Linear $\lambda\mu$ is CP (more or less)

Jennifer Paykin and Steve Zdancewic

University of Pennsylvania
Philadelphia, Pennsylvania 19104, USA
jpaykin@seas.upenn.edu, stevez@cis.upenn.edu

Abstract. In this paper we compare Wadler's CP calculus for classical linear processes to a linear version of Parigot's $\lambda\mu$ calculus for classical logic. We conclude that linear $\lambda\mu$ is "more or less" CP, in that it equationally corresponds to a polarized version of CP. The comparison is made by extending a technique from Melliès and Tabareau's tensor logic that correlates negation with polarization. The polarized CP, which is written CP$^{\pm}$ and pronounced "CP more or less," is an interesting bridge in the landscape of Curry-Howard interpretations of logic.

1 Introduction

In 2012 Philip Wadler introduced CP, a language of "classical processes" in the style of Caires and Pfenning's session-typed processes (2010). CP is a recent advance in the long and distinguished line of work that connects logic to computation via the Curry-Howard correspondence. In this instance, Wadler connects Girard's (classical) linear logic (1987) to a variant of Milner's π-calculus (1992) by using types to express communication protocols—sessions—that synchronize concurrent threads of execution. The connection to logic endows CP with strong correctness properties: type safety and deadlock freedom.

Over the years, research into the Curry-Howard isomorphism has built up a vast landscape of languages and logics. One particularly rich and relevant domain is calculi typed by classical (but not necessarily linear) logic. Among these are Griffin's $\lambda_{\mathcal{C}}$-calculus (1990), Parigot's $\lambda\mu$-calculus (1992), Curien and Heberlin's $\overline{\lambda}\mu\tilde{\mu}$-calculus/System L (2000), and Wadler's own dual calculus (2003). With CP in mind, we can even consider linear versions of these calculi, such as Linear L (Munch-Maccagnoni, 2009; Spiwack, 2014) and linear $\lambda\mu$, the natural linear variant of Parigot's $\lambda\mu$-calculus.

A natural question then arises:

How does CP *relate to linear interpretations of classical calculi?*

In this paper, we answer the question by showing that CP is *almost* (but not quite) the same thing as linear $\lambda\mu$. In particular, linear $\lambda\mu$ corresponds exactly with a *polarized* version of CP, which we introduce here and dub CP$^{\pm}$. Compared to Wadler's original CP, CP$^{\pm}$ (which can be read "CP more or less") separates positive (sending) from negative (receiving) types and inserts shift

connectives between them. Operationally, polarization reduces the amount of nondeterminism, statically making more choices about how processes synchronize.[1] $\mathrm{CP}^{\pm}$ is thus "more or less" CP, where the scheduling is statically decided. We summarize our results in the following Wadleresque slogan:

Linear $\lambda\mu$ is CP *(more or less).*

The relationship between linear $\lambda\mu$ and CP hinges on *duality*. Linear $\lambda\mu$ is a classical logic of two-sided sequents, while CP (and consequently $\mathrm{CP}^{\pm}$) is a classical logic of one-sided sequents. To relate them, we follow a plan laid out by Melliès and Tabareau (2010) in the context of *tensor logic*, which similarly comes in two flavors. The two-sided presentation of tensor logic is an intuitionistic logic of (non-involutive) negation; the one-sided presentation is a polarized logic of (non-invertible) shift operators. When linear $\lambda\mu$ takes the place of the two-sided presentation, $\mathrm{CP}^{\pm}$ arises naturally from the dualization procedure.

The contributions of this paper are summarized as follows. Section 2 presents a core formulation of Wadler's CP, using a novel and elegant operational semantics. Section 3 introduces linear $\lambda\mu$, including both call-by-value and call-by-name operational semantics. We define $\mathrm{CP}^{\pm}$ in Section 4, first describing the duality derived from tensor logic, and then inferring the syntax and semantics from the type structure. Section 5 gives the main result, establishing the equational correspondence between linear $\lambda\mu$ and $\mathrm{CP}^{\pm}$. We briefly conclude with a discussion of the relationships between CP, $\mathrm{CP}^{\pm}$, and other computational interpretations of classical logics.

2 CP and Session Types

In this section we describe a variation on Wadler's CP calculus (2012; 2014), based on the line of work by Caires and Pfenning (2010) that treats the types of linear logic as descriptions of session protocols and the proofs of linear logic as processes obeying these protocols. A (binary) session is just a communication channel between two processes. The types of linear logic describe the protocols by which these channels should behave.

$$X, Y ::= 1 \mid X \otimes Y \mid \bot \mid X \parr Y \mid 0 \mid X \oplus Y \mid \top \mid X \mathbin{\&} Y$$

Consider a process P that offers a channel x obeying protocol X. The process at the other end of the channel, Q, must obey the *dual* protocol $X^{\bot}$. Duality is involutive—that is, we have $(X^{\bot})^{\bot} = X$. We write the composition of these two processes as $\nu x.\langle P \mid Q \rangle$, and the session type X describes their behavior as shown in Figure 1.

Each connective has an associated direction—processes *send* messages over channels behaving like a tensor $\otimes$, but they *receive* messages over channels behaving like a par $\parr$. However, the direction of each channel is not fixed. For

[1] Pfenning and Griffith (2015) have also studied polarization in their work on intuitionistic session types, where it distinguishes synchronous and asynchronous communication.

$x : X$	P	$x : X^\perp$	Q
1	sends an empty message over x and immediately halts	$\perp$	receives an empty message on x and closes the channel
$X_1 \otimes X_2$	sends a channel obeying X_1 over x, then behaves like X_2	${X_1}^\perp \parr {X_2}^\perp$	receives a channel obeying ${X_1}^\perp$ over x, then behaves like ${X_2}^\perp$
0	(no channels obey 0)	$\top$	aborts
$X_1 \oplus X_1$	sends a flag indicating whether x will behave as X_1 or X_2	${X_1}^\perp \mathbin{\&} {X_2}^\perp$	receives a flag indicating that x will behave as ${X_1}^\perp$ or ${X_2}^\perp$

Fig. 1. Session types and their intended semantics. Process P interacts with Q via the cut $\nu x.\langle P \mid Q\rangle$. By duality, P could equally well offer a channel behaving as $X^\perp$.

example, a process may send a value on a channel, then immediately receive a flag on the same channel. The session-based type of such a channel would be $X_1 \otimes (X_2 \mathbin{\&} X_3)$.

