From shift and reset to polarized linear logic

Chung-chieh Shan

Division of Engineering and Applied Sciences
Harvard University
ccshan@post.harvard.edu

Abstract

Griffin [22] pointed out that, just as the pure λ-calculus corresponds to intuitionistic logic, a λ-calculus with first-class continuations corresponds to classical logic. We study how first-class *delimited* continuations [13], in the form of Danvy and Filinski's shift and reset operators [10, 11], can also be logically interpreted.

First, we refine Danvy and Filinski's type system for shift and reset to distinguish between pure and impure functions. This refinement not only paves the way for answer type polymorphism, which makes more terms typable, but also helps us invert the continuation-passing-style (CPS) transform: any pure λ-term with an appropriate type is $\beta\eta$-equivalent to the CPS transform of some shift-reset expression. We conclude that the λ-calculus with shift and reset and the pure λ-calculus have the same logical interpretation, namely good old intuitionistic logic.

Second, we mix delimited continuations with undelimited ones. Informed by the preceding conclusion, we translate the λ-calculus with shift and reset into a *polarized* variant of linear logic [34] that integrates classical and intuitionistic reasoning. Extending previous work on the $\lambda\mu$-calculus [36, 37, 40], this unifying intermediate language expresses computations with and without control effects, on delimited and undelimited continuations, in call-by-value and call-by-name settings.

1 Introduction

The formulas-as-types correspondence, also known as the Curry-Howard isomorphism, treats propositions in a logic as types in a programming language, and proofs of these propositions as programs of these types [21]. In the original instance of this correspondence, Church's λ-calculus [7, 8] is seen to be not just a functional programming language but also a proof system for intuitionistic logic.

Extending the correspondence to classical logic, Griffin [22] pointed out that logical axioms for classical reasoning are types of control operators in programming languages with first-class continuations. For example, the call-with-current-continuation (call/cc) operator in Scheme [31] can be assigned the type $((A \to B) \to A) \to A$, which, read as a proposition, is Pierce's law, a classical reasoning principle invalid in intuitionistic logic. Another example of such a law is double negation, $\neg\neg A \to A$, which is the type of Felleisen's $\mathscr{C}$ operator [14, 15] and the basis for Parigot's $\lambda\mu$-calculus [40] if we read the type $\neg A$ as a continuation waiting for A.

First-class continuations represent "the entire (default) future for the computation" [31]. Refining this concept, Felleisen [13] introduced *delimited* continuations, which encapsulate only a prefix of that future. (Delimited continuations are also called composable, partial, or truncated continuations.) In Felleisen's work, first-class delimited continuations are manipulated using two control operators, # ("prompt") and $\mathscr{F}$. Many variants of these operators have been introduced [10, 11, 23, 42].

Given Griffin's logical interpretation of undelimited continuations, it is natural to ask: is there also a logical interpretation for delimited continuations? Kameyama [28, 29] answers this question for a λ-calculus equipped with what he calls statically scoped delimited continuations. In this paper, we study delimited continuations that in Kameyama's terminology are dynamically scoped. Specifically, we study how to logically interpret Danvy and Filinski's shift and reset operators [10, 11], a popular choice of control operators for delimited continuations. We give two answers to the question of how to logically interpret shift and reset.

From shift and reset to intuitionistic logic The first part of this paper (Sections 2 and 3) argues that, despite the connection Griffin made between first-class continuations and classical logic, a proper logical interpretation of shift and reset requires returning to intuitionistic logic via the continuation-passing-style (CPS) transform. This answer makes sense at the level of types as well as terms.

At the level of types: With delimited continuations, the answer type of a program is not fixed at $\bot$ but changes as the program executes, so the answer type can no longer be left implicit in the classical negation connective. Rather, any logical interpretation of shift and reset must keep track of answer types explicitly, which is precisely what the CPS transform does in the types of its output. Conversely, as we will see in Section 2.2, when the answer type is irrelevant because control effects are absent, it is natural to refine Danvy and Filinski's type system to reflect that fact.

At the level of terms: By showing a "direct-style" transform in Section 3, we formalize and prove the folklore that shift and reset, as control operators, let the programmer tap into *all* of the power in the pure λ-calculus that is the target of the CPS transform. More precisely, any pure λ-term with an appropriate type is $\beta\eta$-equivalent to the CPS conversion of some shift-reset expression.[1] Thus any logical interpreta-

[1] Note the phrase "$\beta\eta$-equivalent to": our "direct-style" transform is less of a CPS-inverse than Danvy and Lawall's [9, 12], which operate up to α-equivalence.

tion of shift and reset must be a logical interpretation of the pure λ-calculus, and vice versa. That is to say, the logical interpretation of shift and reset is simply intuitionistic logic.

From call/cc, shift, and reset to polarized linear logic The second part of this paper (Section 4) gives a single logical interpretation, or equivalently, a unifying intermediate language, that encompasses delimited and undelimited continuations. Since we have found that shift and reset correspond to intuitionistic, not classical, logic, this language must integrate classical and intuitionistic reasoning. Our approach is the judicious use of exponential connectives in linear logic to distinguish between classical and intuitionistic implication, as exemplified by Girard's Logic of Unity [20]. We take Laurent's LLP [34], a *polarized* variant of linear logic that expresses first-class (undelimited) continuations in both call-by-value and call-by-name settings [36, 37], and add a connective that lets us encode intuitionistic logic through linearly used continuations.

Technical contributions and outline To summarize the technical contributions of this paper:

Section 2 refines Danvy and Filinski's type system for shift and reset to distinguish functions with and without control effects.

Section 3 inverts the CPS transform up to $\beta\eta$-equivalence, transforming pure λ-terms back to shift-reset expressions.

Section 4 specifies a polarized variant of classical linear logic that expresses computations with and without control effects, on delimited and undelimited continuations, in call-by-value and call-by-name settings.

2 Calculi for shift and reset

In this section, we present Danvy and Filinski's original λ-calculus with shift and reset, which we call the $\lambda\xi$-calculus. Motivated by a desire to type more terms, we then extend the type system to distinguish pure functions from others.

We assume a fixed set of base types, notated ι. (In examples below, we use Int and Bool as base types, along with term constants like 3.) We use lowercase Roman letters x, f, g, c to name variables (freshly generated as needed), and lowercase Greek letters τ, ω to name types. Typing antecedents Γ are finite maps from variable names to types. Terms are denoted with uppercase Roman letters E, F.

2.1 The original $\lambda\xi$-calculus

Figure 1 presents the $\lambda\xi$-calculus. The only types in this calculus are base types ι and function types $\tau_1/\omega_1 \to \tau_2/\omega_2$. Types of the latter form are to be thought of as functions from τ_1 to τ_2 whose control effect changes the answer type from ω_1 to ω_2. (Product and sum types can easily be added to the system, and the results in this paper carried over.)

Our presentation of the $\lambda\xi$-calculus uses sequents of two kinds. An *impure sequent* is of the form

$$\Gamma \mathbin{/\!\!/} \omega_1 \vdash E : \tau \mathbin{/\!\!/} \omega_2.$$

Its intuitive meaning is that, provided with an environment of type Γ, the expression E can be evaluated in a context whose answer type is ω_1, yielding a result of type τ while changing the answer type to ω_2. The word "impure" here refers to the control effects that may occur when E is evaluated, as indicated by the presence of answer types ω_1 and ω_2 in the sequent. By contrast, a *pure sequent* has the form

$$\Gamma \vdash^{\circ} E : \tau,$$

and guarantees that evaluating E will incur no control effect.

