


# A Reflection on Continuation-Composing Style

**Dariusz Biernacki** 

Institute of Computer Science, University of Wrocław, Poland  
<http://ii.uni.wroc.pl/~dabi/>  
dabi@cs.uni.wroc.pl

**Mateusz Pyzik** 

Institute of Computer Science, University Wrocław, Poland  
matp@cs.uni.wroc.pl

**Filip Sieczkowski** 

Institute of Computer Science, University Wrocław, Poland  
efes@cs.uni.wroc.pl

---

## Abstract

We present a study of the continuation-composing style (CCS) that describes the image of the CPS translation of Danvy and Filinski's **shift** and **reset** delimited-control operators. In CCS continuations are composable rather than abortive as in the traditional CPS, and, therefore, the structure of terms is considerably more complex. We show that the CPS translation from Moggi's computational lambda calculus extended with **shift** and **reset** has a right inverse and that the two translations form a reflection i.e., a Galois connection in which the target is isomorphic to a subset of the source (the orders are given by the reduction relations). Furthermore, we use this result to show that Plotkin's call-by-value lambda calculus extended with **shift** and **reset** is isomorphic to the image of the CPS translation. This result, in particular, provides a first direct-style transformation for delimited continuations that is an inverse of the CPS transformation up to syntactic identity.

**2012 ACM Subject Classification** Theory of computation → Control primitives; Theory of computation → Lambda calculus

**Keywords and phrases** delimited control, continuation-passing style, reflection, call-by-value lambda calculus, computational lambda calculus

**Digital Object Identifier** 10.4230/LIPIcs.FSCD.2020.18

**Acknowledgements** We thank the anonymous reviewers and participants of NWPT'19 as well as the anonymous reviewers of FSCD 2020 for helpful comments on the presentation of this work.

## 1 Introduction

In higher-order programming languages based on the  $\lambda$ -calculus, continuation-passing style (CPS) is a program format in which functions accept an additional parameter – a continuation – that represents the entire rest of the computation [17]. In CPS, computations are explicitly sequentialised according to a given evaluation strategy, the intermediate results are named, and there are no nested function calls, i.e., all function calls are tail calls. A notion associated with CPS is the notion of a CPS translation that transforms a term in direct style, i.e., where continuations are not passed around, to the corresponding term in CPS [8, 16]. Such translations have been routinely used both to define continuation semantics of higher-order programs, where object-level constructs are CPS-translated to the meta-level  $\lambda$ -calculus [21, 18], and as a compilation step bridging the gap between higher-order and low-level languages [23, 1].

In the context of compilation, a critical concern is correctness of the CPS translation. In his seminal work [16], Plotkin introduced the call-by-value lambda calculus  $\lambda_v$ , equipped with a reduction  $\rightarrow$  and equality  $=$  theories, along with a CPS translation  $*$  to the call-by-name lambda calculus  $\lambda_n$ , for which he proved equational soundness, i.e.,  $M = N$  in



© Dariusz Biernacki, Mateusz Pyzik, and Filip Sieczkowski;  
licensed under Creative Commons License CC-BY

5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020).

Editor: Zena M. Ariola; Article No. 18; pp. 18:1–18:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the source implies  $M^* = N^*$  in the target, and he showed that the converse does not hold, i.e., the translation is not complete. Completeness has been obtained by Moggi who devised a monad translation, including a CPS translation as a special case, from the computational lambda calculus  $\lambda_c$  (an extension of  $\lambda_v$ ) to the monadic metalanguage  $\lambda_{ml}$  (equipped with an equational theory only) [15]. This result was strengthened by Hatcliff and Danvy [11], who showed that Moggi’s monad translation  $*$  has an inverse translation  $\#$  such that  $M = M^*\#$  in the source and  $N^{\#\#} = N$  in the target, i.e., it is an equational correspondence. Finally, Sabry and Felleisen further improved on Plotkin’s result by devising a CPS translation that forms an equational correspondence between  $\lambda_c$  and  $\lambda_n$  [19].

Whereas all these results concern equality theories, Sabry and Wadler obtained stronger results in which equality is replaced with reduction (viewed as directed code optimisation) [20]. In particular, they presented a CPS translation  $*$  from  $\lambda_c$  to  $\lambda_n$  along with its inverse  $\#$  that form a Galois connection satisfying:

- soundness: if  $M \rightarrow N\#$  in the source then  $M^* \rightarrow N$  in the target;
- completeness: if  $M^* \rightarrow N$  in the target then  $M \rightarrow N\#$  in the source;

This means that evaluation in the source language is equivalent to compiling, evaluating in the target and decompiling. Moreover, this Galois connection is a reflection by satisfying an additional condition: compiling is a left inverse to decompiling,  $M^{\#\#} \equiv M$ , where  $\equiv$  is syntactic identity. Interestingly, Danvy developed a direct-style transformation from  $\lambda_n$  to  $\lambda_v$  that is a left inverse to Plotkin’s CPS translation [2], but he did not consider reduction or equality theories.

A particularly interesting application of CPS is in defining the semantics of control operators, i.e., constructs that access and manipulate the continuation [17, 24, 5]. For abortive control operators such as `call/cc`, which model jumps, the image of the CPS translation is more challenging to characterise than in the pure case. The main reason is that in the pure case one continuation identifier suffices, whereas the abortive control operators may use any of the lexically visible continuation identifiers – continuations can be used out of turn. Sabry and Felleisen [19] considered an extension of Felleisen et al.’s  $\lambda_v\text{-}\mathcal{C}$ -calculus (including `call/cc` and the `abort` operator) [6], and they showed a CPS-translation to  $\lambda_n$  that forms an equational correspondence. A direct-style translation for `call/cc` was also developed by Danvy and Lawall [4]. Their transformation is related to the CPS translation via a Galois connection, induced from the translations and based on the syntactic structure of terms rather than on reduction relations.

In this work, we study the continuation-composing style (CCS), which arises as the image of the CPS translation of Danvy and Filinski’s delimited-control operators `shift` and `reset` [3]. In CCS continuations are composable rather than abortive, which means that not all calls are tail calls and the conditions imposed on where continuation identifiers occur in terms are further relaxed. Continuation composability is central to the expressibility of arbitrary computational effects with continuations [25, 7]. There exist some work devoted to the image of the double CPS translation of `shift` and `reset` in which a meta-continuation is introduced to eliminate nested computations of CCS [3]. Most notably, Kameyama and Hasegawa introduced a direct-style translation which led to a direct-style equational characterisation of the image of the double CPS translation with  $\beta\eta$ -equality (a subset of  $\lambda_n$ ) [12]. However, we are not aware of any published study of the reduction theory of CCS, and the goal of the present work is to fill this vacuum.

To that end, we follow the programme of Sabry and Wadler [20], and we construct a reflection of CCS in two calculi with `shift`, considered as a combinator, and `reset`. We first focus on the  $\lambda_{cS}$ -calculus, which is Moggi’s computational lambda calculus  $\lambda_c$ , extended with

**shift** and **reset** for which we give a CPS translation that eliminates administrative redexes. The image of the translation is  $\lambda_{cS}^*$ , a call-by-value lambda calculus equipped with a set of dedicated reduction rules. We then define a direct-style translation from this calculus back to a subset of  $\lambda_{cS}$  that we call  $\lambda_{cS}^\flat$  (the kernel of  $\lambda_{cS}$ ) and we prove that it is a right inverse to the CPS translation. We then show that the two translations form a reflection with respect to orders given by the respective reduction relations and that the reflection decomposes into an inclusion of  $\lambda_{cS}$  in the kernel  $\lambda_{cS}^\flat$  and an order isomorphism of  $\lambda_{cS}^\flat$  and  $\lambda_{cS}^*$ .

Second, we consider  $\lambda_S$ , a subcalculus of  $\lambda_{cS}$  that is a more traditional calculus of delimited control and that coincides with  $\lambda_v$  extended with **shift** and **reset**. Building on the results for  $\lambda_{cS}$  and restricting the CPS translation to  $\lambda_S$ , we show that  $\lambda_S$  is isomorphic to  $\lambda_S^*$ , its image through the CPS translation and a subcalculus of  $\lambda_{cS}^*$ . A byproduct of this development is a one-pass direct-style translation for delimited-control operators, a long missing continuation of the work by Danvy for pure call-by-value lambda calculus [2], and by Danvy and Lawall for abortive control operators [4]. Such transformations make it possible to automatically map continuation-passing programs to their more concise, but at the same time more challenging to design, direct-style counterparts.