2.1 Syntax and Static Semantics

As in the π-calculus, a process P is made up of bodies and prefixes to manipulate channels.

$$\begin{array}{lll} P ::= x \leftrightarrow y & & \mid \nu x.\langle P_1 \mid P_2\rangle \\ \quad \mid x[].0 & & \mid x().P \\ \quad \mid x[y].(P_1 \mid P_2) & & \mid x(y).P \\ & & \mid x.\mathbf{case}() \\ \quad \mid x[\mathbf{inj}_i].P & & \mid x.\mathbf{case}(P_1 \mid P_2) \end{array}$$

The link operator $x \leftrightarrow y$ connects the channels x and y together. The cut operator $\nu x.\langle P_1 \mid P_2\rangle$ synchronizes communication on a channel whose endpoints lie in P_1 and P_2. The other syntactic forms are *prefix actions*, which send or receive on channels according to one of the session protocols.

The typing judgment $P \vdash \Omega$ specifies that the channels offered by P obey certain protocols, specified by Ω. Here Ω is a collection of channel names x with their types X. The typing rules for CP are shown in Figure 2.

2.2 Operational Semantics

Wadler (2012) nicely summarizes the Curry-Howard interpretation of session types as follows: propositions are session types, proofs are processes, and *cut elimination is communication*. The operational behavior of processes is the description of how communication occurs over channels. The end goal of this communication is a process that is free of both cuts and (non-atomic) links. With that goal in mind, each step of communication falls into one of three groups.

$$\frac{}{y \leftrightarrow x \vdash y : A^{\perp}, x : A}\ \text{AX} \qquad \frac{P_1 \vdash \Omega_1, x : A \quad P_2 \vdash x : A^{\perp}, \Omega_2}{\nu x.\langle P_1 \mid P_2\rangle \vdash \Omega_1, \Omega_2}\ \text{CUT}$$

$$\frac{}{x[].0 \vdash x : 1}\ 1 \qquad \frac{P \vdash \Omega}{x().P \vdash x : \perp, \Omega}\ \perp$$

$$\frac{P_1 \vdash \Omega_1, y : A \quad P_2 \vdash \Omega_2, x : B}{x[y].(P_1 \mid P_2) \vdash \Omega_1, \Omega_2, x : A \otimes B}\ \otimes \qquad \frac{P \vdash y : A, x : B, \Omega}{x(y).P \vdash x : A \parr B, \Omega}\ \parr$$

$$\text{(no 0 rule)} \qquad \frac{}{x.\mathbf{case}() \vdash x : \top, \Omega}\ \top$$

$$\frac{P \vdash \Omega, x : A_i}{x[\mathbf{inj}_i].P \vdash \Omega, x : A_1 \oplus A_2}\ \oplus \qquad \frac{P_1 \vdash x : A, \Omega \quad P_2 \vdash x : B, \Omega}{x.\mathbf{case}(P_1 \mid P_2) \vdash x : A \,\&\, B, \Omega}\ \&$$

Fig. 2. The CP calculus

First, β-reduction rules describe the synchronous communication between two processes.

$$\begin{aligned}
\nu x.\langle y \leftrightarrow x \mid P\rangle &\to_\beta P\{y/x\} \\
\nu x.\langle x[].0 \mid x().P\rangle &\to_\beta P \\
\nu x.\langle x[y].(P_1 \mid P_2) \mid x(y).P_3\rangle &\to_\beta \nu y.\langle P_1 \mid \nu x.\langle P_2 \mid P_3\rangle\rangle \\
\nu x.\langle x[\mathbf{inj}_i].P \mid x.\mathbf{case}(P_1 \mid P_2)\rangle &\to_\beta \nu x.\langle P \mid P_i\rangle
\end{aligned}$$

Structural equivalence rules allow us to swap the order of cuts and forwarding links in these rules to avoid unnecessary duplication.

$$\nu x.\langle P_1 \mid P_2\rangle \equiv_s \nu x.\langle P_2 \mid P_1\rangle \qquad\qquad x \leftrightarrow y \equiv_s y \leftrightarrow x \tag{1}$$

Second, link forwarding is defined by η-expansion rules.

$$\begin{aligned}
(y \leftrightarrow x \vdash y : \perp, x : 1) &\to_\eta y().x[].0 \\
(y \leftrightarrow x \vdash y : X^{\perp} \parr Y^{\perp}, x : X \otimes Y) &\to_\eta y(y').x[x'].(y' \leftrightarrow x' \mid y \leftrightarrow x) \\
(y \leftrightarrow x \vdash y : \top, x : 0) &\to_\eta y.\mathbf{case}() \\
(y \leftrightarrow x \vdash y : A^{\perp} \,\&\, B^{\perp}, x : A \oplus B) &\to_\eta y.\mathbf{case}(x[\mathbf{inj}_1].y \leftrightarrow x \mid x[\mathbf{inj}_2].y \leftrightarrow x)
\end{aligned}$$

The rest of the operational behavior of processes deals with *commuting conversions*, by which non-interfering actions can be executed in any order. Presentations of commuting conversions are often very intricate, as there are a quadratic number of interleavings of non-interfering actions. Here, we present a unified view of commuting conversions by classifying them into three groups.

The first group of commuting conversions permutes an action (corresponding to one of the session protocols) around a cut. For example, when x does not occur in P_2 and y does not occur in P_1, then

$$\nu x.\langle P_1 \mid y[z].(P_2 \mid P_3)\rangle \to_{CC} y[z].(P_2 \mid \nu x.\langle P_1 \mid P_3\rangle).$$

The presence of the additives, $\top$ and &, adds even stranger looking conversions:

$$\nu x.\langle y.\mathbf{case}() \mid Q\rangle \to_{CC} y.\mathbf{case}()$$
$$\nu x.\langle y.\mathbf{case}(P_1 \mid P_2) \mid Q\rangle \to_{CC} y.\mathbf{case}(\nu x.\langle P_1 \mid Q\rangle \mid \nu x.\langle P_2 \mid Q\rangle)$$

This class of conversions is *directed*, to push the cut after the action so that it becomes closer to its own synchronizing channel. The combination of this class of conversions with the β-reduction rules results in a cut elimination procedure (Theorem 1).

