Extracting the Range of CPS from Affine Typing

Extended Abstract

Josh Berdine, Peter W. O'Hearn
Queen Mary, University of London
{berdine, ohearn}@dcs.qmul.ac.uk

Hayo Thielecke
The University of Birmingham
H.Thielecke@cs.bham.ac.uk

Abstract

Increasing degrees of reasoning about programs are being mechanized, and hence more formality is needed. Here we present an instance of this formalization in the form of a precise characterization of the range of the CPS transformation using an affine type system. The point is that the range of CPS is defined with a type system, that is, in a way quite accessible to tools.

1 Introduction

Increasing degrees of reasoning about programs are being mechanized. For instance, witness the trend: C, ML, Proof Carrying Code, and beyond. In order to support this, the properties to be reasoned about must be expressed in a language which tools can handle, instead of the informal and highly undecidable language of mathematical discourse. For type preserving or certifying compilers; or other automated (not necessarily automatic) checkers, analyzers, or verifiers; knowing a property holds, even with a proof, isn't enough: the property must be captured by the tool's formalism. If an inexpressible property is taken advantage of, then at some point the tool will require reasoning outside its formal system, causing problems. Hence more formality is needed.

Here we present an instance of this formalization of useful properties in the form of a precise characterization of the range of the CPS transformation using an affine type system. Here, by precise we mean that all terms in the target language come from some term in the source language. This provides a sort of completeness or fullness result (no junk), which while not the most theoretically desirable due to the heavy restriction on types and its syntactic nature, ought to be good enough to reduce loss of precision in compiler analyses. But the point is that the junk-free target language is defined with a type system, that is, in a way quite accessible to tools.

As an example of a property this formalization may allow to be exploited, in a certifying compiler which wishes to stack-allocate activation records, the type systems of the intermediate languages must be tight enough to exclude all programs which violate this discipline. Unlike in a standard compiler, it is not sufficient to simply "know" that none of the intermediate language programs which actually come from earlier stages of the compiler will exhibit such behavior: in a certifying compiler the stage which makes this assumption will generate code which fails to typecheck, and hence no proof of the generated code can be produced.

2 Affine CPS Interpretation

The source language we consider is the usual call-by-value, left-to-right, λ-calculus. One standard CPS interpretation is Fischer's continuation-first one [5] in which source procedures are interpreted with the type:

$$\mu D.\,(D \to \mathbf{R}) \to (D \to \mathbf{R})$$

As investigated by Berdine *et.al.* [2], this interpretation can be refined to capture the stylized use of continuations inherent in the procedure call/return mechanism:

$$D \overset{\text{def}}{=} \mu D.\,(D \to \mathbf{R}) \multimap (D \to \mathbf{R})$$

As usual, toplevel programs are interpreted by answers parameterized by a toplevel continuation:

$$(D \to \mathbf{R}) \multimap \mathbf{R}$$

Functions of affine arrow ($\multimap$) type cannot duplicate their argument. Hence, continuation arguments of affine functions cannot be invoked multiple times. This serves to capture the control-flow property

Once a procedure returns, it will not return again until it is called again.

by making the CPS versions of such procedures inexpressible.

Now, before we have defined the translation of terms, we know enough about the CPS transformation to characterize its range.

3 Range of CPS

First we briefly present an affine variant of the target language of the CPS transformation used in [2]. Types are given by the grammar:

$$\begin{array}{ll} P ::= \mathbf{R} \mid A \to P \mid P \multimap P \mid X \mid \mu X.\,P & \textit{pointed types} \\ A ::= \mathbf{N} \mid P & \textit{types} \end{array}$$

and the typing judgment is given by the axioms and rules:

$$\frac{}{\Gamma\,,x:A\,;\Delta \vdash x:A} \qquad \frac{}{\Gamma\,;\Delta\,,x:P \vdash x:P} \qquad \frac{\Gamma\,;\Delta \vdash M:B}{\Gamma\,;\Delta \vdash M:A}\,B=A$$

$$\frac{\Gamma\,,x:A\,;\Delta \vdash M:P}{\Gamma\,;\Delta \vdash \lambda x.\,M:A \to P} \qquad \frac{\Gamma\,;\Delta \vdash M:A \to P \quad \Gamma\,;- \vdash N:A}{\Gamma\,;\Delta \vdash M\,N:P}$$

$$\frac{\Gamma\,;\Delta\,,x:P \vdash M:Q}{\Gamma\,;\Delta \vdash \delta x.\,M:P \multimap Q} \qquad \frac{\Gamma\,;\Delta \vdash M:P \multimap Q \quad \Gamma\,;\Delta' \vdash N:P}{\Gamma\,;\Delta\,,\Delta' \vdash M \smile N:Q}$$

This language is sufficiently general to transform many source languages into, but our purpose here is to characterize the range, not the codomain, of the CPS transformation. So we specialize this system by restricting the form of types to only those exercised by the CPS interpretation. That is, we restrict the grammar of types to:

$$A, P ::= D \to \mathbf{R} \mid D \mid (D \to \mathbf{R}) \multimap D \to \mathbf{R} \mid (D \to \mathbf{R}) \multimap \mathbf{R} \mid \mathbf{R} \quad \textit{types}$$

Note that there is only a single type identifier, and only a single unfolding of the recursive continuation transformer type is needed, or allowed.

