RETROSPECTIVE:

# The Essence of Compiling with Continuations

Cormac Flanagan
Systems Research Center
HP Laboratories
flanagan@hpl.hp.com

Amr Sabry
Dept. of Computer Science
Indiana University
sabry@indiana.edu

Bruce F. Duba
Dept. of Computer Science
Seattle University
bduba@seattleu.edu

Matthias Felleisen
College of Computer Science
Northeastern University
matthias@ccs.neu.edu

Continuation-passing style (CPS) became a popular intermediate representation for compilers of higher-order functional languages during the 1980's (Rabbit [20], Orbit [13], SML/NJ [3, 4]). The authors of such compilers often cited conventional engineering benefits. Appel [1, p.4] also stressed that one can perform $\beta\eta$-reduction on the CPS intermediate language even though this is unsound on the source language, which uses call-by-value. Indeed this observation is consistent with Plotkin's [15] earlier work who formalized the reasoning principles associated with call-by-value languages before and after CPS conversion. For optimizing a call-by-value source program one can only use $\beta_v\eta_v$ reductions; after conversion to CPS, one can use $\beta\eta$-reductions. Plotkin went on to prove that one can perform strictly more optimizations using $\beta\eta$ on the CPS-converted program than using $\beta_v\eta_v$ on source programs.

This situation provided us with the motivation to study and understand reductions on CPS terms and how they relate to reductions on source programs. Building on Felleisen's work on $\lambda$-calculi [8, 7, 9], Sabry and Felleisen produced a calculus for source programs that exactly corresponds to $\beta\eta$ on CPS terms [17, 18]. The key insight is to relate every transformation step on CPS terms (including the administrative reductions) to a corresponding transformation on source terms. The additional reduction relations correspond to the administrative reductions on CPS terms. Sabry and Felleisen called those the *A*-reductions, and showed that $\beta_v\eta_v + A$ on a call-by-value language is equivalent to $\beta\eta$ on a CPS'ed call-by-name language. Better still, the set of *A* reductions is strongly normalizing, and transforming a source term in *A*-normal form into a continuation-passing style term produces a term without administrative redexes. Sabry and Felleisen called this set of terms A-normal forms (ANF).

Upon further experimentation with the abstract machines developed by Felleisen *et al.* [6], it became clear that everything that CPS compilers do to their intermediate representations could be done just as naturally on A-normal forms. In fact, the abstract machines that define the meaning of the intermediate forms are almost identical. The selected paper describes the result of this theoretical and practical experimentation.

Surprisingly, the paper immediately received much attention in the functional compiler community. The reviews, though, were mixed. A large majority of compiler writers, including those who had been historically dubious of CPS, reported that our paper confirmed their understanding in a precise and formal way. Some of the strong advocates of CPS compilers, however, were unconvinced that our analysis had captured the "essence" of their compilers. In particular our $\beta\eta$ model of CPS optimizations did not capture some of the optimizations that CPS compilers perform. In particular, Appel and Kelsey considered those additional optimizations as an essential part of compiling with continuations.

This criticism motivated a follow-up investigation. In the next year's PLDI, Sabry and Felleisen [19] partially answered the question of the effect of the CPS transformation on the control and data flow analysis. They explain the precise impact of CPS on the results of the most widely used analyses.

One last sticky point still remained to the story. In the initial phase of the compilation, CPS compilers represent continuations as procedures and all calls to known procedures are converted to immediate jumps. Naturally this also converts returns to known continuations to jumps. Because continuations are not explicit in the ANF representation this particular optimization could not be expressed naturally. So in some sense, our model fails to capture part of the "essence" of compiling with continuations. Yet, compiler writers abandoned CPS over the ten years following our paper anyway. This includes the SML/NJ compiler, which was redesigned with a new intermediate form close to ANF (Private communication: Daniel Wang) as well as other compilers written since then [21].

Both ANF and CPS have been shown to be closely related to the SSA form [2, 12]. More recent compilers, such as Moby [16] and MLton [10], exploit this connection by using a mixture of ANF, SSA, and CPS to address the sticky point regarding known continuations: only functions with known continuations are converted to CPS to produce a representation that is closely related to SSA. This enables conventional analyses and transformations to later convert uses of known return continuations to direct jumps. This limited use of CPS is called "contification" [10] or "local CPS conversion" [16].

As a program representation, ANF had success beyond its original role as an intermediate representation suitable for compiling and analyzing functional programs. For example, it became quite standard in the study of partial evaluation [11], and even in the type-theoretic treatment of module systems [5, 14].