The second group of commuting conversions says that two cuts on different channels can be performed in either order. Assuming that x does not occur in P_2 and y does not occur in P_1, we have

$$\nu x.\langle P_1 \mid \nu y.\langle P_2 \mid P_3\rangle\rangle \equiv_{CC} \nu y.\langle P_2 \mid \nu x.\langle P_1 \mid P_3\rangle\rangle.$$

This type of conversion is classified by Wadler as a structural equivalence, which we denote $\equiv_s$. However, while the two structural equivalences in Equation (1) are necessary to define the normalization or communication procedure of processes, the *conversion* in this section is not.

The third class of conversions says that prefix actions on different channels can occur in any order. For example, if x (which is not equal to z) does not occur in P_1, then

$$x[\mathbf{inj}_1].z[y].(P_1 \mid P_2) \equiv_{CC} z[y].(P_1 \mid x[\mathbf{inj}_1].P_2).$$

The motivation behind this type of commuting conversion is the independence of non-interfering sessions. That is, when two prefixes do not refer to each other, they should not interfere. Again, this class of conversions is undirected; it is not necessary for the definition of a communcation/cut-elimination procedure. The relation $\equiv_{CC}$ instead defines *behavioral equivalence* of processes, as described by Pérez et al. (2014).

Commuting Conversions via Contexts. A context C is any process with holes (possibly zero or more than one) for other processes. The number of holes in C is called its arity. We write $C[P_1, .., P_i]$ for the operation that fills holes with processes.[2] Two contexts are disjoint, written $C_1 \perp C_2$, if the set of channels referred to by C_1 is disjoint from the set of channels referred to by C_2.

The composition of two contexts $C_2 \circ C_1$ replicates the inner context C_1 in all of the holes of the outer context C_2. This is illustrated by the following examples:

$$\nu x.\langle \Box \mid P\rangle \circ C = \nu x.\langle C \mid P\rangle$$
$$x.\mathbf{case}() \circ C = x.\mathbf{case}()$$
$$x.\mathbf{case}(\Box \mid \Box) \circ C = x.\mathbf{case}(C \mid C)$$

[2] To give the definitions in this section precisely, it would be necessary to use named holes and substitutions to fill in the holes. For the purposes of this paper we leave these operations informal.

The arity of $C_1 \circ C_2$ is the product of the arities of the two subcontexts.

For any context C with a single hole, we have a congruence rule to reduce processes under actions.

$$\frac{P \equiv_s P'}{C[P] \equiv_s C[P']} \tag{2}$$

To define commuting conversions, we pick out certain classes of contexts that refer to the different components of the commuting conversion rules. Evaluation contexts, written C^e, consist of cuts with a single top-level hole, and action contexts, written C^a, consist of prefix actions also with top-level holes.

$$\begin{aligned} C^e &::= \nu x.\langle \Box \mid P_2 \rangle \mid \nu x.\langle P_1 \mid \Box \rangle \\ C^a &::= x().\Box \mid x[y].(\Box \mid P_2) \mid x[y].(P_1 \mid \Box) \mid x(y).\Box \\ &\quad \mid x.\mathbf{case}() \mid x[\mathbf{inj}_i].\Box \mid x.\mathbf{case}(\Box \mid \Box) \end{aligned}$$

Notice that $x.\mathbf{case}()$ is a context with arity zero, and $x.\mathbf{case}(\Box \mid \Box)$ is a context with arity two.

The three classes of commuting conversion rules can be summarized neatly as follows:

$$\frac{C^e \perp C^a}{C^e \circ C^a[P_1, .., P_i] \rightarrow_{CC} C^a \circ C^e[P_1, .., P_i]} \text{ ACTION-CUT}$$

$$\frac{C_1^e \perp C_2^e}{C_1^e \circ C_2^e[P] \equiv_{CC} C_2^e \circ C_1^e[P]} \text{ CUT-CUT}$$

$$\frac{C_1^a \perp C_2^a}{C_1^a \circ C_2^a[P_1, .., P_i] \equiv_{CC} C_2^a \circ C_1^a[P_1, .., P_i]} \text{ ACTION-ACTION}$$

2.3 Reduction and Equivalence

The step relation on processes is the least relation generated by $\rightarrow_\beta$, $\rightarrow_\eta$, $\rightarrow_{CC}$, and $\equiv_s$. Behavioral equivalence, written $\equiv$, is defined to be the least congruence containing the step relation and $\equiv_{CC}$.

Theorem 1 (Progress and Preservation).

1. *If $P \vdash \Omega$ then either P is cut- and link-free, or P can take a step.*
2. *If $P \vdash \Omega$ and P steps to P', then $P' \vdash \Omega$.*

The structural equivalence relation $\equiv_s$ makes reduction in CP extremely non-deterministic. We conjecture that the step relation is confluent up to commuting conversions $\equiv_{CC}$, but have not proved this fact. In addition, there is no meaningful notion of evaluation order; the properties of call-by-name and call-by-value are not even expressible. In Section 4 we will see that $\text{CP}^{\pm}$ has the ability to resolve this non-determinism by employing different evaluation strategies, in the style of Wadler's dual calculus (2003).

3 Linear $\lambda\mu$ and Linear Classical Logic

In this section we answer the question: what is a linearized classical language, as opposed to a language for classical linear logic? Just as the simply-typed λ-calculus can be *linearized* to obtain a calculus for intuitionistic linear logic, we can similarly linearize Parigot's $\lambda\mu$-calculus for classical logic (1992). Such systems have been considered before, for example by Munch-Maccagnoni (2009) and Spiwack (2014).

Following Wadler (2005), we take the types of linear $\lambda\mu$ to consist of negation, units, products and sums—implication is defined in terms of negation and products. Products and sums are written in the style of linear logic.

$$A, B ::= \neg A \mid 1 \mid A \otimes B \mid 0 \mid A \oplus B$$

3.1 Syntax and Static Semantics

The syntax of $\lambda\mu$ consists of two parts: terms and commands. Terms t are typed expressions, while commands c have no types, but instead define a relation between the two different sorts of variables. Term variables, written x, can be thought of in the usual way—as providing input to an expression. Continuation variables, written α, consume the output of an expression.

The quintessential command feeds a term t into a continuation variable α, and is written $[\alpha]t$. On the other hand, if a continuation variable α is used in a command c, the term $\mu\alpha.c$ extracts the value passed to α inside of c.

