Delimited continuations

1 Readings

1.1 Set papers

The week's set papers are as follows:

- *Delimited control in OCaml, abstractly and concretely* (Kiselyov, 2012)
- *Implementing first-class polymorphic delimited continuations by a type-directed selective CPS-transform* (Rompf, Maier, and Odersky, 2009)
- *Continuing WebAssembly with Effect Handlers* (Phipps-Costin, Rossberg, Guha, et al., 2023)

You are invited to write an essay about two of these papers, following the guidelines on the course assessment page [1].

1.2 Background

Appendix A of *Foundations for Programming and Implementing Effect Handlers* (Hillerström, 2021) is a recent overview of delimited continuations, with details on the history, on the various operators with their semantics and types, on applications, and on implementation approaches.

2 History

2.1 Beginnings

Continuations were used to define the semantics of jumps in programming languages from the early years of computer science (*The Discoveries of Continuations* (Reynolds, 1993) lists some examples in the early 1960s). *Undelimited* continuations as a programming construct were introduced in *Definitional interpreters for higher-order programming languages* (Reynolds, 1972), and available in early versions of Scheme (Steele Jr and Sussman, 1975).

2.2 Developments

1987 *Beyond Continuations* (Felleisen, Friedman, Duba, et al., 1987) and *Abstract Continuations: A Mathematical Semantics for Handling Full Jumps* (Felleisen, Wand, Friedman, et al., 1988) noted some complications with the axioms for undelimited continuations, and introduced a delimited form, $\mathcal{F}$ (`control`).

1989 *A Functional Abstraction of Typed Contexts* (Danvy and Filinski, 1989) introduced an alternative operator for capturing delimited continuations, `shift`, noting that it could be statically typed.

1995 *A Generalization of Exceptions and Control in ML-like Languages* (Gunter, Rémy, and Riecke, 1995) introduced yet another delimited control operator, `cupto`, integrated into an ML-style language and type system.

2007 *Polymorphic Delimited Continuations* (Asai and Kameyama, 2007) introduced a type system for a language with delimited continuations that distinguishes between effectful expressions (that use `shift` and `reset`) and pure expressions (that do not).

2.3 Status

Delimited continuations have proved to be a very general mechanism for implementing effects of various types. (*Representing Monads* (Filinski, 1994) shows that they can be used to embed any expressible monad.) They have been usefully employed, for example, to simulate exceptions, nondeterminism, mutable state, concurrency, generators, and in more specialized applications in probabilistic programming, automatic differentiation, partial evaluation, and linguistics.

However, despite these many applications and despite the long history of study, relatively few general-purpose languages provide built-in support for delimited continuations. OCaml 5 is one exception, and there is a proposal under consideration to add support for delimited continuations to WebAssembly (Phipps-Costin et al., 2023), where it may be used as a uniform basis for a variety of non-local control features including threads and async/await.

[1] `https://www.cl.cam.ac.uk/teaching/2425/R277/assessment.html`

Values

V	$::=$	x	*(variable)*
	$\mid$	$\lambda x.M$	*(abstraction)*

Terms

L, M	$::=$	V	*(value)*	$\mid$	$L\ M$	*(application)*
	$\mid$	$\langle M \rangle$	*(reset)*	$\mid$	$\mathcal{S}\, k.M$	*(shift)*

Reductions

$$E[(\lambda x.M)\ V] \rightsquigarrow E[M\{V/x\}]$$
$$E[\langle V \rangle] \rightsquigarrow E[V]$$
$$E[\langle E_2[\mathcal{S}\, k.M] \rangle] \rightsquigarrow E[\langle M\{(\lambda y.\langle E_2[y]\rangle)/k\}\rangle]$$

NB: $E_2[-]$ *must not enclose its body in* $\langle - \rangle$

Contexts

$$E[\cdot] ::= [\cdot] \mid E[[\cdot]\ M] \mid E[V\ [\cdot]] \mid E[\langle[\cdot]\rangle]$$

Figure 1: Delimited continuations: syntax and evaluation

Enclose captured context E_2*?*

$E[\langle E_2[?\, k.M]\rangle] \rightsquigarrow$

	✓ (enclose E_2)	✗ (enclose E_2)	*retain enclosing reset?*
	shift ($? = \mathcal{S}$) $E[\langle M\{(\lambda y.\langle E_2[y]\rangle)/k\}\rangle]$	*control* ($? = \mathcal{F}$) $E[\langle M\{(\lambda y.E_2[y])/k\}\rangle]$	✓
	shift0 ($? = \mathcal{S}_0$) $E[M\{(\lambda y.\langle E_2[y]\rangle)/k\}]$	*control0* ($? = \mathcal{F}_0$) $E[M\{(\lambda y.E_2[y])/k\}]$	✗

Figure 2: Alternatives to *shift*

3 Basics

Figure 1 defines a minimal lambda calculus with support for delimited continuations. Values are either variables or lambda abstractions; terms are either values, applications, or can be built from the two delimited control constructs, *reset* and *shift*. As the reduction rules show, these two operators work in concert: $\mathcal{S}\, k.M$ captures the context up to the nearest enclosing $\langle - \rangle$ and inserts it into M in place of the bound variable k.