The remainder of this article is structured as follows. In Section 2, we briefly introduce the basic notions related to Galois connections and reflections. In Section 3, we introduce the calculi  $\lambda_{cS}$  and  $\lambda_{cS}^*$  along with the CPS translation from  $\lambda_{cS}$  to  $\lambda_{cS}^*$ . In Section 4, we characterise the image of the CPS translation and we define its right inverse – the direct style translation. In Section 5, we prove that the two translations form a reflection and we identify the kernel of the reflection in  $\lambda_{cS}$ . In Section 6, we show that when restricted to  $\lambda_S$ , the CPS translation has an inverse such that the two transformations form an isomorphism. We conclude in Section 7.

## 2 Galois Connections and Reflections

Below, we recall the essential facts about Galois connections that we use throughout the article. We refer the reader to Sabry and Wadler’s work [20] for more detailed background. We treat each set  $A$  as equipped with a preorder (i.e., reflexive and transitive) relation  $\rightarrow_A$ . In our development, we define these in two ways: either by applying reflexive-transitive closure on a reduction relation  $\rightarrow_A$ , or by truncating a preorder (multi-step reduction) relation  $\rightarrow_X$  of a superset  $X \supseteq A$  ( $\rightarrow_A$  is then an induced preorder). In the following,  $\equiv_A$  denotes the syntactic identity on  $A$ .

In the following, it may be helpful to think about  $A$  and  $B$  as a source and target calculi, respectively, whereas  $f$  and  $g$  can be thought of as compiling and decompiling, respectively. We start with the standard notion of monotonicity, i.e, preservation of reduction by the compiling map.

► **Definition 1** (Monotone function). *A function  $f : A \rightarrow B$  is monotone if, and only if  $\forall x_1, x_2 \in A. x_1 \rightarrow_A x_2 \implies f(x_1) \rightarrow_B f(x_2)$ .*

A Galois connection expresses a form of harmony of compiling and decompiling with respect to reduction relations.

► **Definition 2** (Galois connection). *Monotone functions  $f : A \rightarrow B$  and  $g : B \rightarrow A$  form a Galois connection if, and only if  $a \rightarrow_A g(b) \iff f(a) \rightarrow_B b$ .*

There is an alternative characterisation of a Galois connection.

## 18:4 A Reflection on Continuation-Composing Style

terms	$L, M, N$	$::= V \mid P$
values	$V, W$	$::= x \mid \lambda x. M \mid \mathcal{S}$
nonvalues	$P, Q$	$::= M N \mid \text{let } x = M \text{ in } N \mid \langle M \rangle$
pure contexts	$J, K$	$::= [] \mid K M \mid V K \mid \text{let } x = K \text{ in } M$
$(\beta.v)$	$(\lambda x. M) V$	$\rightarrow M[x := V]$
$(\eta.v)$	$\lambda x. V x$	$\rightarrow V$
$(\beta.\text{let})$	$\text{let } x = V \text{ in } M$	$\rightarrow M[x := V]$
$(\eta.\text{let})$	$\text{let } x = M \text{ in } x$	$\rightarrow M$
$(\text{assoc})$	$\text{let } x = \text{let } y = L \text{ in } M \text{ in } N$	$\rightarrow \text{let } y = L \text{ in let } x = M \text{ in } N$
$(\text{let.1})$	$P N$	$\rightarrow \text{let } x = P \text{ in } x N$
$(\text{let.2})$	$V Q$	$\rightarrow \text{let } y = Q \text{ in } V y$
$(\beta.\mathcal{S})$	$\langle J[\mathcal{S} N] \rangle$	$\rightarrow \langle N (\lambda y. \langle J[y] \rangle) \rangle$
$(\beta.\mathcal{R})$	$\langle V \rangle$	$\rightarrow V$

■ **Figure 1** Direct style calculus  $\lambda_{c\mathcal{S}}$ .

► **Theorem 3** (Equivalent definition of Galois connection). *Monotone functions  $f : A \rightarrow B$  and  $g : B \rightarrow A$  form a Galois connection if, and only if*

- $a \rightarrow_A g(f(a))$  and
- $f(g(b)) \rightarrow_B b$ .

When compiling is a left inverse to decompiling, then we have a reflection.

► **Definition 4** (Reflection). *A Galois connection  $(f : A \rightarrow B, g : B \rightarrow A)$  is a reflection if, and only if  $f(g(b)) \equiv_B b$ .*

In case compiling is also a right inverse to decompiling, we have an isomorphism.

► **Definition 5** (Order isomorphism). *A reflection  $(f : A \rightarrow B, g : B \rightarrow A)$  is an order isomorphism if, and only if  $a \equiv_A g(f(a))$ .*

Every reflection factors into an inclusion and an order isomorphism.

► **Theorem 6** (Reflection decomposition). *Every reflection  $(f : A \rightarrow B, g : B \rightarrow A)$  decomposes into a reflection (called inclusion)  $(g \circ f : A \rightarrow g[B], id_{g[B]} : g[B] \rightarrow A)$  and an order isomorphism  $(f : g[B] \rightarrow B, g : B \rightarrow g[B])$ , where  $g[B] \subseteq A$  has an induced preorder.*

It follows from Theorem 6 that given a reflection  $(f : A \rightarrow B, g : B \rightarrow A)$ , the source calculus has a kernel  $g[B]$  (or equivalently,  $g[f[A]]$ ) that is isomorphic with  $B$ , or that reflects  $B$ . The goal of this work is to identify such a reflection of CCS in call-by-value lambda calculi with delimited continuations.

### 3 Delimited-Control Operators Shift and Reset

We begin with Moggi's calculus of computations,  $\lambda_c$ , extended with shift and reset delimited control operators, which we dub  $\lambda_{c\mathcal{S}}$ . The syntax and semantics are presented in Figure 1. The terms of the calculus are divided into values, which include variables, lambda abstractions and the shift combinator  $\mathcal{S}$ , and computations, which include applications, let-bindings and the reset operator, which serves to delimit the scope of the continuation. Moreover, we introduce the syntactic domain of pure evaluation contexts, which encode a left-to-right call-by-value evaluation strategy and, crucially do not contain the reset operators.

$*$	$:\lambda_{cS} \rightarrow \lambda$	
$M^*$		$= \lambda k.(M : k)$
$V : K$		$= K V^\dagger$
$(P Q) : K$		$= P : (\lambda x.(Q : (\lambda y.xy K)))$
$(P W) : K$		$= P : (\lambda x.x W^\dagger K)$
$(V Q) : K$		$= Q : (\lambda y.V^\dagger y K)$
$(V W) : K$		$= V^\dagger W^\dagger K$
$(\text{let } x = M \text{ in } N) : K$		$= M : (\lambda x.(N : K))$
$\langle M \rangle : K$		$= K (M : (\lambda x.x))$
$x^\dagger$		$= x$
$(\lambda x.M)^\dagger$		$= \lambda x.M^*$
$S^\dagger$		$= \lambda w j.w (\lambda y k.k (j y)) (\lambda x.x)$

■ **Figure 2** Conversion from  $\lambda_{cS}$  to Continuation-Composing Style.

The operational semantics of the calculus is given by a contraction relation, which may be performed within any context, as we consider general reduction rather than evaluation. Nonetheless, we still require the pure fragment of evaluation contexts: these are used to match the shift operator with the enclosing reset by the  $(\beta.S)$  rule. This rule matches a  $S$  operator applied to some term  $N$  in a pure context  $J$  closed by a reset operator, *captures* the latter context (together with the reset), reifies it as a function and passes it as an argument to  $N$ . Note the duplication of the reset operator in the contractum, which is an important characteristic of shift/reset [22].

Except for  $(\beta.S)$  and the simple  $(\beta.R)$  rule, the rules are those of  $\lambda_c$ , including  $\beta$  and  $\eta$  rules for applications and the let-bindings, as well as a rule for association, or hoisting, of let bindings. Note, however, that we do not include any  $\eta$  rules for the control operators, restricting ourselves to the appropriate  $\beta$ -reductions, which leads to a minimal extension of  $\lambda_c$  with delimited control. It can be shown that the resulting calculus is confluent. While most presentations treat  $S$  as an operator with a binder (for a continuation variable) rather than as a combinator, the latter approach is hardly non-standard: in particular, most implementations provide shift as a combinator.

