, 60(3):274–297, 2006.

[2] O. Danvy. Back to direct style. In B. Krieg-Brückner, editor, *Proceedings of the Fourth European Symposium on Programming*, volume 582 in *Lecture Notes in Computer Science*, pages 130–150. Springer, 1992.

[3] O. Danvy. Back to direct style. *Sci. Comput. Program.*, 22(3):183–195, 1994.

[4] O. Danvy and A. Filinski. Abstracting control. In M. Wand, editor, *Proc. of 1990 ACM Conf. on Lisp and Functional Programming*, pages 151–160. ACM Press, 1990.

[5] O. Danvy and J. L. Lawall. Back to direct style II: First-class continuations. In W. Clinger, editor, *Proc. of 1992 ACM Conf. on Lisp and Functional Programming*, pages 299–310. ACM Press, 1992. Also in *LISP Pointers*, 5(1):299–310.

[6] O. Danvy and F. Pfenning. The occurrence of continuation parameters in CPS terms. Technical report CMU-CS-95-121, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, Feb. 1995.

[7] A. Filinski. Representing monads. In H.-J. Boehm, editor, *Proc. of 21st Ann. ACM Symp. on Principles of Programming Languages*, pages 446–457. ACM Press, 1994.

[8] M. J. Fischer. Lambda-calculus schemata. In *Proc. of ACM Conf. on Proving Assertions about Programs*, pages 104–109. ACM, 1972. Also in *ACM SIGPLAN Not.*, 7(1):104–109, 1972. Reprinted in *Lisp Symb. Comput.*, 6(3–4):259–288, 1993.

[9] Y. Forster, O. Kammar, S. Lindley, and M. Pretnar. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *Proc. ACM Program. Lang.*, 1(ICFP):13:1–13:29, 2017.

[10] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theor. Comput. Sci.*, 1:125–159, 1975.

[11] J. C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proc. of 25th ACM National Conf.*, pages 717–740. ACM, 1972. Reprinted in *Higher-Order Symb. Comput.*, 11(4):363–397, 1998.

[12] A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997.