Variants of these rules are given by many authors, including Kiselyov, Shan, and Sabry (2006) and Ishio and Asai (2023).

3.1 Operator taxonomy

There are three common alternatives to *shift* (Figure 2), which vary (on the right of the reduction rule) as to whether they retain the *reset* in the current context and whether they enclose the captured context. Ishio and Asai (2023) and many others give details.

3.2 Examples

We start by showing how delimited continuations can implement exceptions. The expression try b h first evaluates b (); if that concludes without invoking a control operator, its value becomes the value of the whole expression. However, b () might also call raise e, discarding the continuation k up to the call to b and passing e to the handler function h:

$$\text{try } b\ h = \text{case } \langle R\ (b\ ())\rangle \text{ of } L\ e \to h\ e \mid R\ v \to v$$
$$\text{raise } e = \mathcal{S}_0\, k.L\ e$$

We next show an example that turns an arbitrary iterator (e.g. OCaml's `List.iter`) into a generator like Python's `yield`:

$$\text{generate iter } l = \langle \text{iter } (\lambda v.\mathcal{S}_0\, k.(v,k))l \rangle$$

Here generate iter l installs a reset and calls the function iter, passing it a function that captures the continuation k up to the reset, passing both the k and the function argument v. The caller can process the value v and re-invoke the continuation k continuation to resume the iteration.

Finally, we show an example that distinguishes various control operators:

$$\langle 1 + \langle \mathcal{S}\, k_1.k_1\ 100 + k_1\ 10 + \mathcal{S}\, k_2.\mathcal{S}\, k_3.1\rangle\rangle$$

As written, this example reduces to 3; however, it can be modified to instead reduce to 1 or 2 by respectively changing $\mathcal{S}$ to $\mathcal{S}_0$ or to $\mathcal{F}$ throughout.

4 Implementations and applications

4.1 Implementation considerations

4.1.1 Types

Type systems for delimited continuations have been studied since at least Danvy and Filinski (1989), but remain challenging due to *answer-type modification*, i.e. changes to the return type of continuations arising from use of `shift`. Kobori, Kameyama, and Kiselyov (2015) give details and a type-preserving translation that eliminates answer-type modification.

4.1.2 Named prompts

Since delimited control operators like `shift` capture continuations up to the nearest enclosing $\langle - \rangle$, multiple uses of `shift` within a program can sometimes interact unexpectedly. To help avoid such interactions, some implementations support named reset points that allow `shift` to indicate explicitly how much of the enclosing continuation to capture.

4.1.3 Multi-shot vs one-shot

Implementations of delimited continuations vary in whether they allow captured continuations to be invoked multiple times (so-called *multi-shot* continuations), or at most once (*single-shot*).

4.2 Implementations

There are a variety of approaches to implementing delimited continuations, many of which involve changing compilation (e.g. to transform programs into continuation-passing style), modifying the language runtime (e.g. to provide primitives for manipulating stacks), or both. Other implementation strategies rely on the expressibility of delimited continuations in terms of other language constructs such as threads or undelimited continuations.

- *Threads Yield Continuations* (Kumar, Bruggeman, and Dybvig, 1998) implements one-shot and multi-shot delimited continuations using threads. (In the other direction, delimited continuations can implement cooperative threads (Dolan et al., 2015).)
- *Representing Monads* (Filinski, 1994) shows how to implement `shift` and `reset` using undelimited continuations and mutable reference cells.
- *Final shift for call/cc: direct implementation of shift and reset* (Gasbichler and Sperber, 2002) gives a direct Scheme48 implementation of `shift` and `reset`, and shows improved efficiency over a simulation using undelimited continuations.
- *Continuations from generalized stack inspection* (Pettyjohn, Clements, Marshall, et al., 2005) uses exceptions and code that saves and restores local data to simulate delimited control
- *Implementing first-class polymorphic delimited continuations by a type-directed selective CPS-transform* (Rompf, Maier, and Odersky, 2009) uses the type system of Asai and Kameyama (2007) to CPS-convert only those parts of a program that actually make use of control effects.
- *Delimited control in OCaml, abstractly and concretely* (Kiselyov, 2012) implements delimited continuations as an OCaml library using low-level stack-manipulation primitives, without modifying the language implementation.
- *Effect Handlers for C via Coroutines* (Alvarez-Picallo, Freund, Ghica, et al., 2024) describes an effect handlers library for C, implemented using coroutines

4.3 Applications

Support for interactions between programs and their contexts has many applications.