The syntax of terms and commands is summarized below. We use expressions e to refer to either syntactic form.

$$\begin{array}{ll}
t ::= x & \mid \mu\alpha.c \\
\mid \overline{\lambda}x.c & \mid \mathbf{let}\, x = t_1\, \mathbf{in}\, t_2 \\
\mid () & \mid \mathbf{let}\, () = t_1\, \mathbf{in}\, t_2 \\
\mid (t_1, t_2) & \mid \mathbf{let}\, (x_1, x_2) = t_1\, \mathbf{in}\, t_2 \\
 & \mid \mathbf{case}\, t\, \mathbf{of}\, () \\
\mid \mathbf{inj}_i\, t & \mid \mathbf{case}\, t\, \mathbf{of}\, (\mathbf{inj}_1\, x_1 \to t_1, \mathbf{inj}_2\, x_2 \to t_2) \\
c ::= [\alpha]t & \\
\mid t_1\, t_2 & \mid \mathbf{let}\, x = t\, \mathbf{in}\, c \\
\mid \mathbf{let}\, () = t\, \mathbf{in}\, c & \mid \mathbf{let}\, (x_1, x_2) = t\, \mathbf{in}\, c \\
\mid \mathbf{case}\, t\, \mathbf{of}\, () & \mid \mathbf{case}\, t\, \mathbf{of}\, (\mathbf{inj}_1\, x_1 \to c_1, \mathbf{inj}_2\, x_2 \to c_2)
\end{array}$$

There are similarly two typing judgments: $\Gamma \vdash t : A \mid \Pi$ for terms, and $\Gamma \vdash c \mid \Pi$ for commands. Here Γ is a context of term variables x, and Π is a context of continuation variables α. For rules that are polymorphic in the judgment, we write $\Gamma \vdash e : \Theta \mid \Pi$, where Θ is a stoup of either zero or one types.

Figure 3 shows the typing rules. Many of the rules are unsurprising for a linear λ-calculus. Note that $\overline{\lambda}$ is used for negation abstraction instead of the usual λ, and every application is a command, not a term.

$$\frac{}{x : A \vdash x : A \mid \cdot}\ \text{VAR} \qquad \frac{\Gamma_1 \vdash t : A \mid \Pi_1 \quad \Gamma_2, x : A \vdash e : \Theta \mid \Pi_2}{\Gamma_1, \Gamma_2 \vdash \mathbf{let}\, x = t\, \mathbf{in}\, e : \Theta \mid \Pi_1, \Pi_2}\ \text{LET}$$

$$\frac{\Gamma \vdash c \mid \Pi, \alpha : A}{\Gamma \vdash \mu\alpha.c : A \mid \Pi}\ \mu\text{-I} \qquad \frac{\Gamma \vdash t : A \mid \Pi}{\Gamma \vdash [\alpha]t \mid \Pi, \alpha : A}\ \mu\text{-E}$$

$$\frac{\Gamma, x : A \vdash c \mid \Pi}{\Gamma \vdash \overline{\lambda} x.c : \neg A \mid \Pi}\ \neg\text{-I} \qquad \frac{\Gamma_1 \vdash t_1 : \neg A \mid \Pi_1 \quad \Gamma_2 \vdash t_2 : A \mid \Pi_2}{\Gamma_1, \Gamma_2 \vdash t_1\, t_2 \mid \Pi_1, \Pi_2}\ \neg\text{-E}$$

$$\frac{}{\cdot \vdash () : 1 \mid \cdot}\ 1\text{-I} \qquad \frac{\Gamma_1 \vdash t : 1 \mid \Pi_1 \quad \Gamma_2 \vdash e : \Theta \mid \Pi_2}{\Gamma_1, \Gamma_2 \vdash \mathbf{let}\, () = t\, \mathbf{in}\, e : \Theta \mid \Pi_1, \Pi_2}\ 1\text{-E}$$

$$\frac{\Gamma_1 \vdash t_1 : A_1 \mid \Pi_1 \quad \Gamma_2 \vdash t_2 : A_2 \mid \Pi_2}{\Gamma_1, \Gamma_2 \vdash (t_1, t_2) : A_1 \otimes A_2 \mid \Pi_1, \Pi_2}\ \otimes\text{-I}$$

$$\frac{\Gamma_1 \vdash t : A \otimes B \mid \Pi_1 \quad \Gamma_2, x : A, y : B \vdash e : \Theta \mid \Pi_2}{\Gamma_1, \Gamma_2 \vdash \mathbf{let}\, (x, y) = t\, \mathbf{in}\, e : \Theta \mid \Pi_1, \Pi_2}\ \otimes\text{-E}$$

$$\frac{\Gamma \vdash t : 0 \mid \Pi}{\Gamma \vdash \mathbf{case}\, t\, \mathbf{of}\, () : \Theta \mid \Pi}\ 0\text{-E} \qquad \frac{\Gamma \vdash t : A_i \mid \Pi}{\Gamma \vdash \mathbf{inj}_i\, t : A_1 \oplus A_2 \mid \Pi}\ \oplus\text{-I}$$

$$\frac{\Gamma_1 \vdash t : A_1 \oplus A_2 \mid \Pi_1 \quad \Gamma_2, x_1 : A_1 \vdash e_1 : \Theta \mid \Pi_2 \quad \Gamma_2, x_2 : A_2 \vdash e_2 : \Theta \mid \Pi_2}{\Gamma_1, \Gamma_2 \vdash \mathbf{case}\, t\, \mathbf{of}\, (\mathbf{inj}_1\, x_1 \to e_1, \mathbf{inj}_2\, x_2 \to e_2) : \Theta \mid \Pi_1, \Pi_2}\ \oplus\text{-E}$$