The Axiom typing rule creates a term that retrieves the value of a variable from the environment; this retrieval is pure because the language is call-by-value. The $\to$I rule creates a λ-abstraction, which is also pure because any control effect in the body of the abstraction is delayed until the function is invoked using the $\to$E rule. A pure sequent can be converted to an impure sequent using the Pure rule, which turns a lack of control effects into a trivial effect that leaves the answer type intact. This conversion is semantically an injection, and is left implicit in terms. An impure sequent can be converted to a pure sequent using the Reset rule, which masks control effects in an expression by delimiting the current continuation during its execution. This conversion is semantically a left inverse to the Pure injection, and is notated with angle brackets $\langle \ldots \rangle$ in terms.

The semantics of the $\lambda\xi$-calculus, in particular the meaning of the Shift rule, can be specified operationally, by the reductions shown in Figure 2, or denotationally, by the CPS transform shown in Figure 3. (A sound and complete axiomatization of term equivalence is also available [30].) The target of the CPS transform is the pure λ-calculus, in which

$$\begin{array}{lr}
\text{Types} & \tau, \omega ::= \iota \mid \tau_1/\omega_1 \to \tau_2/\omega_2 \\
\text{Antecedents} & \Gamma ::= x_1 : \tau_1, \ldots, x_n : \tau_n \\
\text{Terms} & E, F ::= x \mid \lambda x.\, E \mid FE \mid \langle E \rangle \mid \xi f.\, E \\
\text{Sequents} & \Gamma \mathbin{/\!\!/} \omega_1 \vdash E : \tau \mathbin{/\!\!/} \omega_2 \quad \text{(impure)} \\
 & \Gamma \vdash^{\circ} E : \tau \quad \text{(pure)}
\end{array}$$

$$\frac{}{\Gamma,\ x : \tau \vdash^{\circ} x : \tau}\ \text{Axiom} \qquad \frac{\Gamma,\ x : \tau_1 \mathbin{/\!\!/} \omega_1 \vdash E : \tau_2 \mathbin{/\!\!/} \omega_2}{\Gamma \vdash^{\circ} \lambda x.\, E : \tau_1/\omega_1 \to \tau_2/\omega_2}\ \to\text{I}$$

$$\frac{\Gamma \mathbin{/\!\!/} \omega_3 \vdash F : \tau_1/\omega_1 \to \tau_2/\omega_2 \mathbin{/\!\!/} \omega_4 \quad \Gamma \mathbin{/\!\!/} \omega_2 \vdash E : \tau_1 \mathbin{/\!\!/} \omega_3}{\Gamma \mathbin{/\!\!/} \omega_1 \vdash FE : \tau_2 \mathbin{/\!\!/} \omega_4}\ \to\text{E}$$

$$\frac{\Gamma \vdash^{\circ} E : \tau}{\Gamma \mathbin{/\!\!/} \omega \vdash E : \tau \mathbin{/\!\!/} \omega}\ \text{Pure} \qquad \frac{\Gamma \mathbin{/\!\!/} \tau \vdash E : \tau \mathbin{/\!\!/} \omega}{\Gamma \vdash^{\circ} \langle E \rangle : \omega}\ \text{Reset}$$

$$\frac{\Gamma,\ f : \tau_1/\omega \to \tau_2/\omega \mathbin{/\!\!/} \omega_1 \vdash E : \omega_1 \mathbin{/\!\!/} \omega_2}{\Gamma \mathbin{/\!\!/} \tau_2 \vdash \xi f.\, E : \tau_1 \mathbin{/\!\!/} \omega_2}\ \text{Shift}$$

Figure 1: The $\lambda\xi$-calculus

$$\begin{array}{lr}
\text{Values} & V ::= \lambda x.\, E \\
\text{Contexts} & C[\,] ::= [\,] \mid (C[\,])E \mid V(C[\,]) \\
\text{Metacontexts} & D[\,] ::= [\,] \mid C[\langle D[\,] \rangle]
\end{array}$$

$$\begin{aligned}
D\big[C[(\lambda x.\, E)V]\big] &\Rightarrow D\big[C[E\{x \mapsto V\}]\big] \\
D\big[C[\langle V \rangle]\big] &\Rightarrow D\big[C[V]\big] \\
D\big[C[\xi f.\, E]\big] &\Rightarrow D\big[E\{f \mapsto \lambda x.\, \langle C[x] \rangle\}\big]
\end{aligned}$$

Figure 2: Reductions for the $\lambda\xi$-calculus

Types and antecedents

$$\lceil \iota \rceil = \iota$$
$$\lceil \tau_1/\omega_1 \to \tau_2/\omega_2 \rceil = \lceil \tau_1 \rceil \to \big(\lceil \tau_2 \rceil \to \lceil \omega_1 \rceil\big) \to \lceil \omega_2 \rceil$$
$$\lceil x_1 : \tau_1, \ldots, x_n : \tau_n \rceil = x_1 : \lceil \tau_1 \rceil, \ldots, x_n : \lceil \tau_n \rceil$$

Pure sequents and terms

$$\big\lceil \Gamma \vdash^{\circ} E : \tau \big\rceil = \lceil \Gamma \rceil \rhd \lceil E \rceil^{\circ} : \lceil \tau \rceil$$
(Axiom) $\lceil x \rceil^{\circ} = x$
(→I) $\lceil \lambda x.\, E \rceil^{\circ} = \lambda x.\, \lceil E \rceil$
(Reset) $\lceil \langle E \rangle \rceil^{\circ} = \lceil E \rceil\, (\lambda x.\, x)$

Impure sequents and terms

$$\big\lceil \Gamma \mathbin{/\!\!/} \omega_1 \vdash E : \tau \mathbin{/\!\!/} \omega_2 \big\rceil = \lceil \Gamma \rceil \rhd \lceil E \rceil : \big(\lceil \tau \rceil \to \lceil \omega_1 \rceil\big) \to \lceil \omega_2 \rceil$$
(→E) $\lceil F E \rceil = \lambda c.\, \lceil F \rceil \big(\lambda f.\, \lceil E \rceil\, (\lambda x.\, f x c)\big)$
(Pure) $\lceil E \rceil = \lambda c.\, c\big(\lceil E \rceil^{\circ}\big)$ if $\lceil E \rceil^{\circ}$ is defined
(Shift) $\lceil \xi f.\, E \rceil = \lambda c.\, \big(\lambda f.\, \lceil E \rceil\big)(\lambda x c'.\, c'(c x))(\lambda x.\, x)$

Figure 3: CPS transform from the $\lambda\xi$-calculus to the pure λ-calculus

terms are typed by sequents of the form

$$\Gamma_\lambda \rhd E_\lambda : \tau_\lambda.$$

Here Γ_λ is an antecedent mapping variable names to pure λ-types, E_λ is a pure λ-term, and τ_λ is its pure λ-type.

Roughly speaking, when a shift-expression $\xi f.\, E$ is evaluated, its context is captured and removed to f, up to the nearest enclosing reset. For example [10], the closed $\lambda\xi$-term

$$1 + \langle 3 + (\xi f.\, f\,0 + f\,1) \rangle : \mathsf{Int}$$

evaluates to 8 via the following sequence of reductions. (The evaluated expression is underlined at each step.)

$$\begin{aligned}
&1 + \big\langle 3 + (\underline{\xi f.\, f\,0 + f\,1}) \big\rangle \\
&\quad \Rightarrow 1 + \big\langle \underline{(\lambda x.\, \langle 3 + x \rangle)(0)} + (\lambda x.\, \langle 3 + x \rangle)(1) \big\rangle \\
&\quad \Rightarrow 1 + \big\langle \big\langle \underline{3 + 0} \big\rangle + (\lambda x.\, \langle 3 + x \rangle)(1) \big\rangle \\
&\quad \Rightarrow 1 + \big\langle \underline{\langle 3 \rangle} + (\lambda x.\, \langle 3 + x \rangle)(1) \big\rangle \\
&\quad \Rightarrow 1 + \big\langle 3 + \underline{(\lambda x.\, \langle 3 + x \rangle)(1)} \big\rangle \Rightarrow 1 + \big\langle 3 + \big\langle \underline{3 + 1} \big\rangle \big\rangle \\
&\quad \Rightarrow 1 + \big\langle 3 + \underline{\langle 4 \rangle} \big\rangle \Rightarrow 1 + \big\langle \underline{3 + 4} \big\rangle \Rightarrow 1 + \underline{\langle 7 \rangle} \Rightarrow \underline{1 + 7} \Rightarrow 8
\end{aligned}$$

Meanwhile, the CPS transform maps the same $\lambda\xi$-term to a pure λ-term that is $\beta\eta$-equivalent to

$$\lambda c.\, c(8) : (\mathsf{Int} \to \omega) \to \omega,$$

which can be applied to the trivial continuation $\lambda x.\, x$ to recover the same answer 8.