In *partial evaluation* and its more explicit variant *multi-stage programming*, delimited continuations can be used to rearrange program fragments so that they specialize more effectively (Lawall and Danvy, 1994; Kameyama, Kiselyov, and Shan, 2011).

In *linguistics*, lambda-calculus based formalisms such as Combinatory Categorical Grammar extended with delimited continuations can capture interactions between linguistic expressions and scope-sensitive phenomena (e.g. quantifiers) (Shan, 2004).

Reverse-mode automatic differentiation can be implemented more straightforwardly using delimited continuations than with traditional methods such as tapes (Wang et al., 2019).

Delimited continuations have also been used to effectively embed domain-specific languages for *probabilistic programming* in general purpose host languages (Kiselyov and Shan, 2009).

4.4 Connection to effect handlers

Effect handlers are a recent approach to supporting user-defined effects. They are close in expressive power to delimited continuations, but have a more convenient programming interface that resembles resumable exceptions. *On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control* (Forster, Kammar, Lindley, et al., 2017) gives a careful expressivity comparison; *Handlers in action* (Kammar, Lindley, and Oury, 2013) shows how delimited continuations can be used to implement effect handlers.

References

Alvarez-Picallo, M., T. Freund, D. R. Ghica, and S. Lindley (Oct. 2024). "Effect Handlers for C via Coroutines". In: *Proc. ACM Program. Lang.* 8.OOPSLA2. DOI: 10.1145/3689798.

Asai, K. and Y. Kameyama (2007). "Polymorphic Delimited Continuations". In: *Programming Languages and Systems, 5th Asian Symposium, APLAS 2007, Singapore, November 29-December 1, 2007, Proceedings*. Ed. by Z. Shao. Vol. 4807. Lecture Notes in Computer Science. Springer, pp. 239–254. DOI: 10.1007/978-3-540-76637-7_16.

Danvy, O. and A. Filinski (1989). *A Functional Abstraction of Typed Contexts*. Tech. rep. Universitetsparken 1, 2100 Copenhagen Ø, Denmark: DIKU — Computer Science Department, University of Copenhagen.

Dolan, S., L. White, K. Sivaramakrishnan, J. Yallop, and A. Madhavapeddy (Sept. 2015). *Effective Concurrency with Algebraic Effects*. OCaml Users and Developers Workshop.

Felleisen, M., D. P. Friedman, B. Duba, and J. Merrill (Feb. 1987). *Beyond Continuations*. Tech. rep. 216. Lindley Hall 101, Bloomington, IN 47405, USA: Computer Science Department, Indiana University.

Felleisen, M., M. Wand, D. P. Friedman, and B. F. Duba (1988). "Abstract Continuations: A Mathematical Semantics for Handling Full Jumps". In: *Proceedings of the 1988 ACM Conference on LISP and Functional Programming, LFP 1988, Snowbird, Utah, USA, July 25-27, 1988*. Ed. by J. Chailloux. ACM, pp. 52–62. DOI: 10.1145/62678.62684.

Filinski, A. (1994). "Representing Monads". In: *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*. Ed. by H. Boehm, B. Lang, and D. M. Yellin. ACM Press, pp. 446–457. DOI: 10.1145/174675.178047.

Forster, Y., O. Kammar, S. Lindley, and M. Pretnar (2017). "On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control". In: *Proc. ACM Program. Lang.* 1.ICFP, 13:1–13:29. DOI: 10.1145/3110257.

Gasbichler, M. and M. Sperber (2002). "Final shift for call/cc: direct implementation of shift and reset". In: *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002*. Ed. by M. Wand and S. L. P. Jones. ACM, pp. 271–282. DOI: 10.1145/581478.581504.

Gunter, C. A., D. Rémy, and J. G. Riecke (1995). "A Generalization of Exceptions and Control in ML-like Languages". In: *Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995*. Ed. by J. Williams. ACM, pp. 12–23. DOI: 10.1145/224164.224173.

Hillerström, D. (2021). "Foundations for Programming and Implementing Effect Handlers". PhD thesis. The University of Edinburgh.

Ishio, C. and K. Asai (2023). "Type System for Four Delimited Control Operators". In: *CoRR* abs/2305.02852. DOI: 10.48550/arXiv.2305.02852. arXiv: 2305.02852.