We now turn to the CPS transformation for  $\lambda_{cS}$ , which is presented in Figure 2. Since the CCS calculus is rather complex, the transformation targets syntactic lambda-terms, and we establish the fact that it only produces terms in CCS a posteriori. This translation extends Sabry and Wadler's CPS translation for  $\lambda_c$  [20], which eliminates unnecessary administrative redexes, to handle the shift and reset delimited control operators. Note that, in contrast to some of the classic one-pass CPS translations for shift and reset, including Danvy and Filinski's [3], the translation does not reduce matching shift-reset pairs at transformation time. Note that without this more conservative approach to source-language redexes we could not hope for establishing the desired reflection.

## 4 Back to Direct Style

Having defined the CPS translation for the extended computational calculus, we now turn to precisely identifying its image. To this end, we introduce a new calculus,  $\lambda_{cS}^*$ , presented in Figure 3. The syntax is given as a mildly context-sensitive grammar in the style of *literal movement grammars* [10]. In this case, context-sensitivity amounts to annotating both term

roots	$R$	$::= \lambda k . M_k$
terms $_{\Delta}$	$M, N$	$::= K_{\Delta} V \mid V W K_{\Delta} \mid K_{\Delta} M_{\bullet}$
values	$V, W$	$::= x \mid \lambda x . R \mid S$
shift	$S$	$::= \lambda w j . w (\lambda y k . k (j y)) (\lambda x . x)$
continuations $_{\Delta}$	$J, K$	$::= (\Delta=k) k \mid (\Delta=\bullet) \lambda x . x \mid \lambda x . M_{\Delta}$
$(\beta.v)$	$(\lambda x k . M_k) V K_{\Delta}$	$\rightarrow M_k[x := V][k := K_{\Delta}]$
$(\eta.v)$	$\lambda x k . V x k$	$\rightarrow V$
$(\beta.let)$	$(\lambda x . M_{\Delta}) V$	$\rightarrow M_{\Delta}[x := V]$
$(\eta.let)$	$\lambda x . K_{\Delta} x$	$\rightarrow K_{\Delta}$
$(\beta.S)$	$S W J_{\bullet}$	$\rightarrow W (\lambda y k . k (J_{\bullet} y)) (\lambda x . x)$
$(\beta.R)$	$(\lambda x . x) V$	$\rightarrow V$

■ **Figure 3** Continuation-composing style calculus  $\lambda_{cS}^*$ .

and continuation nonterminals with  $\Delta$ , which ranges over the set of variables extended with  $\bullet$ , and limiting certain productions to particular annotations. This serves to distinguish the “tail-recursive” parts of the term, where there is a current continuation that needs to be used, from the “returning” calls, where there is no access to the current continuation (and thus the only trivial continuation is the identity). Throughout the following, we use  $\Delta \vdash_C^N M$  to mean that a term  $M$  is derived as a member of syntactic class  $N$  of calculus  $C$  under assumptions  $\Delta$  of the shape appropriate for the given calculus and non-terminal combination; in the case of standard context-free grammars, this assumption context is always empty.

For the semantics of our calculus, we follow the methodology of Sabry and Wadler [20], extending their  $\lambda_{cps}$  calculus with reductions that notionally match our control operators. Note that while this calculus can be considered a subsystem of the lambda-calculus, its reductions take much larger steps, and thus the system is not closed under general  $\lambda_v$  reductions. We begin by establishing that the image of the CPS translation defined in previous section is indeed contained within  $\lambda_{cS}^*$ .

► **Lemma 7** (Characterisation of CCS). *For all  $M \in \lambda_{cS}$ ,  $M^* \in \lambda_{cS}^*$ .*

**Proof.** We prove the following propositions by mutual structural induction on the term:

- $\vdash_{\lambda_{cS}}^M M \implies \vdash_{\lambda_{cS}^*}^R M^*$ ,
- $\vdash_{\lambda_{cS}}^M M \wedge \Delta \vdash_{\lambda_{cS}^*}^K K \implies \Delta \vdash_{\lambda_{cS}^*}^M (M : K)$ ,
- $\vdash_{\lambda_{cS}}^V V \implies \Delta \vdash_{\lambda_{cS}^*}^V V^\dagger$ . ◀

#### 4.1 Direct-Style Transformation

With the calculus  $\lambda_{cS}^*$  defined, we now turn to a translation to direct style. The target of such translation is  $\lambda_{cS}$ , the computational calculus with shift and reset, and the translation is defined in Figure 4, with the definition proceeding inductively on the structure of terms of  $\lambda_{cS}^*$ . We can now show that the CPS and DS translations form a retraction pair with respect to syntactic equality (as usual, up to implicit  $\alpha$ -equivalence) in  $\lambda_{cS}^*$ . We take  $I_{\Delta}$  to denote the trivial continuation for  $\Delta$ , i.e.,  $I_k = k$  and  $I_{\bullet} = \lambda x . x$ .

► **Theorem 8** (Right inverse of  $*$ ). *For all  $R \in \lambda_{cS}^*$ ,  $R^{\#\#} \equiv R$ . Also, the following equalities hold:  $M_{\Delta}^{\#} : I_{\Delta} \equiv M_{\Delta}$ ,  $V^{\#\dagger} \equiv V$ ,  $K_{\Delta}^b[M] : I_{\Delta} \equiv M : K_{\Delta}$ .*

**Proof.** By mutual structural induction. ◀

$\# : \lambda_{cS}^* \rightarrow \lambda_{cS}$	
$(\lambda k . M_k)^\#$	$= M_k^\#$
$(K_\Delta V)^\#$	$= K_\Delta^b[V^\natural]$
$(V W K_\Delta)^\#$	$= K_\Delta^b[V^\natural W^\natural]$
$(K_\Delta M_\bullet)^\#$	$= K_\Delta^b[\langle M_\bullet^\# \rangle]$
$x^\natural$	$= x$
$(\lambda x . R)^\natural$	$= \lambda x . R^\#$
$(\lambda w j . w (\lambda y k . k (j y)) (\lambda x . x))^\natural$	$= \mathcal{S}$
$k^b$	$= []$
$(\lambda x . x)^b$	$= []$
$(\lambda x . N_\Delta)^b$	$= \text{let } x = [] \text{ in } N_\Delta^\#$

■ **Figure 4** Back to Direct Style from  $\lambda_{cS}^*$ .

terms	$M, N ::= K[V] \mid K[P]$
values	$V, W ::= x \mid \lambda x . M \mid \mathcal{S}$
nonvalues	$P, Q ::= V W \mid \langle M \rangle$
pure contexts	$J, K ::= [] \mid \text{let } x = [] \text{ in } M$
$(\beta.v)$	$K[(\lambda x . M) V] \rightarrow M[x := V] : K \quad K \text{ maximal}$
$(\eta.v)$	$\lambda x . V x \rightarrow V$
$(\beta.let)$	$\text{let } x = V \text{ in } M \rightarrow M[x := V]$
$(\eta.let)$	$\text{let } x = [] \text{ in } K[x] \rightarrow K$
$(\beta.\mathcal{S})$	$\langle J[\mathcal{S} W] \rangle \rightarrow \langle W (\lambda y . \langle J[y] \rangle) \rangle$
$(\beta.\mathcal{R})$	$\langle V \rangle \rightarrow V$
$V : K$	$= K[V]$
$P : K$	$= K[P]$
$(\text{let } x = V \text{ in } M) : K$	$= \text{let } x = V \text{ in } (M : K)$
$(\text{let } x = P \text{ in } M) : K$	$= \text{let } x = P \text{ in } (M : K)$

■ **Figure 5** The kernel direct style calculus  $\lambda_{cS}^\flat$ .

This result establishes that  $\lambda_{cS}^*$  does not overestimate the set of valid CCS terms, as any root in  $\lambda_{cS}^*$  can be obtained by the CPS transformation from its own translation to  $\lambda_{cS}$ . However, not all terms of  $\lambda_{cS}$  can be obtained as the result of the direct style translation. Thus, we define yet another calculus,  $\lambda_{cS}^\flat$ , which characterises the *kernel* of  $\lambda_{cS}$ . The definition presented in Figure 5 again follows and extends Sabry and Wadler’s take on a refined calculus (this time extending their  $\lambda_{c**}$ ); the major difference with respect to  $\lambda_{cS}$  is the fact that all the let-bindings are hoisted, i.e., normalised with respect to associativity rule. The reduction rules need to preserve this fact, which again leads to larger reduction steps: this time, when reducing an application, we may need to reassociate arbitrarily many let-bindings. We finish this section by establishing that the image of our direct style translation falls within  $\lambda_{cS}^\flat$ .