Fig. 3. The Linear $\lambda\mu$-calculus

To understand how these language features interact, consider the encoding of linear implication $A \multimap B$ in classical logic as $\neg\,(A \otimes \neg\, B)$. We can encode the application rule as

$$\frac{\Gamma_1 \vdash t_1 : A \multimap B \mid \Pi_1 \qquad \Gamma_2 \vdash t_2 : A \mid \Pi_2}{\Gamma_1, \Gamma_2 \vdash t_1\, t_2 : B \mid \Pi_1, \Pi_2}\ \multimap\text{-E} \qquad \Rightarrow$$

$$\dfrac{\dfrac{\Gamma_1 \vdash t_1 : \neg\,(A \otimes \neg\, B) \mid \Pi_1 \qquad \dfrac{\Gamma_2 \vdash t_2 : A \mid \Pi_2 \qquad \dfrac{\dfrac{\dfrac{}{y : B \vdash y : B \mid \cdot}}{y : B \vdash [\beta]y \mid \beta : B}}{\cdot \vdash \overline{\lambda} y.[\beta]y : \neg\, B \mid \beta : B}}{\Gamma_2 \vdash (t_2, \overline{\lambda} y.[\beta]y) : A \otimes \neg\, B \mid \Pi_2, \beta : B}}{\Gamma_1, \Gamma_2 \vdash t_1\, (t_2, \overline{\lambda} y.[\beta]y) \mid \Pi_1, \Pi_2, \beta : B}}{\Gamma_1, \Gamma_2 \vdash \mu\beta.t_1\, (t_2, \overline{\lambda} y.[\beta]y) : B \mid \Pi_1, \Pi_2}$$

We can also consider the encoding of Felleisen's $\mathcal{C}$ operator (1988), which is typed by double negation elimination.

$$\dfrac{\dfrac{\Gamma \vdash t : \neg\neg A \mid \Delta \qquad \overline{\cdot \vdash \overline{\lambda}x.[\alpha]x : \neg A \mid \alpha : A}}{\Gamma \vdash t\,(\overline{\lambda}x.[\alpha]x) \mid \Delta, \alpha : A}}{\Gamma \vdash \mu\alpha.t\,(\overline{\lambda}x.[\alpha]x) : A \mid \Delta}$$

3.2 Operational Semantics

In this section we define an operational semantics for linear $\lambda\mu$ in both call-by-value and call-by-name style. We first give the β and η rules that are shared by the two reduction strategies. The β rules are given in terms of let bindings, and the difference between CBV and CBN comes down to the treatment of let reduction and evaluation contexts, as shown in Figure 4.

First, the β rules:

(μ)	$-$	
$(\neg)$	$(\overline{\lambda}x.c)\,t$	$\to_\beta \mathbf{let}\, x = t \,\mathbf{in}\, c$
(1)	$\mathbf{let}\,() = ()\,\mathbf{in}\, e$	$\to_\beta e$
$(\otimes)$	$\mathbf{let}\,(x_1, x_2) = (t_1, t_2)\,\mathbf{in}\, e$	$\to_\beta \mathbf{let}\, x_1 = t_1 \,\mathbf{in}\, \mathbf{let}\, x_2 = t_2 \,\mathbf{in}\, e$
(0)	$-$	
$(\oplus)$	$\mathbf{case}\,\mathbf{inj}_i\, t\,\mathbf{of}\,(\mathbf{inj}_1\, x_1 \to e_1, \mathbf{inj}_2\, x_2 \to e_2)$	$\to_\beta \mathbf{let}\, x_i = t \,\mathbf{in}\, e_i$

Next we have local η expansions:

(μ)	$t : A$	$\to_\eta \mu\alpha.[\alpha]t$
$(\neg)$	$t : \neg A$	$\to_\eta \overline{\lambda}x.t\,x$
(1)	$t : 1$	$\to_\eta \mathbf{let}\,() = t\,\mathbf{in}\,()$
$(\otimes)$	$t : A_1 \otimes A_2$	$\to_\eta \mathbf{let}\,(x_1, x_2) = t\,\mathbf{in}\,(x_1, x_2)$
(0)	$t : 0$	$\to_\eta \mathbf{case}\, t\,\mathbf{of}\,()$
$(\oplus)$	$t : A_1 \oplus A_2$	$\to_\eta \mathbf{case}\, t\,\mathbf{of}\,(\mathbf{inj}_1\, x_1 \to \mathbf{inj}_1\, x_1, \mathbf{inj}_2\, x_2 \to \mathbf{inj}_2\, x_2)$

Terms reduce under arbitrary evaluation contexts, and under μ binders. Notice that evaluation contexts are closed under term variables x; we only consider reduction over closed terms. However, evaluation contexts are open under continuation variables α, and these contexts characterize the capturing of μ binders inside larger expressions.

$$\frac{t \to t'}{E[t] \to E[t']} \qquad \frac{c \to c'}{\mu\alpha.c \to \mu\alpha.c'} \qquad \frac{E[\mu\alpha.c] \text{ is a command}}{E[\mu\alpha.c] \to c\{E/\alpha\}}$$

As an example of the last rule, consider the application $(\mu\alpha.c)\,t$ which captures the current context $\Box\, t$ as the continuation α inside the command c. To achieve

CBV	CBN
$\mathbf{let}\, x = v\, \mathbf{in}\, e \to_\beta e\{v/x\}$	$\mathbf{let}\, x = t\, \mathbf{in}\, e \to_\beta e\{t/x\}$
$v ::= \overline{\lambda} x.c \mid () \mid (v_1, v_2) \mid \mathbf{inj}_i\, v$	$v ::= \overline{\lambda} x.c \mid () \mid (t_1, t_2) \mid \mathbf{inj}_i\, t$
$E ::= \square \mid [\alpha]E$	$E ::= \square \mid [\alpha]E$
$\mid E\, t \mid \mathbf{let}\, x = E\, \mathbf{in}\, e$	$\mid E\, t$
$\mid \mathbf{let}\, () = E\, \mathbf{in}\, e$	$\mid \mathbf{let}\, () = E\, \mathbf{in}\, e$
$\mid (E, t) \mid (v, E) \mid \mathbf{let}\, (x_1, x_2) = E\, \mathbf{in}\, e$	$\mid \mathbf{let}\, (x_1, x_2) = E\, \mathbf{in}\, e$
$\mid \mathbf{case}\, E\, \mathbf{of}\, (\mathbf{inj}_1\, x_1 \to e_1, \mathbf{inj}_2\, x_2 \to e_2)$	$\mid \mathbf{case}\, E\, \mathbf{of}\, (\mathbf{inj}_1\, x_1 \to e_1, \mathbf{inj}_2\, x_2 \to e_2)$
$\mid \mathbf{inj}_i\, E$	

Fig. 4. CBV and CBN for linear $\lambda\mu$: let reduction, values, and evaluation contexts

this, the continuation variable α should be replaced everywhere by the actual continuation $\square\, t$. We write this substitution $c\{E/\alpha\}$, with the defining clause as

$$([\beta]t)\{E/\alpha\} = \begin{cases} E[t] & \text{if } \alpha = \beta \\ [\beta]t\{E/\alpha\} & \text{otherwise} \end{cases}$$

Note that in the first case, substitution does not continue inside of t because the use of continuation variables is linear.