Kameyama, Y., O. Kiselyov, and C. Shan (2011). "Shifting the stage - Staging with delimited control". In: *J. Funct. Program.* 21.6, pp. 617–662. DOI: 10.1017/S0956796811000256.

Kammar, O., S. Lindley, and N. Oury (2013). "Handlers in action". In: *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*. Ed. by G. Morrisett and T. Uustalu. ACM, pp. 145–158. DOI: 10.1145/2500365.2500590.

Kiselyov, O. (2012). "Delimited control in OCaml, abstractly and concretely". In: *Theor. Comput. Sci.* 435, pp. 56–76. DOI: 10.1016/j.tcs.2012.02.025.

Kiselyov, O. and C. Shan (2009). "Embedded Probabilistic Programming". In: *Domain-Specific Languages, IFIP TC 2 Working Conference, DSL 2009, Oxford, UK, July 15-17, 2009, Proceedings*. Ed. by W. M. Taha. Vol. 5658. Lecture Notes in Computer Science. Springer, pp. 360–384. DOI: 10.1007/978-3-642-03034-5_17.

Kiselyov, O., C. Shan, and A. Sabry (2006). "Delimited dynamic binding". In: *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming, ICFP 2006, Portland, Oregon, USA, September 16-21, 2006*. Ed. by J. H. Reppy and J. Lawall. ACM, pp. 26–37. DOI: 10.1145/1159803.1159808.

Kobori, I., Y. Kameyama, and O. Kiselyov (2015). "Answer-Type Modification without Tears: Prompt-Passing Style Translation for Typed Delimited-Control Operators". In: *Proceedings of the Workshop on Continuations, WoC 2016, London, UK, April 12th 2015*. Ed. by O. Danvy and U. de'Liguoro. Vol. 212. EPTCS, pp. 36–52. DOI: 10.4204/EPTCS.212.3.

Kumar, S., C. Bruggeman, and R. K. Dybvig (1998). "Threads Yield Continuations". In: *LISP Symb. Comput.* 10.3, pp. 223–236.

Lawall, J. L. and O. Danvy (1994). "Continuation-Based Partial Evaluation". In: *Proceedings of the 1994 ACM Conference on LISP and Functional Programming, Orlando, Florida, USA, 27-29 June 1994*. Ed. by R. R. Kessler. ACM, pp. 227–238. DOI: 10.1145/182409.182483.

Pettyjohn, G., J. Clements, J. Marshall, S. Krishnamurthi, and M. Felleisen (2005). "Continuations from generalized stack inspection". In: *Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming, ICFP 2005, Tallinn, Estonia, September 26-28, 2005*. Ed. by O. Danvy and B. C. Pierce. ACM, pp. 216–227. DOI: 10.1145/1086365.1086393.

Phipps-Costin, L., A. Rossberg, A. Guha, D. Leijen, D. Hillerström, K. Sivaramakrishnan, M. Pretnar, and S. Lindley (2023). "Continuing WebAssembly with Effect Handlers". In: *Proc. ACM Program. Lang.* 7.OOPSLA.

Reynolds, J. C. (1972). "Definitional interpreters for higher-order programming languages". In: *Proceedings of the ACM annual conference, ACM 1972, 1972, Volume 2*. Ed. by J. J. Donovan and R. Shields. ACM, pp. 717–740. DOI: 10.1145/800194.805852.

Reynolds, J. C. (1993). "The Discoveries of Continuations". In: *LISP Symb. Comput.* 6.3-4, pp. 233–248.

Rompf, T., I. Maier, and M. Odersky (2009). "Implementing first-class polymorphic delimited continuations by a type-directed selective CPS-transform". In: *Proceeding of the 14th ACM SIGPLAN international conference on Functional programming, ICFP 2009, Edinburgh, Scotland, UK, August 31 - September 2, 2009*. Ed. by G. Hutton and A. P. Tolmach. ACM, pp. 317–328. DOI: 10.1145/1596550.1596596.

Shan, C. (2004). "Delimited continuations in natural language: quantification and polarity sensitivity". In: *CoRR* cs.CL/0404006.

Steele Jr, G. L. and G. J. Sussman (1975). *Scheme, an interpreter for extended lambda calculus*. Tech. rep. 349. Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Wang, F., D. Zheng, J. M. Decker, X. Wu, G. M. Essertel, and T. Rompf (2019). "Demystifying differentiable programming: shift/reset the penultimate backpropagator". In: *Proc. ACM Program. Lang.* 3.ICFP, 96:1–96:31. DOI: 10.1145/3341700.