The typing rules specialized to the restricted form of types are (where $\neg D$ abbreviates $D \rightarrow \mathbf{R}$):

$$\frac{\Gamma \,;\, \Delta \,,\, k : \neg D \vdash_{\mathrm{s}} M : \mathbf{R}}{\Gamma \,;\, \Delta \vdash_{\mathrm{s}} \delta k.\, M : \neg D \multimap \mathbf{R}} \qquad \frac{}{\Gamma \,;\, \Delta \,,\, k : \neg D \vdash_{\mathrm{s}} k : \neg D}$$

$$\frac{\Gamma \,,\, x : D \,;\, \Delta \vdash_{\mathrm{s}} M : \mathbf{R}}{\Gamma \,;\, \Delta \vdash_{\mathrm{s}} \lambda x.\, M : \neg D} \qquad \frac{\Gamma \,;\, \Delta \vdash_{\mathrm{s}} M : \neg D \multimap \neg D \quad \Gamma \,;\, \Delta' \vdash_{\mathrm{s}} N : \neg D}{\Gamma \,;\, \Delta \,,\, \Delta' \vdash_{\mathrm{s}} M \llcorner N : \neg D}$$

$$\frac{}{\Gamma \,,\, x : D \,;\, \Delta \vdash_{\mathrm{s}} x : D} \qquad \frac{\Gamma \,;\, \Delta \,,\, k : \neg D \vdash_{\mathrm{s}} M : \neg D}{\Gamma \,;\, \Delta \vdash_{\mathrm{s}} \delta k.\, M : \neg D \multimap \neg D} \qquad \begin{array}{c} \Gamma \,;\, \Delta \vdash_{\mathrm{s}} M : \neg D \multimap \neg D \\ \hline\hline \Gamma \,;\, \Delta \vdash_{\mathrm{s}} M : D \end{array}$$

$$\frac{\Gamma \,;\, \Delta \vdash_{\mathrm{s}} M : \neg D \quad \Gamma \,;\, - \vdash_{\mathrm{s}} N : D}{\Gamma \,;\, \Delta \vdash_{\mathrm{s}} M\, N : \mathbf{R}} \qquad \frac{\Gamma \,;\, \Delta \vdash_{\mathrm{s}} M : \neg D \multimap \mathbf{R} \quad \Gamma \,;\, \Delta' \vdash_{\mathrm{s}} N : \neg D}{\Gamma \,;\, \Delta \,,\, \Delta' \vdash_{\mathrm{s}} M \llcorner N : \mathbf{R}}$$

Note that $\Gamma \,;\, \Delta \vdash_{\mathrm{s}} M : A$ if and only if $\Gamma \,;\, \Delta \vdash M : A$ and all types which appear in the derivation of the latter judgment are in the restricted type language.

Observe that if $\Gamma \,;\, \Delta \vdash_{\mathrm{s}} M : A$, then $\Delta = -$ or $\Delta = k : \neg D$. This indicates that with the restricted form of types, affine, linear, and ordered linear typing all coincide.[1] We use an affine system since, for the full calculus, it yields the strongest no junk result. Taking this special form of contexts into account, we can further specialize the presentation of the typing rules:

$$\frac{\Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} M : \mathbf{R}}{\Gamma \,;\, - \vdash_{\mathrm{s}} \delta k.\, M : \neg D \multimap \mathbf{R}} \qquad \frac{}{\Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} k : \neg D}$$

$$\frac{\Gamma \,,\, x : D \,;\, k : \neg D \vdash_{\mathrm{s}} M : \mathbf{R}}{\Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} \lambda x.\, M : \neg D} \qquad \frac{\Gamma \,;\, - \vdash_{\mathrm{s}} M : \neg D \multimap \neg D \quad \Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} N : \neg D}{\Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} M \llcorner N : \neg D}$$