Since this rule only applies to evaluation contexts that form a command, how do such contexts reduce when $E[\mu\alpha.c]$ is a term? First, the term undergoes η-expansion, and then μ-capturing applies to the new context $[\beta]E$:[3]

$$E[\mu\alpha.c] \to_\eta \mu\beta.[\beta](E[\mu\alpha.c]) \to \mu\beta.c\{[\beta]E/\alpha\}.$$

We define two classes of normal forms, t^N for terms and c^N for commands:

$$t^N ::= v \mid \mu\alpha.c^N \qquad\qquad c^N ::= [\beta]v$$

Theorem 2 (Progress and Preservation).

1. *If $\Gamma \vdash e : \Theta \mid \Pi$ then e is either one of t^N or c^N, or e can take a step.*
2. *If $\Gamma \vdash e : \Theta \mid \Pi$ and $e \to e'$, then $\Gamma \vdash e' : \Theta \mid \Pi$.*

4 Dualization and CP$^{\pm}$

On the surface, the programming style of CP seems alien to the programming style of $\lambda\mu$. To rephrase Bernardy et al. (2014), the $\lambda\mu$-calculus is still a λ-calculus; CP is not. To go even further, the type systems are not obviously equivalent; it is not clear how negation relates to the operators of CLL.

[3] The operational semantics we present here is quite a bit different from Selinger (2001) and Wadler (2005). Their $\mu - \beta$ rules are encompassed by our notion of μ-capturing. Their ξ rule corresponds to the procedure of μ-capturing inside a term described here.

Tensor logic (Melliès and Tabareau, 2010) provides insight into the relationship between a λ calculus and a one-sided judgment. Introduced in the setting of game theory, it also has applications in category theory (Melliès and Tabareau, 2010) and as a type theory for CPS (Paykin et al., 2015). Perhaps the defining feature of tensor logic is that it can be studied equally well from two perspectives. The two-sided perspective results in a logic of non-involutive negation, while the one-sided perspective results in a logic of non-invertible polarization.

In this section we will replicate the two perspectives of tensor logic in the setting of classical linear logic. The question we must answer is: how does the two-sided $\lambda\mu$ calculus relate to the one-sided CP?

4.1 Duality in $\lambda\mu$

De Morgan duality, written $(-)^\perp$, is the key to permuting types around the turnstile in a derivation. The canonical rules for duality are

$$\frac{\Gamma \vdash A, \Pi}{\Gamma, A^\perp \vdash \Pi} \qquad \frac{\Gamma, A \vdash \Pi}{\Gamma \vdash A^\perp, \Pi}$$

Duality is strictly involutive ($(A^\perp)^\perp = A$), which means that the above rules are also invertible. This flexibility allows us to shift perspective by freely moving hypotheses to different parts of the judgment. Therefore, any judgment $\Gamma \vdash \Pi$ in a two-sided logic can be viewed in a one-sided logic as $\cdot \vdash \Gamma^\perp, \Pi$.

In classical logics, De Morgan duality is a meta-operation on types. We saw this operation in CLL, where $(X \otimes Y)^\perp = X^\perp \parr Y^\perp$, for example. But what is duality in the $\lambda\mu$ calculus, where $\parr$ is not a type operator? We could try to encode duality as negation $\neg A$, except that negation is not *strictly* involutive.

And yet, for any type A of linear $\lambda\mu$, we may imagine its *syntactic* dual $A^\perp$, not as a meta-operation, but as an explicit constructor. If we respect the fact that $A^{\perp\perp}$ is syntactically equal to A, we find that there are two disjoint classes of propositions: negative propositions A^-, which are syntactic duals, and positive propositions A^+, which correspond to the original types.[4]

We assign names to the duals of various connectives to simplify their presentation. In the style of CLL, we write $A^\perp \parr B^\perp$ for $(A \otimes B)^\perp$ where A and B are positive types. Thus each connective gets attributed both a positive and a negative copy, as summarized in Figure 5. Seen another way, the positive and negative types are axiomatized as follows:

$$A^+ ::= 1 \mid {A_1}^+ \otimes {A_2}^+ \mid 0 \mid {A_1}^+ \oplus {A_2}^+$$
$$A^- ::= \bot \mid {A_1}^- \parr {A_2}^- \mid \top \mid {A_1}^- \mathbin{\&} {A_2}^-$$

[4] Drawing the connection to category theory, every object $A \in \mathcal{C}$ has a dual object $A^{\mathrm{op}} \in \mathcal{C}^{\mathrm{op}}$. Thus A and its dual live in distinct categories.

Operator	Positive Copy	Negative Copy
1	1	$\bot$
$\otimes$	$\otimes$	$⅋$
0	0	$\top$
$\oplus$	$\oplus$	$\&$
$\neg$	$\downarrow$	$\uparrow$

Fig. 5. Positive and negative copies of linear operators

Negation. To incorporate negation into this analysis, first consider that it is self-dual: $(\neg A)^\perp = \neg A^\perp$. Close examination tells us that these are really two distinct copies of negation, one positive and one negative.

However, this presentation of negation is incompatible with the one-sided sequent calculus, because negation is *contravariant*: it moves a proposition from one side of the judgment to the other. In the presence of syntactic duality, we could imagine negation being partitioned into two parts: the contravariant duality operator $(-)^\perp$, along with a covariant shift operator written $\downarrow$, which turns a negative proposition into a positive proposition. The left and right negation rules from the two-sided formulation might instead be written as follows, where $\neg A^+ = \downarrow(A^+)^\perp$:

$$\dfrac{\dfrac{\Gamma \vdash A, \Pi}{\Gamma, A^\perp \vdash \Pi}}{\Gamma, \downarrow A^\perp \vdash \Pi} \qquad \dfrac{\dfrac{\Gamma, A \vdash \Pi}{\Gamma \vdash A^\perp, \Pi}}{\Gamma \vdash \downarrow A^\perp, \Pi}$$

The shift operator $\downarrow$, as well as its dual copy $\uparrow$, explicitly change the polarity of their argument. These fit nicely into Figure 5, and they augment the syntax of types as follows:

$$A^+ ::= \cdots \mid \downarrow A^-$$
$$A^- ::= \cdots \mid \uparrow A^+$$

Thus the act of dualizing a logic with negation naturally generates polarized types, in the sense of Girard (1991). This relationship between negation and polarization is not new. It has been explored by Laurent and Regnier (2003) relating polarization procedures of classical logic to CPS translations using negation. Similarly, Laurent (2003) gave a semantics of the (non-linear) $\lambda\mu$-calculus in terms of polarized proof nets. Finally, Zeilberger (2008) takes negation to be a primitive operator in his polarized logic and observes that $\neg A^+$ is isomorphic to $\downarrow(A^+)^\perp$.