► **Lemma 9** (Characterisation of kernel DS). *For all  $M \in \lambda_{cS}^*$ ,  $M^\# \in \lambda_{cS}^\flat$ .*

**Proof.** We prove the following propositions by mutual structural induction on the term:

- $\vdash_{\lambda_{cS}^*}^R R \implies \vdash_{\lambda_{cS}^\flat}^M R^\#$ ,
- $\Delta \vdash_{\lambda_{cS}^*}^M M \implies \vdash_{\lambda_{cS}^\flat}^M M^\#$ ,



- $\vdash_{\lambda_{cS}^*}^V V \implies \vdash_{\lambda_{cS}^\flat}^V V^\dagger$ ,
- $\Delta \vdash_{\lambda_{cS}^*}^K K \implies \vdash_{\lambda_{cS}^\flat}^K K^\flat$ . ◀

## 5 Reflection: Computational $\lambda$ -Calculus with Shift and Reset

Having introduced the three main calculi involved in the reflection and established a syntactic inverse in one direction (in Theorem 8), we now turn to establishing a Galois connection between  $\lambda_{cS}$  and  $\lambda_{cS}^*$ . By Theorem 6, such a connection will decompose into an isomorphism and a reflection: in the following we establish that  $\lambda_{cS}^\flat$  is such a factorisation.

### 5.1 Monotonicity

In order for our CPS and DS transformations to form a Galois connection, we must first establish that they are monotone maps, i.e., that they preserve the *order* given by reflexive-transitive closure of the reduction relation of, respectively,  $\lambda_{cS}$  and  $\lambda_{cS}^*$ . Since for some intermediate results we require zero or one reduction steps, we also introduce  $\rightarrow^?$  as a reflexive closure of the relation  $\rightarrow$ . We begin by establishing that any pure evaluation context  $J$  of  $\lambda_{cS}$  can be matched by a continuation of  $\lambda_{cS}^*$ .

► **Lemma 10** (Existence of a continuation for each context). *For all  $J, \Delta$  and  $K_\Delta$ , exists  $\hat{J}_\Delta$  such that for all  $M$ ,  $J[M] : K_\Delta \equiv M : \hat{J}_\Delta$ .*

**Proof.** By structural induction on  $J$ . ◀

Next, we show that any reduction of  $\lambda_{cS}^*$  continuations extends to the colon translations of a common  $\lambda_{cS}$  term.

► **Lemma 11** (Single-step reduction preservation by  $:$  in the second argument). *For any  $\lambda_{cS}$  term  $M$  and  $\lambda_{cS}^*$  continuations  $J_\Delta$  and  $K_\Delta$  such that  $J_\Delta \rightarrow K_\Delta$  we have  $M : J_\Delta \rightarrow^? M : K_\Delta$ .*

**Proof.** By structural induction on  $M$ . ◀

Finally, we can show that the CPS translation preserves single-step reductions, possibly without making a transition in  $\lambda_{cS}^*$ . Monotonicity of CPS follows as a simple corollary.

► **Lemma 12** (Single-step reduction preservation by  $;$ ,  $*$  and  $\dagger$ ). *The following implications hold:*

- $M \rightarrow N \implies \forall K. M : K_\Delta \rightarrow^? N : K_\Delta$ ,
- $M \rightarrow N \implies M^* \rightarrow^? N^*$ ,
- $V \rightarrow W \implies V^\dagger \rightarrow^? W^\dagger$

**Proof.** We prove the statements by mutual induction on the structure of the term, and invert the reduction relation as necessary. Preservation in the second argument is used for some congruences and ( $\eta$ .*let*). Base cases (*let*.1), (*let*.2), (*assoc*) follow by definition. The existence of a continuation is used for ( $\beta$ . $\mathcal{S}$ ).

We show the case for the ( $\beta$ . $\mathcal{S}$ ) reduction as an interesting example. We have the following reduction:

$$\langle J[\mathcal{S} W] \rangle \rightarrow \langle W (\lambda y. \langle J[y] \rangle) \rangle,$$

and need to prove that  $\langle J[\mathcal{S} W] \rangle : K_\Delta \rightarrow^? \langle W (\lambda y. \langle J[y] \rangle) \rangle : K_\Delta$ .



We proceed as follows:

$$\begin{array}{l}
\langle J[S W] \rangle : K_{\Delta} \\
\equiv_{\text{def.}} K_{\Delta}(J[S W] : (\lambda x . x)) \\
\equiv_{\text{existence of cont.}} K_{\Delta}(S W : J_{\bullet}) \\
\equiv_{\text{def.}} K_{\Delta}(S W^{\dagger} J_{\bullet}) \\
\rightarrow_{(\beta . S)} K_{\Delta}(W^{\dagger}(\lambda y k . k(J_{\bullet} y))(\lambda x . x)) \\
\equiv_{\text{def.}} K_{\Delta}(W^{\dagger}(\lambda y k . k(y : J_{\bullet}))(\lambda x . x)) \\
\equiv_{\text{existence of cont.}} K_{\Delta}(W^{\dagger}(\lambda y k . k(J[y] : (\lambda x . x)))(\lambda x . x)) \\
\equiv_{\text{def.}} K_{\Delta}((W(\lambda y . \langle J[y] \rangle)) : (\lambda x . x)) \\
\equiv_{\text{def.}} \langle W(\lambda y . \langle J[y] \rangle) \rangle : K_{\Delta}
\end{array}$$

As another examples, consider the sample congruence case, where we have:

$$\frac{V \rightarrow W}{P V \rightarrow P W},$$

and need to show that  $P V : K_{\Delta} \rightarrow^? P W : K_{\Delta}$ .

We proceed as follows:

$$\begin{array}{l}
\implies \text{ind. hyp.} \\
\implies \text{congruence} \\
\implies \text{second arg. preservation} \\
\equiv_{\text{def.}}
\end{array}
\begin{array}{l}
V \rightarrow W \\
V^{\dagger} \rightarrow W^{\dagger} \\
\lambda x . x V^{\dagger} K_{\Delta} \rightarrow \lambda x . x W^{\dagger} K_{\Delta} \\
P : (\lambda x . x V^{\dagger} K_{\Delta}) \rightarrow P : (\lambda x . x W^{\dagger} K_{\Delta}) \\
(P V) : K_{\Delta} \rightarrow (P W) : K_{\Delta} \quad \blacktriangleleft
\end{array}$$

► **Corollary 13** (Monotonicity of  $*$ ). *For all  $M_0, M_1 \in \lambda_{cS}$ ,  $M_0 \rightarrow M_1$  implies  $M_0^* \rightarrow M_1^*$ .*

Monotonicity of the direct-style transformation is simpler to prove: we show that all component parts preserve single-step reductions in  $\lambda_{cS}^{\circ}$  (which themselves are possibly multi-step reduction sequences in  $\lambda_{cS}$ ), and obtain monotonicity as a simple corollary.

► **Lemma 14** (Single-step reduction preservation by  $\#$ ,  $\sharp$ ,  $\natural$  and  $\flat$ ). *The following implications hold:*

- $R_0 \rightarrow R_1' \implies R_0^{\#} \rightarrow R_1^{\#}$ ,
- $M_k \rightarrow N_k \implies M_k^{\sharp} \rightarrow N_k^{\sharp}$ ,
- $M_{\bullet} \rightarrow N_{\bullet} \implies \langle M_{\bullet}^{\sharp} \rangle \rightarrow \langle N_{\bullet}^{\sharp} \rangle$ ,
- $V \rightarrow W \implies V^{\natural} \rightarrow W^{\natural}$ ,
- $J_{\Delta} \rightarrow K_{\Delta} \implies \forall M . J_{\Delta}^{\flat}[M] \rightarrow K_{\Delta}^{\flat}[M]$ .