The CPS transform given in Figure 3 leaves administrative redexes unreduced. This excess can be removed [10, 11], but is irrelevant to our present purpose, which is to study the logical interpretation of shift and reset. We need only regard pure λ-terms up to $\beta\eta$-equivalence.

2.2 Purity as answer type polymorphism

The following $\lambda\xi$-term cannot be typed.

$$\xi f.\, \mathsf{if}\ \langle f\,2 \rangle\ \mathsf{and}\ f\,3\ \mathsf{then}\ 4\ \mathsf{else}\ 5 \qquad (1)$$

Types $\tau ::= \cdots \mid \tau_1 \xrightarrow{\circ} \tau_2$
Terms $E ::= \cdots \mid \lambda^{\circ} x.\, E \mid F \circ E$

$$\frac{\Gamma,\ x : \tau_1 \vdash^{\circ} E : \tau_2}{\Gamma \vdash^{\circ} \lambda^{\circ} x.\, E : \tau_1 \xrightarrow{\circ} \tau_2}\ {\xrightarrow{\circ}}\mathrm{I} \qquad \frac{\Gamma \vdash^{\circ} F : \tau_1 \to \tau_2 \quad \Gamma \vdash^{\circ} E : \tau_1}{\Gamma \vdash^{\circ} F \circ E : \tau_2}\ {\xrightarrow{\circ}}\mathrm{E}$$

$$\frac{\Gamma \mathbin{/\!\!/} \omega_2 \vdash F : \tau_1 \xrightarrow{\circ} \tau_2 \mathbin{/\!\!/} \omega_3 \quad \Gamma \mathbin{/\!\!/} \omega_1 \vdash E : \tau_1 \mathbin{/\!\!/} \omega_2}{\Gamma \mathbin{/\!\!/} \omega_1 \vdash F \circ E : \tau_2 \mathbin{/\!\!/} \omega_3}\ {\xrightarrow{\circ}}\mathrm{E}'$$

$$\frac{\Gamma,\ f : \tau_1 \xrightarrow{\circ} \tau_2 \mathbin{/\!\!/} \omega_1 \vdash E : \omega_1 \mathbin{/\!\!/} \omega_2}{\Gamma \mathbin{/\!\!/} \tau_2 \vdash \xi f.\, E : \tau_1 \mathbin{/\!\!/} \omega_2}\ \text{Shift (replacing old rule)}$$

Figure 4: The $\lambda\xi^{\circ}$-calculus, where it differs from $\lambda\xi$

The reason is that the two uses of f place conflicting requirements on the answer types part of its type. The first use of f, inside the reset, forces f to take the type $\mathsf{Int/Bool} \to \mathsf{Bool/Bool}$. The second use, outside the reset, calls for the type $\mathsf{Int/Int} \to \mathsf{Bool/Int}$. The Shift rule in Figure 1 lets f take either type, but only one.

A captured continuation is always polymorphic in its answer type, as can be seen in the Shift rule where it assigns to f the type $\tau_1/\omega \to \tau_2/\omega$ for arbitrary ω. Thus the type we really want to assign to f is $\forall\omega.\, \mathsf{Int}/\omega \to \mathsf{Bool}/\omega$. This desire tempts us to invoke let-bound polymorphism [6, 26, 38] on answer types, but even that fails on the following λ-bound version of (1).

$$\xi g.\, (\lambda f.\, \mathsf{if}\ \langle f\,2 \rangle\ \mathsf{and}\ f\,3\ \mathsf{then}\ 4\ \mathsf{else}\ 5)(g) \qquad (2)$$

The reason f is polymorphic in its answer type is that it is pure. That all delimited continuations captured by shift are devoid of control effects can be seen in Figure 2: when reducing a term of the form $\xi f.\, E$, the variable f is bound to a function whose entire body is enclosed in reset. It can also be seen in Figure 3: the CPS transform of $\xi f.\, E$ binds to f a function that only deals with the current continuation c' by passing it the value cx. Unfortunately, the purity of f is lost from the type system by the time f is applied to an argument. There may also be other, user-defined functions, such as $\lambda x.\, x + 1$, that are pure and would benefit from answer type polymorphism.

One way to work around this problem is as follows. In the body E of a shift-expression $\xi f.\, E$, replace all occurrences of f with $\lambda x.\, \langle f x \rangle$. Now the captured continuation f can always take the type $\tau_1/\tau_2 \to \tau_2/\tau_2$; every time we use f, we use reset to "remind" the type system that it is pure. Rewritten thus, the term (1) type-checks successfully.

$$\xi f.\, \mathsf{if}\ \langle (\lambda x.\, \langle f x \rangle)(2) \rangle\ \mathsf{and}\ (\lambda x.\, \langle f x \rangle)(3)\ \mathsf{then}\ 4\ \mathsf{else}\ 5$$

The same workaround applies to λ-bound pure functions, such as f in (2) above: to express a pure function from τ_1 to τ_2 in the type system, use the type $\tau_1/\tau_2 \to \tau_2/\tau_2$ as a proxy, and remind the type system of purity when invoking the function.

$$\xi g.\, (\lambda f.\, \mathsf{if}\ \langle (\lambda x.\, \langle f x \rangle)(2) \rangle\ \mathsf{and}\ (\lambda x.\, \langle f x \rangle)(3)\ \mathsf{then}\ 4\ \mathsf{else}\ 5)(g)$$

We will return to this workaround in Section 3.

A more principled approach is to use an effect system [16, 44, 45] to mechanically keep track of which functions are pure. Figure 4 shows the $\lambda\xi^{\circ}$-calculus, which adds to

$$\begin{array}{ll}\text{Values} & V ::= \cdots \mid \lambda^\circ x.\, E \\ \text{Contexts} & C[\,] ::= \cdots \mid C[\,] \circ E \mid V \circ C[\,] \\ & D\big[C[(\lambda^\circ x.\, E) \circ V]\big] \Rightarrow D\big[C[E\{x \mapsto V\}]\big]\end{array}$$

Figure 5: Reductions for the $\lambda\xi^\circ$-calculus, where it extends $\lambda\xi$

Types

$$\lceil \tau_1 \xrightarrow{\circ} \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$$

Pure terms

$$(\xrightarrow{\circ}\text{I}) \quad \lceil \lambda^\circ x.\, E \rceil^\circ = \lambda x.\, \lceil E \rceil$$

$$(\xrightarrow{\circ}\text{E}) \quad \lceil F \circ E \rceil^\circ = \lceil F \rceil^\circ\, \lceil E \rceil^\circ \text{ if } \lceil F \rceil^\circ,\, \lceil E \rceil^\circ \text{ are defined}$$

Impure terms

$$(\xrightarrow{\circ}\text{E}') \quad \lceil F \circ E \rceil = \lambda c.\, \lceil F \rceil \left(\lambda f.\, \lceil E \rceil \left(\lambda x.\, c(fx)\right)\right)$$

$$(\text{Shift}) \quad \lceil \xi f.\, E \rceil = \lambda f.\, \lceil E \rceil \left(\lambda x.\, x\right)$$

Figure 6: CPS transform from the $\lambda\xi^\circ$-calculus to the pure λ-calculus, where it differs from $\lambda\xi$

the $\lambda\xi$-calculus the types $\tau_1 \xrightarrow{\circ} \tau_2$ and terms $\lambda^\circ x.\, E$ for pure functions. Pure functions are created either by abstracting over an expression using the $\xrightarrow{\circ}$I rule, or by capturing the current continuation using the revised Shift rule. They are invoked using either the $\xrightarrow{\circ}$E rule or the $\xrightarrow{\circ}$E′ rule, depending on whether the computations producing the function and the argument are themselves pure. For clarity, we write $f \circ x$ to apply a pure function f to an argument x, though we could reuse the syntax fx for impure function application.