4.2 Polarizing CP

It remains to see how polarization can be incorporated into the type system of CP. This question has already been addressed, at least in the case of intuitionistic session types, by Pfenning and Griffith (2015). In their setting, channels

$$\frac{}{y \leftrightarrow x \vdash y : (A^+)^\perp; x : A^+}\ \text{AX} \qquad \frac{P_1 \vdash \Delta_1; \Pi_1, x : A \quad P_2 \vdash x : A^\perp, \Delta_2; \Pi_2}{\nu x.\langle P_1 \mid P_2\rangle \vdash \Delta_1, \Delta_2; \Pi_1, \Pi_2}\ \text{CUT}$$

$$\frac{P \vdash \Delta, x : A^-; \Pi}{\downarrow x.P \vdash \Delta; x : \downarrow A^-, \Pi}\ \downarrow \qquad \frac{P \vdash \Delta; x : A^+, \Pi}{\uparrow x.P \vdash \Delta, x : \uparrow A^+; \Pi}\ \uparrow$$

$$\frac{}{x[].0 \vdash \cdot; x : 1}\ 1 \qquad \frac{P \vdash \Delta; \Pi}{x().P \vdash x : \perp, \Delta; \Pi}\ \perp$$

$$\frac{P_1 \vdash \Delta_1; \Pi_1, y : A^+ \quad P_2 \vdash \Delta_2; \Pi_2, x : B^+}{x[y].(P_1 \mid P_2) \vdash \Delta_1, \Delta_2; \Pi_1, \Pi_2, x : A^+ \otimes B^+}\ \otimes \qquad \frac{P \vdash y : A^-, x : B^-, \Delta; \Pi}{x(y).P \vdash x : A^- \parr B^-, \Delta; \Pi}\ \parr$$

$$\text{(no 0 rule)} \qquad \frac{}{x.\mathbf{case}() \vdash x : \top, \Delta; \Pi}\ \top$$

$$\frac{P \vdash \Delta; \Pi, x : {A_i}^+}{x[\mathbf{inj}_i].P \vdash \Delta; \Pi, x : {A_1}^+ \oplus {A_2}^+}\ \oplus \qquad \frac{P_1 \vdash x : A^-, \Delta; \Pi \quad P_2 \vdash x : B^-, \Delta; \Pi}{x.\mathbf{case}(P_1 \mid P_2) \vdash x : A^- \,\&\, B^-, \Delta; \Pi}\ \&$$

Fig. 6. $\mathrm{CP}^{\pm}$

are directed, where positive channels are outgoing, and negative channels are incoming. Shift operations explicitly change the direction of the channel.

This intuition carries over naturally to the classical case. We define a term language $\mathrm{CP}^{\pm}$, pronounced "CP more or less," to be a polarized version of CP. The syntax of processes is identical to that of CP, with the addition of two actions for shifting the direction of a channel. The syntax of processes is as follows:

$$P := \cdots \mid \downarrow x.P \mid \uparrow x.P$$

The process $\downarrow x.P$ offers an output channel $x : \downarrow A^-$, which sends a special "shift" message over x, before reversing the direction of the channel and continuing with an input channel $x : A^-$. Similarly, $\uparrow x.Q$ waits to receive a shift message on x, before treating x as output in Q.

Judgments of $\mathrm{CP}^{\pm}$ have the form $P \vdash \Delta; \Pi$, where Δ consists of negative (incoming) channels, and P_i consists of positive (outgoing) channels. The typing rules are given in Figure 6.

The operational semantics of $\mathrm{CP}^{\pm}$ augments the semantics of CP with rules for the shift operators. The β and η rules are as follows:

$$\nu x.\langle \downarrow x.P_1 \mid \uparrow x.P_2\rangle \to_\beta \nu x.\langle P_2 \mid P_1\rangle$$
$$(y \leftrightarrow x \vdash y : \uparrow A^+; x : \Downarrow (A^+)^\perp) \to_\eta \downarrow x.\uparrow y.y \leftrightarrow x$$

The structural equivalences $\equiv_s$ in Equation (1) are no longer applicable in $\mathrm{CP}^{\pm}$. The reason is that only one of $\nu x.\langle P_1 \mid P_2\rangle$ and $\nu x.\langle P_2 \mid P_1\rangle$ is ever well-typed in $\mathrm{CP}^{\pm}$, and similarly for $x \leftrightarrow y$ and $y \leftrightarrow x$.

Action contexts in $\mathrm{CP}^{\pm}$ are extended with the shift operators.

$$C^a ::= \cdots \mid \downarrow x.\Box \mid \uparrow x.\Box$$

4.3 Evaluation Order

Taking the commuting conversion rules verbatim from CP results in a similarly nondeterministic reduction strategy for $\mathrm{CP}^{\pm}$. However, $\mathrm{CP}^{\pm}$ has the ability to express other reduction strategies. In particular, we can specify either a call-by-name and call-by-value evaluation strategy in a way similar to Wadler's dual calculus (2003).

To illustrate what we mean by CBV and CBN in this context, consider the following encoding of implication as the polarized type $\Downarrow(A^+ \multimap B^-)$. Application can be encoded as follows:

$$\dfrac{P \vdash x : \Downarrow(A^+ \multimap B^-) \qquad \dfrac{\dfrac{Q \vdash y : A^+ \qquad \overline{z \leftrightarrow x \vdash z : B^-; x : (B^-)^{\perp}}}{x[y].(Q \mid z \leftrightarrow x) \vdash z : B^-; x : A^+ \otimes (B^-)^{\perp}}}{\uparrow x.x[y].(Q \mid z \leftrightarrow x) \vdash z : B^-, x : \Uparrow(A^+ \otimes (B^-)^{\perp}); \cdot}}{\nu x.\langle P \mid \uparrow x.x[y].(Q \mid z \leftrightarrow x)\rangle \vdash z : B^-; \cdot}$$

The process P is the equivalent of a λ-abstraction if it has the form $\downarrow x.x(y).P'$ for $P' \vdash x : B^-, y : (A^+)^{\perp}$. In this case, the application reduces to $\nu y.\langle Q \mid P'\{z/x\}\rangle$.