**Proof.** Mutual structural induction on the first term and then each case by inversion on single-step reduction. ◀

► **Corollary 15** (Monotonicity of  $\#$ ). *For all  $R_0, R_1 \in \lambda_{cS}^*$ ,  $R_0 \rightarrow R_1$  implies  $R_0^{\#} \rightarrow R_1^{\#}$ .*

## 5.2 Reflection theorem

Recall from Section 2 that in order to establish that  $*$  and  $\#$  form a Galois connection, it is enough to show that both compositions are extensive with respect to the appropriate

## 18:10 A Reflection on Continuation-Composing Style

reduction orderings, i.e.,  $M \rightarrow M^{*\#}$  and  $R \rightarrow R^{\#\#}$ , respectively. Since in Theorem 8 we have shown that  $R \equiv R^{\#\#}$ , the latter ordering holds trivially. In this section we establish the remaining property.

► **Lemma 16** (Generalised associativity). *The following reduction holds:*

$$\blacksquare K_{\Delta}^{\flat}[\text{let } x = L \text{ in } N] \rightarrow^? \text{let } x = L \text{ in } K_{\Delta}^{\flat}[N].$$

**Proof.** By cases on  $K_{\Delta}$ . ◀

► **Lemma 17** (Left near inverse of  $:$  and  $\dagger$ ). *The following reductions hold:*

$$\blacksquare K_{\Delta}^{\flat}[M] \rightarrow (M : K_{\Delta})^{\sharp},$$

$$\blacksquare V \rightarrow V^{\dagger\sharp}.$$

**Proof.** By mutual structural induction. The generalised associativity is used in several cases, it conveniently wraps potential uses of (*let.assoc*) rule, as presented in the following example cases for let-binders and **reset**.

$$\begin{array}{l} \rightarrow_{\text{gen. assoc.}}^? \\ \twoheadrightarrow_{\text{ind. hyp.}} \\ \equiv_{\text{def.}} \\ \twoheadrightarrow_{\text{ind. hyp.}} \end{array} \quad \begin{array}{l} K_{\Delta}^{\flat}[\text{let } x = L \text{ in } N] \\ \text{let } x = L \text{ in } K_{\Delta}^{\flat}[N] \\ \text{let } x = L \text{ in } (N : K_{\Delta})^{\sharp} \\ (\lambda x . (N : K_{\Delta}))^{\flat}[L] \\ ((\text{let } x = L \text{ in } N) : K_{\Delta})^{\sharp} \end{array}$$

$$\begin{array}{l} \equiv_{\text{def.}} \\ \twoheadrightarrow_{\text{ind. hyp.}} \\ \equiv_{\text{def.}} \\ \equiv_{\text{def.}} \end{array} \quad \begin{array}{l} K_{\Delta}^{\flat}[\langle M \rangle] \\ K_{\Delta}^{\flat}[\langle I_{\bullet}^{\flat}[M] \rangle] \\ K_{\Delta}^{\flat}[\langle (M : I_{\bullet})^{\sharp} \rangle] \\ (K_{\Delta}(M : I_{\bullet}))^{\sharp} \\ (\langle M \rangle : K_{\Delta})^{\sharp} \end{array} \quad \blacktriangleleft$$

► **Theorem 18** (Left near inverse of  $*$ ). *For all  $M \in \lambda_{cS}$ ,  $M \rightarrow M^{*\#}$ .*

**Proof.** Follows from the left near inverse for the  $:$  transformation. ◀

► **Corollary 19** (Reflection). *Transformations  $*$  and  $\#$  form a reflection.*

### 5.3 Reflection decomposition

By Theorem 6, any reflection decomposes into an order isomorphism and an inclusion. In our case, this means that the reflection  $(*, \#)$  has a kernel that is isomorphic to  $\lambda_{cS}^*$ . This is our calculus  $\lambda_{cS}^{\flat}$  – although we still need to establish that it is in fact isomorphic to  $\lambda_{cS}^*$ . To this end, we first calculate the CPS translation as specialised to  $\lambda_{cS}^{\flat}$  (i.e., as a composition of inclusion of  $\lambda_{cS}^{\flat}$  in  $\lambda_{cS}$  and  $*$ ), which we dub  $\star$ ; this transformation is presented in Figure 6. We can then establish that  $\star$  and  $\#$  compose to identity, which, together with one-to-one matching of reductions established in Lemma 14 establishes the isomorphism.

$$\begin{array}{lcl}
\star : \lambda_{cS}^\triangleright \rightarrow \lambda_{cS}^* & & \\
M^\star & = & \lambda k . M_k^\circ \\
(K[V]_\Delta)^\circ & = & K_\Delta^\ddagger V^\dagger \\
(K[VW]_\Delta)^\circ & = & V^\dagger W^\dagger K_\Delta^\ddagger \\
(K[\langle M \rangle]_\Delta)^\circ & = & K_\Delta^\ddagger M_\bullet^\circ \\
x^\dagger & = & x \\
(\lambda x . M)^\dagger & = & \lambda x . M^\star \\
S^\dagger & = & \lambda w j . w (\lambda y k . k (j y)) (\lambda x . x) \\
[]_k^\ddagger & = & k \\
[]_\bullet^\ddagger & = & \lambda x . x \\
(\text{let } x = [] \text{ in } N)_\Delta^\ddagger & = & \lambda x . N_\Delta^\circ
\end{array}$$

■ **Figure 6** Order isomorphism from  $\lambda_{cS}^\triangleright$  to  $\lambda_{cS}^*$ .

$$\begin{array}{lcl}
\triangleright : \lambda_{cS} \rightarrow \lambda_{cS}^\triangleright & & \\
M^\triangleright & = & M : [] \\
V : K & = & K[V^\dagger] \\
(PQ) : K & = & P : (\text{let } x = [] \text{ in } (Q : (\text{let } y = [] \text{ in } K[x y]))) \\
(PW) : K & = & P : (\text{let } x = [] \text{ in } K[x W^\dagger]) \\
(VQ) : K & = & Q : (\text{let } y = [] \text{ in } K[V^\dagger y]) \\
(VW) : K & = & K[V^\dagger W^\dagger] \\
(\text{let } x = M \text{ in } N) : K & = & M : (\text{let } x = [] \text{ in } (N : K)) \\
\langle M \rangle : K & = & K[\langle M^\triangleright \rangle] \\
x^\dagger & = & x \\
(\lambda x . M)^\dagger & = & \lambda x . M^\triangleright \\
S^\dagger & = & S
\end{array}$$

■ **Figure 7** Inclusion in  $\lambda_{cS}$  of  $\lambda_{cS}^\triangleright$ .

► **Lemma 20** (Left inverse of  $\star$ ). *For all  $M \in \lambda_{cS}^\triangleright$  we have  $M \equiv M^{\star\#}$ .*

**Proof.** By mutual induction on the structure of terms, including analogous statements for the auxiliary transformations. ◀

Having established that  $\lambda_{cS}^\triangleright$  is isomorphic to  $\lambda_{cS}^*$ , we obtain an inclusion between  $\lambda_{cS}^\triangleright$  and the main calculus,  $\lambda_{cS}$ . Thus, to conclude this section we present the one-pass transformation  $\triangleright : \lambda_{cS} \rightarrow \lambda_{cS}^\triangleright$ , which is computed from the composition of  $\star$  and  $\#$ . This transformation, presented in Figure 7, forms a final reflection for these calculi, together with identity:  $(\triangleright, \text{id}_{\lambda_{cS}^\triangleright})$  is a reflection between  $\lambda_{cS}$  and  $\lambda_{cS}^\triangleright$ .

## 6 Isomorphism: $\lambda_v$ -Calculus with Shift and Reset

While the results we have obtained thus far provide some fundamental insight into the structure and reductions of computations in continuation composing style, the computational calculus  $\lambda_{cS}$  we took as our source language differs somewhat from calculi with shift and reset that are most commonly studied. Thus, in this section we study  $\lambda_S$ , the call-by-value lambda calculus extended with delimited control operators, and apply the results we have obtained thus far to this restricted setting.