In summary, our paper succeeded in making compiler writers reconsider their decisions about intermediate representations. It became clear that their publicly stated reasons for choosing CPS had been invalidated. They had to analyze their decisions in depth. The result is that compilers now mostly use intermediate representations based on ANF but with a local CPS transformation to enable additional optimization. We believe that our theoretical investigation has thus produced a well thought out practical compromise.

## REFERENCES

[1] Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.

*20 Years of the ACM/SIGPLAN Conference on Programming Language Design and Implementation (1979-1999): A Selection,* 2003.

[2] Andrew W. Appel. SSA is functional programming. *ACM SIGPLAN Notices*, 33(4):17–20, April 1998.

[3] Andrew W. Appel and David B. MacQueen. Standard ML of New Jersey. In J. Małuszyński and M. Wirsing, editors, *Proceedings of the 3rd Int. Symposium on Programming Language Implementation and Logic Programming, PLILP91, Passau, Germany*, Lecture Notes in Computer Science, pages 1–13. Springer-Verlag, August 1991.

[4] Andrew W. Appel and David B. MacQueen. Standard ML of New Jersey. Technical Report TR-329-91, Princeton University, Computer Science Department, June 1991.

[5] Matthias Blume and Andrew W. Appel. Lambda-splitting: A higher-order approach to cross-module optimizations. In *Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming*, pages 112–124, Amsterdam, The Netherlands, 9–11 June 1997.

[6] M. Felleisen and D. P. Friedman. *Control operators, the SECD-machine, and the λ-calculus*, pages 193–217. North-Holland, 1986.

[7] Matthias Felleisen. λ-v-CS: An extended λ-calculus for Scheme. In *Proc. of 1988 ACM Conf. on Lisp and Functional Programming, Snowbird, UT, USA, 25–27 July 1988*, pages 72–85. ACM Press, New York, 1988.

[8] Matthias Felleisen, Daniel P. Friedman, Eugene Kohlbecker, and Bruce Duba. Reasoning with continuations. In *Proc. of 1st Ann. IEEE Symp. on Logic in Computer Science, LICS'86, Cambridge, MA, USA, 16–18 June 1986*, pages 131–141. IEEE Computer Society Press, Washington, DC, 1986.

[9] Matthias Felleisen and Robert Hieb. A revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.

[10] Matthew Fluet and Stephen Weeks. Contification using dominators. In Cindy Norris and Jr. James B. Fenwick, editors, *Proceedings of the Sith ACM SIGPLAN International Conference on Functional Programming (ICFP-01)*, volume 36, 10 of *ACM SIGPLAN notices*, pages 2–13, New York, September 3–5 2001. ACM Press.

[11] John Hatcliff and Olivier Danvy. A computational formalization for partial evaluation. *Mathematical Structures in Computer Science*, 7(5):507–541, October 1997.

[12] Richard A. Kelsey. A correspondence between continuation passing style and static single assignment form. *ACM SIGPLAN Notices*, 30(3):13–22, March 1995.

[13] David Kranz, , Richard Kelsey, Jonathan Rees, Paul Hudak, James Philbin, and Norman Adams. Orbit: An optimizing compiler for Scheme. *SIGPLAN Notices*, 21(7):219–233, July 1986. *Proceedings of the ACM SIGPLAN '86 Symposium on Compiler Construction.*

[14] Xavier Leroy. A syntactic theory of type generativity and sharing. *Journal of Functional Programming*, 6(5):667–698, September 1996.

[15] G. Plotkin. Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.

[16] John Reppy. Local CPS conversion in a direct-style compiler. In *Proceedings of the Third ACM SIGPLAN Workshop on Continuations (CW'01)*, pages 13–22, January 2001.

[17] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In *Proc. of 1992 ACM Conf. on Lisp and Functional Programming, San Francisco, CA, USA, 22–24 June 1992*, pages 288–298. ACM Press, New York, 1992.

[18] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3–4):289–360, 1993.

[19] Amr Sabry and Matthias Felleisen. Is continuation-passing useful for data flow analysis? In *Proceedings of the Conference on Programming Language Design and Implementation*, pages 1–12, New York, NY, USA, June 1994. ACM Press.

[20] Jr. Steele, Guy L. Rabbit: A compiler for Scheme. Technical Report AITR-474, Massachusetts Institute of Technology, May 1978.

[21] D. Tarditi, G. Morrisett, P. Cheng, C. Stone, R. Harper, and P. Lee. TIL : A type-directed optimizing compiler for ML. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implemantation*, pages 181–192, New York, May 21–24 1996. ACM Press.