The terms (1) and (2) are directly typable in the new calculus: unlike in the $\lambda\xi$-calculus, the subterms $f\,2$ and $f\,3$ can now coexist, as $f \circ 2$ and $f \circ 3$.

The reductions of $\lambda\xi^\circ$, shown in Figure 5, trivially extend those of $\lambda\xi$: simply β-reduce pure function applications as one would impure ones. The CPS transform from $\lambda\xi^\circ$ to the pure λ-calculus is shown in Figure 6. The typing rule and CPS transform for shift-expressions are simpler here than in the original $\lambda\xi$-calculus, because they no longer need to convert the captured continuation from pure to impure.[2]

One caveat of the typing rules of the $\lambda\xi^\circ$-calculus is that two derivations may culminate in what is syntactically the same term. For example, if Γ contains $f : \tau_1 \xrightarrow{\circ} \tau_2$ and $x : \tau_1$, then the sequent $\Gamma \mathbin{/\!\!/} \omega \vdash f \circ x : \tau_2 \mathbin{/\!\!/} \omega$ can be derived in two different ways.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash^\circ f : \tau_1 \xrightarrow{\circ} \tau_2}\,\text{Axiom} \quad \dfrac{}{\Gamma \vdash^\circ x : \tau_1}\,\text{Axiom}}{\Gamma \vdash^\circ f \circ x : \tau_2}\,\xrightarrow{\circ}\text{E}}{\Gamma \mathbin{/\!\!/} \omega \vdash f \circ x : \tau_2 \mathbin{/\!\!/} \omega}\,\text{Pure}$$

[2] A further opportunity remains to take advantage of purity as answer type polymorphism: note that the Shift rules in both calculi are still polymorphic in the answer type ω_1. We can simplify Shift by changing its premise to a pure sequent.

$$\dfrac{\Gamma,\, f : \tau_1 \xrightarrow{\circ} \tau_2 \vdash^\circ E : \omega_2}{\Gamma \mathbin{/\!\!/} \tau_2 \vdash \xi f.\, E : \tau_1 \mathbin{/\!\!/} \omega_2}$$

With this change, our term syntax would diverge from Danvy and Filinski's: their $\xi f.\, E$ would become $\xi f.\, \langle E \rangle$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash^\circ f : \tau_1 \xrightarrow{\circ} \tau_2}\,\text{Axiom}}{\Gamma \mathbin{/\!\!/} \omega \vdash f : \tau_1 \xrightarrow{\circ} \tau_2 \mathbin{/\!\!/} \omega}\,\text{Pure} \quad \dfrac{\dfrac{}{\Gamma \vdash^\circ x : \tau_1}\,\text{Axiom}}{\Gamma \mathbin{/\!\!/} \omega \vdash x : \tau_1 \mathbin{/\!\!/} \omega}\,\text{Pure}}{\Gamma \mathbin{/\!\!/} \omega \vdash f \circ x : \tau_2 \mathbin{/\!\!/} \omega}\,\xrightarrow{\circ}\text{E}'$$

This ambiguity is a natural consequence of the (even very weak) effect subtyping that our new calculus engages in. It is harmless because the syntax is still coherent with respect to the semantics: On one hand, the reduction rules are oblivious to purity. On the other hand, the CPS transform maps different derivations of the same $\lambda\xi^\circ$-term to pure λ-terms that are $\beta\eta$-equivalent—in fact, α-equivalent once administrative redexes are reduced.

3 Direct-style transforms

Given that pure functions are explicitly represented in the $\lambda\xi^\circ$-calculus, there is a trivial sense in which the CPS transform from it to the pure λ-calculus is surjective.

Proposition 1. *Every sequent $\Gamma_\lambda \rhd E_\lambda : \tau_\lambda$ derivable in the pure λ-calculus is the CPS-transform image of some (pure) sequent $\Gamma \vdash^\circ E : \tau$ derivable in the $\lambda\xi^\circ$-calculus.*

Proof. The pure λ-calculus embeds into the $\lambda\xi^\circ$-calculus if we map $\to$ to $\xrightarrow{\circ}$ in types, and λ to λ° and fx to $f \circ x$ in terms. □

In a less trivial sense, the CPS transform from both the $\lambda\xi$-calculus and the $\lambda\xi^\circ$-calculus are surjective.

Proposition 2. *Let $\mathscr{L}$ be either the $\lambda\xi$-calculus or the $\lambda\xi^\circ$-calculus. Let E_λ be any pure λ-term, and suppose that Γ is an $\mathscr{L}$-antecedent and τ is an $\mathscr{L}$-type.*

1. *If the sequent*

$$\lceil \Gamma \rceil \rhd E_\lambda : \lceil \tau \rceil$$

is derivable in the pure λ-calculus, then there exists an $\mathscr{L}$-term $\lfloor E_\lambda \rfloor$ so that the pure sequent

$$\Gamma \vdash^\circ \lfloor E_\lambda \rfloor : \tau$$

is derivable in $\mathscr{L}$, and $\lceil \lfloor E_\lambda \rfloor \rceil^\circ$ is $\beta\eta$-equivalent to E_λ.

2. *For any $\mathscr{L}$-types ω_1 and ω_2, if the sequent*

$$\lceil \Gamma \rceil \rhd E_\lambda : \big(\lceil \tau \rceil \to \lceil \omega_1 \rceil\big) \to \lceil \omega_2 \rceil$$

is derivable in the pure λ-calculus, then there exists an $\mathscr{L}$-term $\lfloor E_\lambda \rfloor$ so that the impure sequent

$$\Gamma \mathbin{/\!\!/} \omega_1 \vdash \lfloor E_\lambda \rfloor : \tau \mathbin{/\!\!/} \omega_2$$

is derivable in $\mathscr{L}$, and $\lceil \lfloor E_\lambda \rfloor \rceil$ is $\beta\eta$-equivalent to E_λ.

The goal of this section is to constructively prove this proposition. We first prove it for $\mathscr{L}$ being the $\lambda\xi^\circ$-calculus, then apply the workaround in Section 2.2 for the $\lambda\xi$-calculus.

Proof. For the $\lambda\xi^\circ$-calculus: Without loss of generality, we assume that E_λ is in long $\beta\eta$-normal form [27, 41]. The proof is by induction on the term E_λ, simultaneously for both clauses of the proposition. By the preceding assumption, the term E_λ must be either:

1. $\lambda x.\, E^0_\lambda$, an abstraction; or

2. $f E^1_\lambda \cdots E^n_\lambda$, a variable applied successively to zero or more terms to give a base-type result.