## 18:12 A Reflection on Continuation-Composing Style

terms	$M, N ::= V \mid P$
values	$V, W ::= x \mid \lambda x. M \mid \mathcal{S}$
nonvalues	$P, Q ::= M N \mid \langle M \rangle$
pure contexts	$J, K ::= [] \mid K M \mid V K$
$(\beta.v)$	$(\lambda x. M) V \rightarrow M[x := V]$
$(\eta.v)$	$\lambda x. V x \rightarrow V$
$(\beta.\mathcal{S})$	$\langle J[\mathcal{S} N] \rangle \rightarrow \langle N(\lambda y. \langle J[y] \rangle) \rangle$
$(\beta.\mathcal{R})$	$\langle V \rangle \rightarrow V$

■ **Figure 8** Subcalculus  $\lambda_{\mathcal{S}}$ :  $\lambda_{c\mathcal{S}}$  without let.

roots	$R ::= V_\varepsilon \mid M_\varepsilon$
terms $_\Sigma$	$M, N ::= (\Sigma = \Sigma_1 \Sigma_2) K_{\Sigma_1} [P_{\Sigma_2}]$
values $_\Sigma$	$V, W ::= (\Sigma = x) x \mid (\Sigma = \varepsilon) x \mid (\Sigma = \varepsilon) \lambda x. R \mid (\Sigma = \varepsilon) \mathcal{S}$
nonvalues $_\Sigma$	$P, Q ::= (\Sigma = \Sigma_1 \Sigma_2) V_{\Sigma_1} W_{\Sigma_2} \mid (\Sigma = \varepsilon) \langle R \rangle$
pure contexts $_\Sigma$	$J, K ::= (\Sigma = \varepsilon) [] \mid \text{let } x = [] \text{ in } M_{\Sigma x}$
$(\beta.v)$	$K_\Sigma[(\lambda x. R) V_\varepsilon] \rightarrow R[x := V_\varepsilon] : K_\Sigma \quad K_\Sigma \text{ maximal}$
$(\eta.v)$	$\lambda x. V_\varepsilon x \rightarrow V_\varepsilon$
$(\beta.\mathcal{S})$	$\langle J_\varepsilon[\mathcal{S} W_\varepsilon] \rangle \rightarrow \langle W_\varepsilon(\lambda y. \langle J_\varepsilon; y \rangle) \rangle$
$(\beta.\mathcal{R})$	$K_\Sigma[(V_\varepsilon)] \rightarrow K_\Sigma; V_\varepsilon \quad K_\Sigma \text{ maximal}$
$[]; V_\varepsilon$	$= V_\varepsilon$
$(\text{let } x = [] \text{ in } M_{\Sigma x}); V_\varepsilon$	$= M_{\Sigma x}[x := V_\varepsilon]$
$V_\varepsilon : K_\Sigma$	$= K_\Sigma; V_\varepsilon$
$P_{\Sigma_2} : K_{\Sigma_1}$	$= K_{\Sigma_1} [P_{\Sigma_2}]$
$(\text{let } x = P_{\Sigma_2} \text{ in } M_{\Sigma x}) : K_{\Sigma_1}$	$= \text{let } x = P_{\Sigma_2} \text{ in } (M_{\Sigma x} : K_{\Sigma_1})$

■ **Figure 9** Subcalculus  $\lambda_{\mathcal{S}}^\triangleright$ : the image of  $\lambda_{\mathcal{S}}$  via  $\triangleright$ .

The syntax and semantics of  $\lambda_{\mathcal{S}}$  are presented in Figure 8. It is clear that this calculus embeds in  $\lambda_{c\mathcal{S}}$ . The reduction relation is defined via contraction relation and is a strict subrelation of the induced reduction relation (e.g.,  $(\lambda x. x)M \rightarrow M$  in  $\lambda_{c\mathcal{S}}$ , but not in  $\lambda_{\mathcal{S}}$ ). The reduction relations for the CCS and kernel calculi, presented later on in this section, are images of this restricted reduction relation. It is worth noting that it would also be sound to use the induced relations instead: all transformations, including all-new  $\triangleleft$ , are designed to be monotone with respect to the original, wider reduction relations. However, we do not pursue the task of characterising the induced relations in this work, and therefore stick to the restricted relations.

We begin by considering the image of  $\lambda_{\mathcal{S}}$  under the reflection defined in previous section: this is the calculus  $\lambda_{\mathcal{S}}^\triangleright$ , defined in Figure 9. We establish that it contains the image of  $\lambda_{\mathcal{S}}$  under  $\triangleright$ , while deferring the other inclusion till later.

► **Lemma 21** (Restriction of  $\triangleright$ ). *For all  $M \in \lambda_{\mathcal{S}}$ ,  $M^\triangleright \in \lambda_{\mathcal{S}}^\triangleright$ .*

With  $\lambda_{c\mathcal{S}}^\triangleright$  defined, we need to define an inverse transformation,  $\triangleleft$ . However, the reduction relation on  $\lambda_{c\mathcal{S}}$  – induced by its super-calculi – is cumbersome to work with. Thus, we define this transformation for the entire calculus  $\lambda_{c\mathcal{S}}$ , and establish its properties on its particular sub-calculi post hoc. First, notice that the shape of  $\lambda_{\mathcal{S}}^\triangleright$  is more constrained than

$$\begin{array}{l}
\triangleleft : \lambda_{cS} \rightarrow \lambda_{cS} \\
(M N)^\triangleleft = M^\triangleleft N^\triangleleft \\
(\text{let } x = M \text{ in } N)^\triangleleft = N^\triangleleft[x := M^\triangleleft] \quad (\text{if } N \equiv K[x] \text{ for some } K \text{ and } x \notin \text{FV}(K)) \\
(\text{let } x = M \text{ in } N)^\triangleleft = \text{let } x = M^\triangleleft \text{ in } N^\triangleleft \quad (\text{otherwise}) \\
\langle M \rangle^\triangleleft = \langle M^\triangleleft \rangle \\
x^\triangleleft = x \\
(\lambda x . M)^\triangleleft = \lambda x . M^\triangleleft \\
S^\triangleleft = S
\end{array}$$

■ **Figure 10** Deletion of inessential let-expressions.

the general calculus  $\lambda_S$ , which is mostly due to lifting some subcomputations as pure, linear let-expressions. Thus, the idea behind  $\triangleleft$  is to *inline* these (and only these) let-expressions. The transformation is presented in Figure 10.

Additionally, we lift  $\triangleleft$  to pure contexts of  $\lambda_{cS}$  as an auxiliary construct in the following. It is easy to check that this makes  $\triangleleft$  distribute over plugging of terms into contexts:

$$K^\triangleleft[M^\triangleleft] \equiv (K[M])^\triangleleft.$$

We can now show that the transformation is monotone with respect to the  $\lambda_{cS}$  reductions.

► **Theorem 22** (Monotonicity of  $\triangleleft$ ). *For all  $M, N \in \lambda_{cS}$ ,  $M \rightarrow N$  implies  $N^\triangleleft \rightarrow N^\triangleleft$ .*

**Proof.** Induction on reflexive-transitive closure of contraction relation. To prove single-step version, apply structural induction on the left-hand-side term and then inversion on contraction relation. The only interesting case is the contraction of **shift**: given  $\langle J[\mathcal{S} M] \rangle \rightarrow \langle M(\lambda y . \langle J[y] \rangle) \rangle$ , show  $\langle J[\mathcal{S} M] \rangle^\triangleleft \rightarrow \langle M(\lambda y . \langle J[y] \rangle) \rangle^\triangleleft$ . Notice that we have  $\langle J[\mathcal{S} M] \rangle^\triangleleft \equiv \langle J^\triangleleft[\mathcal{S} M^\triangleleft] \rangle \rightarrow \langle M^\triangleleft(\lambda y . \langle J^\triangleleft[y] \rangle) \rangle \equiv \langle M(\lambda y . \langle J[y] \rangle) \rangle^\triangleleft$ . Crucially, transformation preserves purity of contexts. ◀

Now we can ensure that inlining the administrative let-expressions in  $\lambda_S^\triangleleft$  produces terms of  $\lambda_S$ , and thus that  $\triangleright$  restricted to  $\lambda_S$  is a section.

► **Lemma 23** (Restriction of  $\triangleleft$ ). *For all  $R \in \lambda_S^\triangleleft$ ,  $R^\triangleleft \in \lambda_S$ .*

► **Theorem 24** (Left inverse of  $\triangleright$ ). *For all  $M \in \lambda_S$ ,  $M \equiv M^{\triangleright\triangleleft}$ . Also, the following identities hold:  $V^{\triangleright\triangleleft} \equiv V$ ,  $(M : K)^\triangleleft \equiv K^\triangleleft[M]$ .*

**Proof.** Mutual structural induction on  $M, V$  and  $M$ , respectively. ◀

To show that  $\triangleright$  is also a retraction, we need a tool that can apply a substitution of variables for  $\lambda_S$  terms to a  $\lambda_S^\triangleleft$  term.