$$\frac{}{\Gamma \,,\, x : D \,;\, - \vdash_{\mathrm{s}} x : D} \qquad \frac{\Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} M : \neg D}{\Gamma \,;\, - \vdash_{\mathrm{s}} \delta k.\, M : \neg D \multimap \neg D} \qquad \begin{array}{c} \Gamma \,;\, - \vdash_{\mathrm{s}} M : \neg D \multimap \neg D \\ \hline\hline \Gamma \,;\, - \vdash_{\mathrm{s}} M : D \end{array}$$

$$\frac{\Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} M : \neg D \quad \Gamma \,;\, - \vdash_{\mathrm{s}} N : D}{\Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} M\, N : \mathbf{R}} \qquad \frac{\Gamma \,;\, - \vdash_{\mathrm{s}} M : \neg D \multimap \mathbf{R} \quad \Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} N : \neg D}{\Gamma \,;\, k : \neg D \vdash_{\mathrm{s}} M \llcorner N : \mathbf{R}}$$

From this type system we can now extract a grammar where we have one syntactic category for terms of each type:

$$\begin{array}{llll} K ::= k \mid \lambda x.\, P \mid W \llcorner K & \neg D & T ::= \delta k.\, P & \neg D \multimap \mathbf{R} \\ W ::= x \mid \delta k.\, K & D = \neg D \multimap \neg D & P ::= K\, W \mid T \llcorner K & \mathbf{R} \end{array}$$

If we draw k from a singleton set of identifiers which is disjoint from the set of identifiers from which x is drawn, then the type system and syntax are equivalent. That is, a term will be an element of a syntactic category if and only if there is a judgment proving the term has the associated type.

[1] Note, however, that while we only consider continuation usage, Polakow and Pfenning [9] also distinguish continuation arguments from function arguments, and refine the treatment of the former using an ordered system. An unordered system such as ours will not suffice for such analyses.

Sabry and Felleisen's analysis of the range of CPS [10] led them to nearly the same syntax. The only difference is that their transformation does not introduce "administrative" redexes [8], and so syntactic category T is not needed, but toplevel programs must be treated specially. This is interesting since Sabry and Felleisen analyzed the syntax of the output of the CPS transformation, while we have analyzed the types of the output of the CPS transformation, obtaining the same result (modulo "administrative" redexes). So this work can be seen as a logical reconstruction of Sabry and Felleisen's: their methods are syntactic, or even lexical (one continuation identifier is enough), while our starting point is a domain equation with, crucially, an affine function space constraining continuation usage.

4 No Junk

We now establish the precision of our characterization of the range of CPS by stating that all terms in the target language come from some term in the source language. Before stating the result though, we need to present the affine CPS transformation of terms:

$$\overline{x} \stackrel{\text{def}}{=} \delta k.\, k\, x \qquad \overline{\lambda x.\, M} \stackrel{\text{def}}{=} \delta k.\, k\, \delta h.\, \lambda x.\, \overline{M} \sqcup h \qquad \overline{M\, N} \stackrel{\text{def}}{=} \delta k.\, \overline{M} \sqcup \lambda m.\, \overline{N} \sqcup \lambda n.\, m \sqcup k\, n$$

Proposition 1 (No Junk) *If $\Gamma\,;\, - \vDash_{\mathrm{s}} M : (D \to \mathbf{R}) \multimap \mathbf{R}$, then there exists a source term N such that $M \stackrel{\beta\eta}{=} \overline{N}$.*

The proof proceeds by constructing an appropriate source term N using a direct style transformation, much like Sabry and Felleisen's:

$$\begin{aligned}
\mathcal{K}(\!|k|\!) &\stackrel{\text{def}}{=} \lambda x.\, x & \mathcal{W}(\!|x|\!) &\stackrel{\text{def}}{=} x \\
\mathcal{K}(\!|\lambda x.\, M|\!) &\stackrel{\text{def}}{=} \lambda x.\, \mathcal{P}(\!|M|\!) & \mathcal{W}(\!|\delta k.\, M|\!) &\stackrel{\text{def}}{=} \mathcal{K}(\!|M|\!) \\
\mathcal{K}(\!|M \sqcup N|\!) &\stackrel{\text{def}}{=} \lambda x.\, \mathcal{K}(\!|N|\!)\, (\mathcal{W}(\!|M|\!)\, x) & \mathcal{P}(\!|M\, N|\!) &\stackrel{\text{def}}{=} \mathcal{K}(\!|M|\!)\, \mathcal{W}(\!|N|\!) \\
\mathcal{T}(\!|\delta k.\, M|\!) &\stackrel{\text{def}}{=} \mathcal{P}(\!|M|\!) & \mathcal{P}(\!|M \sqcup N|\!) &\stackrel{\text{def}}{=} \mathcal{K}(\!|N|\!)\, \mathcal{T}(\!|M|\!)
\end{aligned}$$