In the first case, we need to invert one of the following three pure-λ sequents to a $\lambda\xi^\circ$-sequent.

$$\lceil\Gamma\rceil \rhd \lambda x.\, E^0_\lambda : \lceil \tau_1/\omega_1 \to \tau_2/\omega_2 \rceil \tag{3}$$

$$\lceil\Gamma\rceil \rhd \lambda x.\, E^0_\lambda : \lceil \tau_1 \xrightarrow{\circ} \tau_2 \rceil \tag{4}$$

$$\lceil\Gamma\rceil \rhd \lambda x.\, E^0_\lambda : \big(\lceil\tau\rceil \to \lceil\omega_1\rceil\big) \to \lceil\omega_2\rceil \tag{5}$$

To invert (3), let $\lfloor E_\lambda \rfloor$ be $\lambda x.\, \lfloor E^0_\lambda \rfloor$. To invert (4), let $\lfloor E_\lambda \rfloor$ be $\lambda^\circ x.\, \lfloor E^0_\lambda \rfloor$. To invert (5), let $\lfloor E_\lambda \rfloor$ be $\xi x.\, \lfloor E^0_\lambda \rfloor$.

In the second case, only the first clause of the proposition is possible, because E_λ has base type. Let τ_0 be the $\lambda\xi^\circ$-type of f in the antecedent Γ. We can write τ_0 as

$$\tau_0 = \tau_1 \xrightarrow{?}_1 \cdots \xrightarrow{?}_{m-1} \tau_m \xrightarrow{?}_m \tau,$$

where, slightly abusing notation, each arrow $\xrightarrow{?}_i$ stands for either "$\xrightarrow{\circ}$" (in which case we call it "pure") or "$/\cdots\to\cdots/$" (in which case we call it "impure"). These arrows associate to the right as usual. For example, if τ_0 is the type

$$\tau_0 = \mathsf{Address/Bool} \to (\mathsf{Char} \xrightarrow{\circ} \mathsf{Double})/(\mathsf{Error} \xrightarrow{\circ} \mathsf{Float}),$$

then we can write it as

$$\tau_0 = \mathsf{Address} \xrightarrow{?}_1 \mathsf{Error} \xrightarrow{?}_2 \mathsf{Float}.$$

The impure arrow $\xrightarrow{?}_1$ stands for "/Bool→(Char$\xrightarrow{\circ}$Double)/"; the pure arrow $\xrightarrow{?}_2$ stands for "$\xrightarrow{\circ}$".

We now iteratively compute terms $E_0, \ldots, E_m$ of the $\lambda\xi^\circ$-calculus, and indices $j_0, \ldots, j_m$ ascending from 0 to n. (The intention is for each E_i to be the direct-style transform of the subterm $f E^1_\lambda \cdots E^{j_i}_\lambda$.) First, set $j_0 = 0$ and $E_0 = f$. Then, for each i from 1 to m, set

$$j_i = j_{i-1} + 1, \qquad E_i = E_{i-1} \circ \lfloor E^{j_i}_\lambda \rfloor \quad \text{if the arrow } \xrightarrow{?}_i \text{ is pure;}$$

$$j_i = j_{i-1} + 2, \qquad E_i = \big\langle \lfloor E^{j_i}_\lambda \rfloor \, \big(E_{i-1}\, \lfloor E^{j_i - 1}_\lambda \rfloor\big) \big\rangle \quad \text{if the arrow } \xrightarrow{?}_i \text{ is impure.}$$

It is easy to check that $j_m = n$. Finally, let $\lfloor E_\lambda \rfloor$ be E_m.

For the $\lambda\xi$-calculus: Take the direct-style transform $\lfloor E_\lambda \rfloor$ constructed above from E_λ for the $\lambda\xi^\circ$-calculus. Replace $F \circ E$ with $(\lambda f x.\, \langle f x\rangle) F E$ throughout the term $\lfloor E_\lambda \rfloor$, and $\tau_1 \xrightarrow{\circ} \tau_2$ with $\tau_1/\tau_2 \to \tau_2/\tau_2$ throughout its type, to obtain a direct-style transform for the $\lambda\xi$-calculus. □

The construction above demonstrates that the CPS transform from shift and reset to the pure λ-calculus covers all of intuitionistic reasoning. Therefore, any logical interpretation of shift and reset must be a logical interpretation of the pure λ-calculus, and vice versa. In other words:

> *The logical interpretation of shift and reset is just intuitionistic logic.*

4 On to polarized linear logic

For the purpose of expressing delimited and undelimited continuations in a single intermediate language, we now consider how to embed classical and intuitionistic reasoning into a single logic. As mentioned in the introduction, we integrate classical and intuitionistic logic by translating both into linear logic.

Positive types	$\phi ::= \iota \mid 1 \mid \phi_1 \otimes \phi_2 \mid {\downarrow}\neg\phi \mid !\neg\phi$
Multiple positive types	$\mu ::= 1 \mid \mu_1 \otimes \mu_2 \mid !\neg\phi$
Antecedents	$\Phi ::= \phi, \ldots, \phi$
Multiple antecedents	$\mathrm{M} ::= \mu, \ldots, \mu$
Sequents	$\Phi \rightsquigarrow (\phi)$

Parenthesized positive formulas like (ϕ) are optional.

$$\frac{}{\phi \rightsquigarrow \phi}\ \text{Axiom} \qquad \frac{\Phi_1,\ \phi \rightsquigarrow (\phi') \quad \Phi_2 \rightsquigarrow \phi}{\Phi_1,\ \Phi_2 \rightsquigarrow (\phi')}\ \text{Cut}$$

$$\frac{\Phi \rightsquigarrow (\phi)}{\Phi,\ 1 \rightsquigarrow (\phi)}\ 1\text{L} \qquad \frac{}{\rightsquigarrow 1}\ 1\text{R}$$

$$\frac{\Phi,\ \phi_1,\ \phi_2 \rightsquigarrow (\phi')}{\Phi,\ \phi_1 \otimes \phi_2 \rightsquigarrow (\phi')}\ \otimes\text{L} \qquad \frac{\Phi_1 \rightsquigarrow \phi_1 \quad \Phi_2 \rightsquigarrow \phi_2}{\Phi_1,\ \Phi_2 \rightsquigarrow \phi_1 \otimes \phi_2}\ \otimes\text{R}$$

$$\frac{\Phi \rightsquigarrow \phi}{\Phi,\ {\downarrow}\neg\phi \rightsquigarrow}\ {\downarrow}\neg\text{L} \qquad \frac{\Phi,\ \phi \rightsquigarrow}{\Phi \rightsquigarrow {\downarrow}\neg\phi}\ {\downarrow}\neg\text{R}$$

$$\frac{\Phi \rightsquigarrow \phi}{\Phi,\ !\neg\phi \rightsquigarrow}\ \text{Dereliction} \qquad \frac{\mathrm{M},\ \phi \rightsquigarrow}{\mathrm{M} \rightsquigarrow !\neg\phi}\ \text{Of Course Not}$$

$$\frac{\Phi \rightsquigarrow (\phi)}{\Phi,\ \mu \rightsquigarrow (\phi)}\ \text{Weakening} \qquad \frac{\Phi,\ \mu,\ \mu \rightsquigarrow (\phi)}{\Phi,\ \mu \rightsquigarrow (\phi)}\ \text{Contraction}$$

Figure 7: CHL, an extension of LLP to incorporate intuitionistic reasoning

Classical implication $A \to B$ can be translated into a linear implication formula $!A \multimap ?B$. The exponential connectives ! ("of course") and ? ("why not") say that the assumption A and the conclusion B, respectively, may be freely copied or discarded in a proof or computation. Because classical logic can be translated into classical linear logic along these lines, the syntax and semantics of linear logic, such as sequent systems and proof nets on one hand, and categorical and game semantics on the other, can be brought to bear on the computational interpretation of classical logic. In particular, programs using undelimited continuations—be they call-by-name or call-by-value—can be translated into a single intermediate language [37], with concomitant theories of confluent proof nets [33, 36] and fully complete game semantics [35]. The image of this translation is Laurent's LLP [34], a *polarized* variant of classical linear logic.