► **Definition 25** (Iterated flattened let-expressions). *Let  $P, Q$  refer to the grammar of  $\lambda_S$ , all other names refer to the grammar of  $\lambda_S^\triangleleft$ .*

$$\begin{array}{l}
\varepsilon \triangleright R = R \\
[x := P] \triangleright x_x = P^\triangleright \\
[x := P, y := Q] \triangleright M_{\bar{x}y} = \overline{[x := P] \triangleright (Q : (\text{let } y = [] \text{ in } M_{\bar{x}y}))}
\end{array}$$

We can now establish that  $\triangleright$  is a retraction and, as a consequence, that  $\lambda_S$  and  $\lambda_S^\triangleleft$  are isomorphic.

roots	$R$	$::= \lambda k . T_k$
trunks $_{\Delta}$	$T$	$::= I_{\Delta} V_{\varepsilon} \mid M_{\Delta, \varepsilon}$
terms $_{\Delta, \Sigma}$	$M, N$	$::= (\Sigma = \Sigma_1 \Sigma_2 \Sigma_3) V_{\Sigma_2} W_{\Sigma_3} K_{\Delta, \Sigma_1} \mid K_{\Delta, \Sigma} T_{\varepsilon}$
values $_{\Sigma}$	$V, W$	$::= (\Sigma = x) x \mid (\Sigma = \varepsilon) x \mid (\Sigma = \varepsilon) \lambda x . R \mid (\Sigma = \varepsilon) S$
shift	$S$	$::= \lambda w j . w (\lambda y k . k (j y)) (\lambda x . x)$
trivial continuations $_{\Delta}$	$I$	$::= (\Delta = k) k \mid (\Delta = \bullet) \lambda x . x$
continuations $_{\Delta, \Sigma}$	$J, K$	$::= (\Sigma = \varepsilon) I_{\Delta} \mid \lambda x . M_{\Delta, \Sigma x}$
$(\beta.v)$	$(\lambda x k . T_k) V_{\varepsilon} K_{\Delta, \Sigma}$	$\rightarrow T_k[x := V_{\varepsilon}] : K_{\Delta, \Sigma}$
$(\eta.v)$	$\lambda x k . V_{\varepsilon} x k$	$\rightarrow V_{\varepsilon}$
$(\beta.S)$	$S W_{\varepsilon} J_{\bullet, \varepsilon}$	$\rightarrow W_{\varepsilon} (\lambda y k . k (J_{\bullet, \varepsilon}; y)) (\lambda x . x)$
$(\beta.R)$	$K_{\Delta, \Sigma} ((\lambda x . x) V)$	$\rightarrow K_{\Delta, \Sigma}; V$
$I_{\Delta}; V_{\varepsilon}$		$= I_{\Delta} V_{\varepsilon}$
$(\lambda x . M_{\Delta, \Sigma x}); V_{\varepsilon}$		$= M_{\Delta, \Sigma x}[x := V_{\varepsilon}]$
$(k V_{\varepsilon}) : K_{\Delta, \Sigma}$		$= K_{\Delta, \Sigma}; V_{\varepsilon}$
$M_{k, \varepsilon} : K_{\Delta, \Sigma}$		$= M_{k, \varepsilon}[k := K_{\Delta, \Sigma}]$

■ **Figure 11** Subcalculus  $\lambda_S^*$ : the CPS image of  $\lambda_S$ .

► **Theorem 26** (Right inverse of  $\triangleright$ ). *The following identities hold:*

- $R^{\triangleleft} \equiv R$ ,
- $(V_{\bar{x}}^{\triangleleft}[x := \bar{P}])^{\triangleright} \equiv [x := \bar{P}] \triangleright V_{\bar{x}}$ ,
- $(M_{\bar{x}}^{\triangleleft}[x := \bar{P}])^{\triangleright} \equiv [x := \bar{P}] \triangleright M_{\bar{x}}$ .

**Proof.** The proof proceeds by mutual structural induction on  $R$ ,  $V$  and  $M$ , respectively. ◀

► **Corollary 27** (Isomorphism of  $\lambda_S$  and  $\lambda_S^{\triangleright}$ ). *Transformations  $\triangleright : \lambda_S \rightarrow \lambda_S^{\triangleright}$  and  $\triangleleft : \lambda_S^{\triangleright} \rightarrow \lambda_S$  form an isomorphism.*

Although we have shown that  $\lambda_S$  is isomorphic to its image under  $\triangleright$ , both these calculi are in direct style. However, as any image of  $\triangleright$  (and thus of its latter component,  $\#$ ),  $\lambda_S^{\triangleright}$  is a sub-calculus of  $\lambda_{cS}^{\triangleright}$ . Therefore, we investigate the final calculus: the image of  $\lambda_S$  under the CPS transformation,  $\lambda_S^*$ . The syntax of the calculus is presented in Figure 11.

Note that, as an image of a subcalculus of  $\lambda_{cS}$  under the CPS transformation,  $\lambda_S^*$  is clearly a sub-calculus of  $\lambda_{cS}^*$ . Therefore, we are able to narrow down an isomorphism of  $\lambda_{cS}^*$  and  $\lambda_{cS}^{\triangleright}$  to the appropriate subcalculi arising from  $\lambda_S$ .

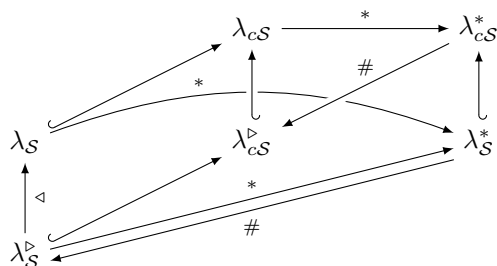
► **Lemma 28** (Isomorphism of  $\lambda_S^{\triangleright}$  and  $\lambda_S^*$ ). *Transformations  $\star : \lambda_S^{\triangleright} \rightarrow \lambda_S^*$  and  $\# : \lambda_S^* \rightarrow \lambda_S^{\triangleright}$  form an isomorphism.*

**Proof.** Isomorphism of wider  $\lambda_{cS}^{\triangleright}$  and  $\lambda_{cS}^*$  calculi can be narrowed. To complete the proof, check that  $\#[\lambda_S^*] \subseteq \lambda_S^{\triangleright}$  and  $\star[\lambda_S^{\triangleright}] \subseteq \lambda_S^*$ . ◀

By now we have established that  $\lambda_S^{\triangleright}$  is isomorphic to both  $\lambda_S$  and  $\lambda_S^*$ . Therefore, by composition of these isomorphisms, we obtain an isomorphism between  $\lambda_S$  and  $\lambda_S^*$ . This establishes a formal connection between a calculus for delimited control in the familiar style and its counterpart in the continuation composing style. In order for the connection to be made more explicit, we compute the composition of  $\#$  and  $\triangleleft$ , which forms the single-pass direct-style transformation from  $\lambda_S^{\triangleright}$  to  $\lambda_S$ . This transformation is dubbed  $\diamond$  and presented in Figure 12. We conclude with the following isomorphism theorem.