Crucially, the DS transformation is inverse to the CPS transformation:

Lemma 2 *1. If $\Gamma\,;\, k : \neg D \vDash_{\mathrm{s}} M : \neg D$, then $\overline{\mathcal{K}(\!|M|\!)} \sqcup N \stackrel{\beta\eta}{=} N\, \delta k.\, M$.*

2. If $\Gamma\,;\, - \vDash_{\mathrm{s}} M : D$ or $\Gamma\,;\, - \vDash_{\mathrm{s}} M : \neg D \multimap \neg D$, then $\overline{\mathcal{W}(\!|M|\!)} \sqcup N \stackrel{\beta\eta}{=} N\, M$.

3. If $\Gamma\,;\, - \vDash_{\mathrm{s}} M : \neg D \multimap \mathbf{R}$, then $\overline{\mathcal{T}(\!|M|\!)} \stackrel{\beta\eta}{=} M$.

4. If $\Gamma\,;\, k : \neg D \vDash_{\mathrm{s}} M : \mathbf{R}$, then $\overline{\mathcal{P}(\!|M|\!)} \stackrel{\beta\eta}{=} \delta k.\, M$.

Laird [7] has generalized this result and semantically proven full abstraction of the affine CPS transformation. He considers the transformation from untyped λ-calculus into a full (without restricted types) calculus, and uses observational equivalence rather than $\beta\eta$-equality.

5 Exceptions

[Space permitting, we will briefly present an analogous analysis of the range of the CPS transformation of a language with basic exception-handling primitives.]

6 Future Directions

In order to make use of a type system such as that studied here in the context of certifying compilers, we also need a lower-level language and type system into which transformations that exploit properties ensured by the higher-level type system can be defined such that typing derivations can also be translated. So while Danvy [4] and Polakow and Pfenning [9] have proofs that continuations can be stack-allocated, we would like to design a type-system for stack-manipulating code and show how terms and types are translated.

It would also be interesting to test the hypothesis that using a junk-free CPS language improves the precision of static analyses, for example by appropriately modifying Shivers' control-flow analysis [12], and relating the findings to Sabry and Felleisen's [11]. Also interesting would be to extend the source language such that some continuations are used affinely and some not, and then show that, for example Bruggeman Waddell and Dybvig's efficient representation [3] is safe for the affinely used continuations, without relying on run-time checks.

Sabry and Felleisen use a very similar CPS language to discover a complete equational theory for call-by-value λ-calculus [10]. It may be interesting to perform a similar analysis for exceptions, and compare with related work such as Benton and Kennedy's [1].

References

[1] N. Benton and A. Kennedy. Exceptional syntax. *Journal of Functional Programming*, 11(4):395–410, July 2001.

[2] J. Berdine, P. W. O'Hearn, U. S. Reddy, and H. Thielecke. Linear continuation-passing. *Higher-Order and Symbolic Computation.* To appear.

[3] C. Bruggeman, O. Waddell, and R. K. Dybvig. Representing control in the presence of one-shot continuations. In *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation*, pages 99–107, 1996.

[4] O. Danvy. Formalizing implementation strategies for first-class continuations. In G. Smolka, editor, *Programming Languages and Systems: 9th European Symposium on Programming, ESOP 2000. Proceedings.*, volume 1782 of *Lecture Notes in Computer Science*, pages 88–103. Springer-Verlag, 2000.

[5] M. J. Fischer. Lambda calculus schemata. In *Proceedings of an ACM Conference on Proving Assertions about Programs*, pages 104–109, New York, Jan. 1972. ACM. SIGPLAN Notices, Vol. 7, No. 1 and SIGACT News, No. 14. Reprinted as [6].

[6] M. J. Fischer. Lambda-calculus schemata. *LISP and Symbolic Computation*, 6(3/4):259–288, 1993. Reprint of [5].

[7] J. Laird. A game semantics of linearly used continuations. Personal communication, Apr. 2002.

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

[9] J. Polakow and F. Pfenning. Properties of terms in continuation-passing style in an ordered logical framework. In J. Despeyroux, editor, *Workshop on Logical Frameworks and Meta-Languages (LFM 2000)*, June 2000. `http://www-sop.inria.fr/certilab/LFM00/Proceedings/`.

[10] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *LISP and Symbolic Computation*, 6(3/4):289–360, 1993.

[11] A. Sabry and M. Felleisen. Is continuation passing useful for data-flow analysis? In *Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, pages 1–12. SIGPLAN, ACM Press, 1994.

[12] O. Shivers. *Control-Flow Analysis of Higher-Order Languages.* PhD thesis, Carnegie Mellon University, 1991.