Intuitionistic implication $A \xrightarrow{\circ} B$ can also be translated into an implication formula in classical linear logic, namely $!A \multimap B$. The absence of ? in this formula reflects the crucial difference between classical and intuitionistic logic: in Gentzen's [17] sequent formulations, classical logic (LK) is distinguished from intuitionistic logic (LJ) in allowing multiple conclusions to the right of the turnstile [19]. Accordingly, to integrate classical and intuitionistic reasoning, we extend LLP with an additional connective, *lowered negation*, and use it to enforce the linearity of intuitionistic conclusions.

Figure 7 shows CHL, our polarized variant of linear logic. The formulas in CHL are called *positive types*. Each positive type can be

- a base type ι;
- the multiplicative unit 1;
- the multiplicative conjunction (tensor) $\phi_1 \otimes \phi_2$ of two positive types;

- the *lowered negation* $\downarrow\neg\phi$ of a positive type; or
- the *boxed negation* $!\neg\phi$ of a positive type.

Additive disjunction (plus) $\phi_1 \oplus \phi_2$, which is needed to translate sum types from λ-calculi, is omitted but easy to add.[3]

The word "polarized" refers to the fact that this logic does not allow formulas like $\iota \otimes \neg\iota$ that mix positive and negative types. In polarized systems of linear logic (both CHL and others), multiplicative conjunction is inherently positive and requires positive components, so the only way to tensor the positive type ι with the negative type $\neg\iota$ is to *lower* the latter to the positive $\downarrow\neg\iota$, then build the positive type $\iota \otimes \downarrow\neg\iota$. Dually, multiplicative disjunction $\parr$ is inherently negative and requires negative components, so the only way to express the linear implication $\iota \multimap \iota$ (equivalent to $\neg\iota \parr \iota$) is to *lift* the second ι to the negative $\uparrow\iota$, then build the negative type $\iota \multimap \uparrow\iota$ (equivalent to $\neg\iota \parr \uparrow\iota$ and $\neg(\iota \otimes \downarrow\neg\iota)$).

Polarization of linear logic makes it easier to construct well-behaved syntax and semantics. For example, in game semantics [35], where types are interpreted as games and values as strategies, positive types are interpreted as games in which the proponent moves first, and negative types as games in which the opponent moves first. To lower a negative type to a positive type is to prepend a dummy proponent move to an opponent-first game; dually, to lift a positive type is to prepend a dummy opponent move.

The presentation here of polarized linear logic differs from Laurent's [34] in using positive types only. Because any negative type $\neg\phi$ is immediately lowered (as in $\downarrow\neg\phi$) or boxed (as in $!\neg\phi$) to make a positive type, we can treat negation $\neg$ as a literal part of the unary connectives $\downarrow\neg$ and $!\neg$, rather than as an involution that maps between positive and negative types.

A sequent in CHL is of the form

$$\phi_1, \ldots, \phi_n \rightsquigarrow (\phi),$$

where the antecedent $\phi_1, \ldots, \phi_n$ is a multiset of zero or more positive types, and the conclusion (ϕ) is either a positive type ϕ or nothing. The intended interpretation of the sequent is

$$\phi_1 \otimes \cdots \otimes \phi_n \multimap \phi \qquad \text{or} \qquad \phi_1 \otimes \cdots \otimes \phi_n \multimap \bot,$$

depending on whether a conclusion ϕ is present. In words, the resources in the antecedent can be combined without waste to produce the resource that is the conclusion, if any. Given that classical logic differs from intuitionistic logic in allowing multiple conclusions in a single sequent, it may seem strange that at most one conclusion formula is allowed in a CHL (or LLP) sequent, yet classical logic is supposed to embed into CHL. As it turns out, assumptions and conclusions in classical logic are both encoded as CHL assumptions, so it is best to think of a CHL sequent as a one-sided sequent or a tableau branch.

The CHL inference rules Axiom, Cut, 1L, 1R, $\otimes$L, and $\otimes$R simply restate multiplicative linear logic. More intriguing are the negating connectives $\downarrow\neg$ and $!\neg$ and their associated rules. A lowered negation $\downarrow\neg\phi$ can be thought of as a linearly used (undelimited) continuation [4, 5, 25, 32]. A value of type $\downarrow\neg\phi$ is created by the $\downarrow\neg$R rule; it waits for a ϕ-value to consume using the $\downarrow\neg$L rule. A boxed negation $!\neg\phi$ can be thought of as a non-linearly-used (undelimited) continuation. A value of type $!\neg\phi$ is created by the Of Course Not rule; it can consume zero or more ϕ-values using the Dereliction rule.

Some positive types are *multiple*; they represent data that can be freely copied or discarded in a proof or computation, as controlled by the Weakening and Contraction rules. The multiplicative unit 1 and any boxed negation are multiple; in addition, a tensor is multiple if its components both are. We write μ instead of ϕ for a positive type that is multiple. In LLP, all types are multiple; that is not the case in CHL, which adds the non-multiple connective $\downarrow\neg$ precisely to encode intuitionistic reasoning through linearly used continuations.

Because CHL extends LLP, the call-by-name and call-by-value translations from classical logic to LLP [37] transfer right away to CHL. What is new in CHL is the ability to express intuitionistic reasoning as well. Figure 8 shows how to translate formulas and proofs of intuitionistic logic (and hence, via the CPS transforms in Figures 3 and 6, shift and reset) into those of CHL. The translation is essentially a polarized version of Girard's "boring" translation $*$ from intuitionistic to linear logic [19, p. 81] (which corresponds [3] to Moggi's call-by-value translation to his computational metalanguage [39]).

At the level of formulas, the translation maps each pure-λ type τ to a multiple positive type $\bar{\tau}$ of CHL—multiple, to match how Weakening and Contraction apply freely to assumptions in intuitionistic logic. A base type ι is mapped to the boxed type

$$!\neg\iota, \tag{6}$$

so as to be multiple. A function type $\tau_1 \to \tau_2$ is mapped to the boxed type

$$!\neg(\bar{\tau}_1 \otimes \downarrow\neg\bar{\tau}_2), \tag{7}$$

with the following intuition. To call a function is to produce an argument $(\bar{\tau}_1)$ and at the same time $(\otimes)$ decide what to do after the function returns $(\neg\bar{\tau}_2)$. A pure function returns exactly once; in other words, the return continuation $\neg\bar{\tau}_2$ is used linearly. Thus to call a pure function is to produce a value of the (positive) type $\bar{\tau}_1 \otimes \downarrow\neg\bar{\tau}_2$. A function can be called any number of times, so it is a non-linearly-used (undelimited) continuation that is happy to consume zero or more values of type $\bar{\tau}_1 \otimes \downarrow\neg\bar{\tau}_2$—hence the type in (7). By comparison, the CHL type

$$!\neg(\bar{\tau}_1 \otimes !\neg\bar{\tau}_2) \tag{8}$$

models a function that may not use the return continuation linearly, that is, an impure function from τ_1 to τ_2. Indeed, the type in (8) is the call-by-value translation of the $\lambda\mu$-calculus type $\tau_1 \to \tau_2$.