$\diamond : \lambda_{\mathcal{S}}^* \rightarrow \lambda_{\mathcal{S}}$	
$(\lambda k . k V_{\varepsilon})^{\diamond}$	$= V_{\varepsilon}^{\natural}$
$(\lambda k . M_{k,\varepsilon})^{\diamond}$	$= M_{k,\varepsilon} \# \varepsilon$
$x_x y_y K_{\Delta,\Sigma} \# \sigma \cdot V \cdot W$	$= K_{\Delta,\Sigma} \flat (\sigma, V W)$
$x_x W_{\varepsilon} K_{\Delta,\Sigma} \# \sigma \cdot V$	$= K_{\Delta,\Sigma} \flat (\sigma, V W_{\varepsilon}^{\natural})$
$V_{\varepsilon} y_y K_{\Delta,\Sigma} \# \sigma \cdot W$	$= K_{\Delta,\Sigma} \flat (\sigma, V_{\varepsilon}^{\natural} W)$
$V_{\varepsilon} W_{\varepsilon} K_{\Delta,\Sigma} \# \sigma$	$= K_{\Delta,\Sigma} \flat (\sigma, \langle V_{\varepsilon}^{\natural} W_{\varepsilon}^{\natural} \rangle)$
$K_{\Delta,\Sigma} ((\lambda x . x) V_{\varepsilon}) \# \sigma$	$= K_{\Delta,\Sigma} \flat (\sigma, \langle V_{\varepsilon}^{\natural} \rangle)$
$K_{\Delta,\Sigma} M_{\bullet,\varepsilon} \# \sigma$	$= K \flat (\sigma, \langle M_{\bullet,\varepsilon} \# \varepsilon \rangle)$
$x^{\natural}$	$= x$
$(\lambda x . R)^{\natural}$	$= \lambda x . R^{\diamond}$
$(\lambda w j . w (\lambda y k . k (j y)) (\lambda x . x))^{\natural}$	$= \mathcal{S}$
$k \flat (\varepsilon, M)$	$= M$
$(\lambda x . x) \flat (\varepsilon, M)$	$= M$
$(\lambda x . N_{\Delta,\Sigma x}) \flat (\sigma, M)$	$= N_{\Delta,\Sigma x} \# \sigma \cdot M$

■ **Figure 12** Back to Direct Style from  $\lambda_{\mathcal{S}}^*$ . We assume  $|\sigma| = |\Sigma|$  for the  $\#$  and  $\flat$  translations. The  $\flat$  translation only works on  $V_{\varepsilon}$  – the other value cases are handled explicitly, hence variables annotated with themselves.



■ **Figure 13** Summary of the relationships between the calculi. Hooked arrows denote sub-calculi,  $*$  denotes the CPS transformation and  $\#$  the DS transformation; both can be retracted along some of the inclusions.

► **Theorem 29** (Isomorphism of  $\lambda_{\mathcal{S}}$  and  $\lambda_{\mathcal{S}}^*$ ). *Transformations  $*$  :  $\lambda_{\mathcal{S}} \rightarrow \lambda_{\mathcal{S}}^*$  and  $\diamond$  :  $\lambda_{\mathcal{S}}^* \rightarrow \lambda_{\mathcal{S}}$  form an isomorphism.*

## 7 Conclusion and Future Work

In this work we established a reflection of the image of the CPS translation for the computational lambda calculus  $\lambda_c$  extended with **shift** and **reset**. We also showed that when restricted to an extension of the call-by-value lambda calculus  $\lambda_v$ , the reflection actually forms an isomorphism. To the best of our knowledge, this is the first study that formally establishes such a tight relationship of the direct-style and CCS reduction theories. In particular, the direct-style translation from CCS to  $\lambda_{\mathcal{S}}$  that is an inverse of the CPS translation, appears to be a first such translation for **shift** and **reset** in the literature. It can be seen as a continuation of the works by Danvy [2], and by Danvy and Lawall [4]. The connections between the various calculi we studied are summarised in Figure 13.



Besides the theoretical aspects of the presented results, one can view them as a source of sound code optimisations that can be performed both at the level of the source and the target of the CPS translation, which is a standard translation step in compilers. Moreover, in the light of Filinski’s seminal result [7], our theory makes it possible to reason about any monadic effect, since the direct-style monad operations `reflect` and `reify` are expressible in terms of `shift` and `reset`.

Several possible directions for future work are on the horizon. First of all, the `shift` operator as considered in this work is a combinator. It seems that if, instead, `shift` was introduced as a special form or as a binder, characterising the image of the CPS translation would require more machinery. Especially in the latter case, we would need to pay special attention to the continuation identifiers bound by `shift`. Introducing a construct `throw` for applying a captured continuation could turn out useful in that scenario.

A delimited-control operator that has been lately gaining currency is `shift0`, a seemingly mild variation on `shift` [3]. This operator is intimately related to the mechanism of algebraic effects and deep handlers [9], a fairly recent and much celebrated approach to computational effects. Establishing a reflection for `shift0`, based on the existing CPS translations [14, 13] would be interesting in its own right, but it could also pave a way to a similar theory for algebraic effects.

Finally, it is quite plausible that following the lines of the present work, one could obtain a similar set of reflections and isomorphisms for a calculus with `call/cc`.

---

## References

- 1 Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, New York, 1992.
- 2 Olivier Danvy. Back to direct style. *Sci. Comput. Program.*, 22(3):183–195, 1994. doi:10.1016/0167-6423(94)00003-4.
- 3 Olivier Danvy and Andrzej Filinski. Abstracting control. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, LFP 1990, Nice, France, 27-29 June 1990*, pages 151–160. ACM, 1990. doi:10.1145/91556.91622.
- 4 Olivier Danvy and Julia L. Lawall. Back to direct style II: first-class continuations. In *Proceedings of the Conference on Lisp and Functional Programming, LFP 1992, San Francisco, California, USA, 22-24 June 1992*, pages 299–310. ACM, 1992. doi:10.1145/141471.141564.
- 5 Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the  $\lambda$ -calculus. In Martin Wirsing, editor, *Formal Description of Programming Concepts - III: Proceedings of the IFIP TC 2/WG 2.2 Working Conference on Formal Description of Programming Concepts - III, Eberup, Denmark, 25-28 August 1986*, pages 193–222. North-Holland, 1987.
- 6 Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992. doi:10.1016/0304-3975(92)90014-7.
- 7 Andrzej Filinski. Representing monads. In Hans-Juergen Boehm, Bernard Lang, and Daniel M. Yellin, editors, *Conference Record of POPL’94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*, pages 446–457. ACM Press, 1994. doi:10.1145/174675.178047.
- 8 Michael J. Fischer. Lambda-calculus schemata. *Lisp and Symbolic Computation*, 6(3-4):259–288, 1993.
- 9 Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *PACMPL*, 1(ICFP):13:1–13:29, 2017. doi:10.1145/3110257.

- 10 Annius Groenink. Literal movement grammars. In Steven P. Abney and Erhard W. Hinrichs, editors, *EACL 1995, 7th Conference of the European Chapter of the Association for Computational Linguistics, March 27-31, 1995, University College Dublin, Belfield, Dublin, Ireland*, pages 90–97. The Association for Computer Linguistics, 1995. URL: <https://www.aclweb.org/anthology/E95-1013/>.
- 11 John Hatcliff and Olivier Danvy. A generic account of continuation-passing styles. In Hans-Juergen Boehm, Bernard Lang, and Daniel M. Yellin, editors, *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*, pages 458–471. ACM Press, 1994. doi:10.1145/174675.178053.
- 12 Yuki Yoshi Kameyama and Masahito Hasegawa. A sound and complete axiomatization of delimited continuations. In Olin Shivers, editor, *Proceedings of the 2003 ACM SIGPLAN International Conference on Functional Programming (ICFP'03)*, SIGPLAN Notices, Vol. 38, No. 9, pages 177–188, Uppsala, Sweden, August 2003. ACM Press.
- 13 Marek Materzok. Axiomatizing subtyped delimited continuations. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPICs*, pages 521–539. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPICs.CSL.2013.521.
- 14 Marek Materzok and Dariusz Biernacki. Subtyping delimited continuations. In Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy, editors, *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 81–93. ACM, 2011. doi:10.1145/2034773.2034786.
- 15 Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23. IEEE Computer Society, 1989. doi:10.1109/LICS.1989.39155.
- 16 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.
- 17 John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998. doi:10.1023/A:1010027404223.
- 18 John C. Reynolds. *Theories of programming languages*. Cambridge University Press, 1998.
- 19 Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3-4):289–360, 1993.
- 20 Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997. doi:10.1145/267959.269968.
- 21 David A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, Inc., 1986.
- 22 Chung-chieh Shan. A static simulation of dynamic delimited control. *Higher-Order and Symbolic Computation*, 20(4):371–401, 2007. doi:10.1007/s10990-007-9010-4.
- 23 Guy L. Steele Jr. Rabbit: A compiler for Scheme. Master's thesis, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1978. Technical report AI-TR-474.
- 24 Christopher Strachey and Christopher P. Wadsworth. Continuations: A mathematical semantics for handling full jumps. *Higher-Order and Symbolic Computation*, 13(1/2):135–152, 2000. doi:10.1023/A:1010026413531.
- 25 Philip Wadler. The essence of functional programming. In Ravi Sethi, editor, *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Albuquerque, New Mexico, USA, January 19-22, 1992*, pages 1–14. ACM Press, 1992. doi:10.1145/143165.143169.