At the level of proofs, Figure 8 shows the crucial inference rules Axiom, $\to$L and $\to$R in intuitionistic logic (more specifically in Gentzen's LJ [17]) as they translate into partial proofs in CHL. Each (proof of a) LJ sequent

$$\tau_1, \ \ldots, \ \tau_n \rhd \omega$$

is translated to (a proof of) the CHL sequent

$$\bar{\tau}_1, \ \ldots, \ \bar{\tau}_n, \ \downarrow\neg\bar{\omega} \rightsquigarrow$$

[3] Briefly:

$$\phi ::= \cdots \mid 0 \mid \phi \oplus \phi \qquad \mu ::= \cdots \mid 0 \mid \mu \oplus \mu$$

$$\frac{}{0,\ \Phi \rightsquigarrow (\phi)}\ 0\text{L} \qquad \frac{\Phi,\ \phi_1 \rightsquigarrow (\phi') \quad \Phi,\ \phi_2 \rightsquigarrow (\phi')}{\Phi,\ \phi_1 \oplus \phi_2 \rightsquigarrow (\phi')}\ \oplus\text{L}$$

$$\frac{\Phi \rightsquigarrow \phi_1}{\Phi \rightsquigarrow \phi_1 \oplus \phi_2}\ \oplus\text{R1} \qquad \frac{\Phi \rightsquigarrow \phi_1}{\Phi \rightsquigarrow \phi_1 \oplus \phi_2}\ \oplus\text{R2}$$

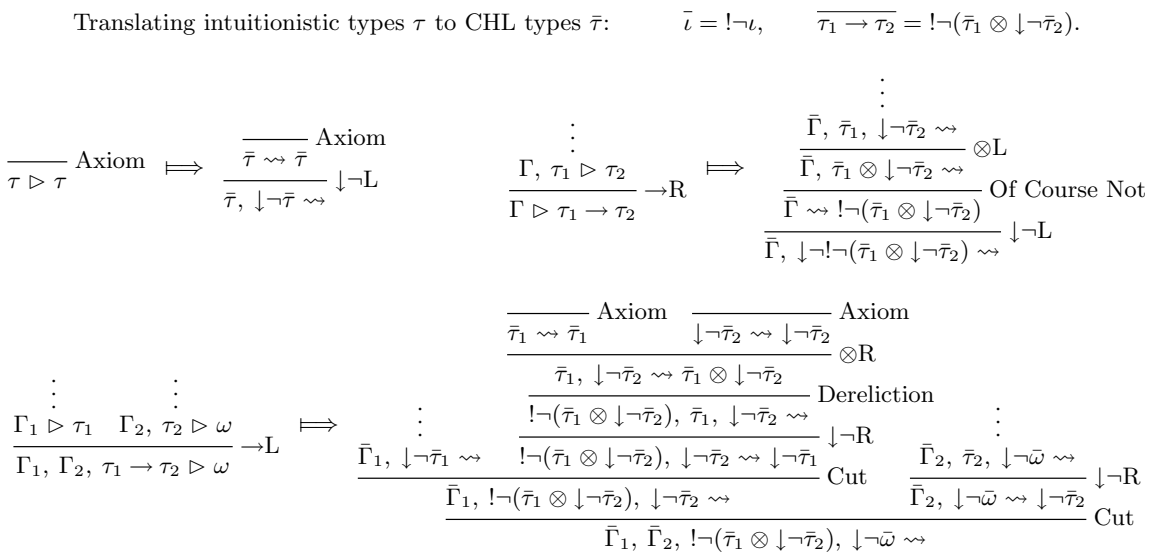

Figure 8: Translating intuitionistic logic into CHL

with no conclusion formula. By contrast, the call-by-value translation from classical logic maps each (proof of a) sequent

$$\tau_1, \ldots, \tau_n \rhd \omega_1, \ldots, \omega_m$$

in Gentzen's LK to (a proof of) the CHL (or LLP) sequent

$$\bar{\tau}_1, \ldots, \bar{\tau}_n, !\neg\bar{\omega}_1, \ldots, !\neg\bar{\omega}_m \rightsquigarrow .$$

Again, these translations of intuitionistic and classical logic differ in whether a return continuation is lowered, as in $\downarrow\neg\omega$, or boxed, as in $!\neg\omega$.

Returning to shift and reset, the translations from LJ and LK to CHL give us a logical interpretation of the distinction between delimited and undelimited continuations that is more refined than the usual statement that an undelimited continuation is a function whose range is the bottom type. In CHL, the type of an undelimited τ-continuation is

$$!\neg\bar{\tau} \cong !\neg(\bar{\tau} \otimes 1) \cong !\neg(\bar{\tau} \otimes !\neg 0) \cong !\neg(\bar{\tau} \otimes \downarrow\neg 0),$$

where 0 is the unit of additive conjunction $\oplus$. Meanwhile, a delimited τ-continuation has the type

$$!\neg(\bar{\tau} \otimes \downarrow\neg\bar{\omega}),$$

where ω is the answer type. Comparing these two types, we see that an undelimited continuation is a delimited continuation with the answer type False, if we add a base type False to the pure λ-calculus and translate it as $\overline{\mathsf{False}} = 0$. Conversely, undelimited continuations are to negation as delimited continuations are to negation relative to an answer type. More generally, intuitionistic implication is *relativized negation*, parameterized by the answer type of a delimited continuation or the return type of a pure function.

Even though the type 0 is multiple, it is not (isomorphic to) the boxed negation of any type, so the base type False is special in that it is not covered by the ordinary translation (6) of base types. This specialness is in contrast to the CHL type $!\neg 1$, which is equivalent to $!\bot$ but not 0. A telling difference between the two "false-looking" types 0 and $!\neg 1$ is that only the former validates Ex Falso Quodlibet. That is, the sequent (in CHL with additives)

$$0, \downarrow\neg\phi \rightsquigarrow$$

is valid for all ϕ, whereas the sequent

$$!\neg 1, \downarrow\neg\phi \rightsquigarrow$$

is not. This difference is explored further in Ariola and Herbelin's work on Minimal Classical Logic [2].

5 Conclusion and related work

According to Section 3, the logical interpretation of shift and reset is simply intuitionistic logic. As explained in Section 4, the difference between intuitionistic and classical logic is that the former uses continuations linearly. Making this linearity explicit in a unifying intermediate language, such as CHL here, allows a programming style that mixes pure code with impure code, or code using shift and reset with code using call/cc.

For example, Berdine et al. [4, 5] express many nontrivial control features (not including full call/cc) under a typing discipline of linearly used continuations. They can maintain this discipline as long as the impurity of the control features can be locally contained—that is, the control effects masked so that the system appears to the environment to be a pure function. This idea is also explored by Thielecke [44], who manages the masking of control effects using an effect system. Like us, Thielecke relates purity to answer type polymorphism and linearly used continuations, though there in the setting of undelimited continuations.

As an intermediate language, CHL tries to be just expressive enough for call/cc, shift, reset, and pure computations. This intention lies in scope between calculi that express classical computations only (such as Parigot's $\lambda\mu$-calculus [40], Laurent and Regnier's LLP and CLC [37], and Wadler's dual calculus [46]) and calculi that provide the full power of classical linear logic (such as Hasegawa [24]'s DCLL). CHL

inherits from LLP its distinction between call-by-value and call-by-name computations; we hope this distinction will let us study call-by-name analogues of shift and reset as control operators.

6 Acknowledgements

Many thanks to Olivier Danvy, Matthias Felleisen, Andrzej Filinski, Masahito Hasegawa, Yukiyoshi Kameyama, Shriram Krishnamurthi, Olivier Laurent, Stuart Shieber, Dylan Thurston, and Philip Wadler for discussions. This work is supported by the United States National Science Foundation under Grant BCS-0236592.

References

[1] ACM. 2003. *ICFP '03: Proceedings of the ACM international conference on functional programming*. New York: ACM Press.

[2] Ariola, Zena M., and Hugo Herbelin. 2003. Minimal classical logic and control operators. In *Proceedings of ICALP 2003: 30th international colloquium on automata, languages, and programming*, ed. Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, 871–885. Lecture Notes in Computer Science 2719, Berlin: Springer-Verlag.

[3] Benton, P. Nick, and Philip Wadler. 1996. Linear logic, monads, and the lambda calculus. In *LICS '96: Proceedings of the 11th symposium on logic in computer science*, 420–431. Washington, DC: IEEE Computer Society Press.

[4] Berdine, Josh, Peter W. O'Hearn, Uday S. Reddy, and Hayo Thielecke. 2001. Linearly used continuations. In [43], 47–54.

[5] ———. 2002. Linear continuation-passing. *Higher-Order and Symbolic Computation* 15(2–3):181–208.

[6] Cardelli, Luca. 1987. Basic polymorphic typechecking. *Science of Computer Programming* 8(2):147–172.

[7] Church, Alonzo. 1932. A set of postulates for the foundation of logic. *Annals of Mathematics* II.33(2):346–366.

[8] ———. 1940. A formulation of the simple theory of types. *Journal of Symbolic Logic* 5(2):56–68.

[9] Danvy, Olivier. 1994. Back to direct style. *Science of Computer Programming* 22(3):183–195.

[10] Danvy, Olivier, and Andrzej Filinski. 1989. A functional abstraction of typed contexts. Tech. Rep. 89/12, DIKU, University of Copenhagen, Denmark. `http://www.daimi.au.dk/~danvy/Papers/fatc.ps.gz`.

[11] ———. 1990. Abstracting control. In *Proceedings of the 1990 ACM conference on Lisp and functional programming*, 151–160. New York: ACM Press.

[12] Danvy, Olivier, and Julia L. Lawall. 1996. Back to direct style II: First-class continuations. Report RS-96-20, BRICS, Denmark.

[13] Felleisen, Matthias. 1988. The theory and practice of first-class prompts. In *POPL '88: Conference record of the annual ACM symposium on principles of programming languages*, 180–190. New York: ACM Press.

[14] Felleisen, Matthias, Daniel P. Friedman, Eugene Kohlbecker, and Bruce Duba. 1987. A syntactic theory of sequential control. *Theoretical Computer Science* 52(3): 205–237.

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

[16] Filinski, Andrzej. 1999. Representing layered monads. In *POPL '99: Conference record of the annual ACM symposium on principles of programming languages*, 175–188. New York: ACM Press.

[17] Gentzen, Gerhard. 1934. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift* 39(2):176–210, 405–431. English translation [18].

[18] ———. 1969. Investigations into logical deduction. In *The collected papers of Gerhard Gentzen*, ed. M. E. Szabo, chap. 3, 68–131. Amsterdam: North-Holland.

[19] Girard, Jean-Yves. 1987. Linear logic. *Theoretical Computer Science* 50:1–101.

[20] ———. 1993. On the unity of logic. *Annals of Pure and Applied Logic* 59(3):201–217.

[21] Girard, Jean-Yves, Paul Taylor, and Yves Lafont. 1989. *Proofs and types*. Cambridge: Cambridge University Press.

[22] Griffin, Timothy G. 1990. A formulae-as-types notion of control. In *POPL '90: Conference record of the annual ACM symposium on principles of programming languages*, 47–58. New York: ACM Press.

[23] Gunter, Carl A., Didier Rémy, and Jon G. Riecke. 1995. A generalization of exceptions and control in ML-like languages. In *Functional programming languages and computer architecture: 7th conference*, ed. Simon Peyton Jones, 12–23. New York: ACM Press.

[24] Hasegawa, Masahito. 2002. Classical linear logic of implications. In *Computer science logic: 16th international workshop, CSL 2002*, ed. Julian C. Bradfield, 458–472. Lecture Notes in Computer Science 2471, Berlin: Springer-Verlag.

[25] ———. 2002. Linearly used effects: Monadic and CPS transformations into the linear lambda calculus. In *Functional and logic programming: Proceedings of FLOPS 2002, 6th international symposium*, ed. Zhenjiang Hu and Mario Rodríguez-Artalejo, 167–182. Lecture Notes in Computer Science 2441, Berlin: Springer-Verlag.

[26] Hindley, J. Roger. 1969. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society* 146:29–60.

[27] Huet, Gérard. 1976. Résolution d'équations dans des langages d'ordre $1, 2, \ldots, \omega$. Thèse de doctorat es sciences mathématiques, Université Paris VII.

[28] Kameyama, Yukiyoshi. 2000. A type-theoretic study on partial continuations. In *Proceedings of IFIP TCS 2000: Theoretical computer science, exploring new frontiers of theoretical informatics*, ed. Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, 489–504. Lecture Notes in Computer Science 1872, Berlin: Springer-Verlag.

[29] ———. 2001. Towards logical understanding of delimited continuations. In [43], 27–33.

[30] Kameyama, Yukiyoshi, and Masahito Hasegawa. 2003. A sound and complete axiomatization of delimited continuations. In [1].

[31] Kelsey, Richard, William Clinger, Jonathan Rees, et al. 1998. Revised^5 report on the algorithmic language Scheme. *Higher-Order and Symbolic Computation* 11(1):7–105. Also as *ACM SIGPLAN Notices* 33(9):26–76.

[32] Laird, James. 2003. A game semantics of linearly used continuations. In *Proceedings of FoSSaCS 2003: Foundations of software science and computational structures, 6th international conference*, ed. Andrew D. Gordon, 313–327. Lecture Notes in Computer Science 2620, Berlin: Springer-Verlag.

[33] Laurent, Olivier. 1999. Polarized proof-nets: Proof-nets for LC (extended abstract). In *TLCA '99: Proceedings of the 4th international conference on typed lambda calculi and applications*, ed. Jean-Yves Girard, 213–227. Lecture Notes in Computer Science 1581, Berlin: Springer-Verlag.

[34] ———. 2002. Étude de la polarisation en logique. Thèse de doctorat, Institut de Mathématiques de Luminy, Université Aix-Marseille II.

[35] ———. 2002. Polarized games (extended abstract). In *LICS 2002: Proceedings of the 17th symposium on logic in computer science*, 265–274. Washington, DC: IEEE Computer Society Press.

[36] ———. 2002. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science* 290(1):161–188.

[37] Laurent, Olivier, and Laurent Regnier. 2003. About translations of classical logic into polarized linear logic. In *LICS 2003: Proceedings of the 18th symposium on logic in computer science*, 11–20. Washington, DC: IEEE Computer Society Press.

[38] Milner, Robin. 1978. A theory of type polymorphism in programming. *Journal of Computer and System Sciences* 17:348–375.

[39] Moggi, Eugenio. 1991. Notions of computation and monads. *Information and Computation* 93(1):55–92.

[40] Parigot, Michel. 1992. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Proceedings of LPAR '92: International conference on logic programming and automated reasoning*, ed. Andrei Voronkov, 190–201. Lecture Notes in Artificial Intelligence 624, Berlin: Springer-Verlag.

[41] Prawitz, Dag. 1971. Ideas and results in proof theory. In *Proceedings of the 2nd Scandinavian logic symposium (1970)*, ed. Jens Erik Fenstad, 235–307. Amsterdam: North-Holland.

[42] Queinnec, Christian, and Bernard Serpette. 1991. A dynamic extent control operator for partial continuations. In *POPL '91: Conference record of the annual ACM symposium on principles of programming languages*, 174–184. New York: ACM Press.

[43] Sabry, Amr, ed. 2001. *CW'01: Proceedings of the 3rd ACM SIGPLAN workshop on continuations*. Tech. Rep. 545, Computer Science Department, Indiana University.

[44] Thielecke, Hayo. 2003. From control effects to typed continuation passing. In *POPL '03: Conference record of the annual ACM symposium on principles of programming languages*, 139–149. New York: ACM Press.

[45] Tofte, Mads. 1990. Type inference for polymorphic references. *Information and Computation* 89(1):1–34.

[46] Wadler, Philip. 2003. Call-by-value is dual to call-by-name. In [1].