It is here we can talk about evaluation order. A call-by-value evaluation order reduces the argument Q before continuing in the evaluation of the result $z : B^-$. We can achieve this by prioritizing commuting conversions on the right of a cut over conversions on the left. This can be done by refining the definition of evaluation contexts.

$$C^e := \nu x.\langle P \mid \Box\rangle \mid \nu x.\langle \Box \mid P^x\rangle$$

where P^x is any process starting with an action on x.

Similarly, a call-by-name evaluation order tries to compute $z : B^-$ before evaluating the argument Q, by prioritizing conversions on the left:

$$C^e := \nu x.\langle \Box \mid P\rangle \mid \nu x.\langle P^x \mid \Box\rangle.$$

5 Equational Correspondence

The dualization described in the previous section shows how typing judgments in the linear $\lambda\mu$-calculus relate to typing judgments in a CP-like language. But the structure of the typing judgments says nothing about the equivalence of the operational semantics of the two languages. The notion of *equational correspondence* (Sabry and Felleisen, 1993) makes this relationship formal.

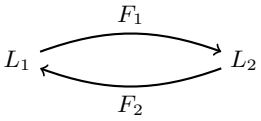
$$\frac{\Gamma \vdash t : A \mid \Pi}{t^{(x)} \vdash \Gamma^{\perp}; \Pi, x : A} \qquad \frac{\Gamma \vdash c \mid \Pi}{c^{*} \vdash \Gamma^{\perp}; \Pi} \qquad \frac{P \vdash \Delta; \Pi}{\Delta^{\perp} \vdash (P)_{*} \mid \Pi}$$

$$\begin{aligned}
y^{(x)} &:= y \leftrightarrow x \\
(\mathbf{let}\ y = t\ \mathbf{in}\ e)^{(x)} &:= \nu y.\langle t^{(y)} \mid e^{(x)} \rangle \\
(\mu\alpha.c)^{(x)} &:= (c^{*})\{x/\alpha\} \\
([\alpha]t)^{*} &:= (t^{(x)})\{\alpha/x\} \\
(\overline{\lambda}x.c)^{(x)} &:= \downarrow x.c^{*} \\
(t_1\ t_2)^{*} &:= \nu x.\langle {t_1}^{(x)} \mid \uparrow x.{t_2}^{(x)} \rangle \\
()^{(x)} &:= x[].0 \\
(\mathbf{let}\ () = t\ \mathbf{in}\ e)^{(x)} &:= \nu y.\langle t^{(y)} \mid y().e^{(x)} \rangle \\
(t_1, t_2)^{(x)} &:= x[x'].({t_1}^{(x')} \mid {t_2}^{(x)}) \\
(\mathbf{let}\ (y', y) = t\ \mathbf{in}\ e)^{(x)} &:= \nu y.\langle t^{(y)} \mid y(y').e^{(x)} \rangle \\
(\mathbf{case}\ t\ \mathbf{of}\ ())^{(x)} &:= \nu y.\langle t^{(y)} \mid y.\mathbf{case}() \rangle \\
(\mathbf{inj}_i\ t)^{(x)} &:= x[\mathbf{inj}_i].t^{(x)} \\
(\mathbf{case}\ t\ \mathbf{of}\ (\mathbf{inj}_1\ y \to e_1, \mathbf{inj}_2\ y \to e_2))^{(x)} &:= \\
\nu y.\langle t^{(y)} \mid y.\mathbf{case}({e_1}^{(x)} \mid {e_2}^{(x)}) \rangle &
\end{aligned}$$

$$\begin{aligned}
(y \leftrightarrow x)_{*} &:= [x]y \\
(\nu y.\langle P_1 \mid P_2 \rangle)_{*} &:= \mathbf{let}\ y = (P_1)_{*}\ \mathbf{in}\ (P_2)_{*} \\
(\downarrow x.P)_{*} &:= [x]\overline{\lambda}x.(P)_{*} \\
(\uparrow y.P)_{*} &:= y\,((P)_{(y)}) \\
(x[].0)_{*} &:= [x]() \\
(y().P)_{*} &:= \mathbf{let}\ () = y\ \mathbf{in}\ (P)_{*} \\
(x[y].(P_1 \mid P_2))_{*} &:= [x]((P_1)_{*}, (P_2)_{*}) \\
(y(y').P)_{*} &:= \mathbf{let}\ (y', y) = y\ \mathbf{in}\ (P)_{*} \\
(y.\mathbf{case}())_{*} &:= \mathbf{case}\ y\ \mathbf{of}\ () \\
(x[\mathbf{inj}_i].P)_{*} &:= [x]\mathbf{inj}_i\,(P)_{*} \\
(y.\mathbf{case}(P_1 \mid P_2))_{*} &:= \\
\mathbf{case}\ y\ \mathbf{of}\ (\mathbf{inj}_1\ y \to (P_1)_{*}, \mathbf{inj}_2\ y \to (P_2)_{*}) &
\end{aligned}$$

Fig. 7. Equational correspondence between linear $\lambda\mu$ and $\mathrm{CP}^{\pm}$

Definition 3. *Let L_1 and L_2 be term languages with equality, written $t_1 =_1 t_1'$ and $t_2 =_2 t_2'$. Suppose that there exist two maps F_1 and F_2 as shown below. These maps form an* equational correspondence *if the following conditions hold:*



1. *If $t_1 =_1 t_1'$ then $F_1\,(t_1) =_2 F_1\,(t_1')$.*
2. *If $t_2 =_2 t_2'$ then $F_2\,(t_2) =_1 F_2\,(t_2')$.*
3. *For all terms $t_1 \in L_1$, $t_1 =_1 F_2\,(F_1\,(t_1))$.*
4. *For all terms $t_2 \in L_2$, $t_2 =_2 F_1\,(F_2\,(t_2))$.*

We thus come to our main theorem:

Theorem 4. *Linear $\lambda\mu$ equationally corresponds with* $\mathrm{CP}^{\pm}$.

Proof (Sketch).

We start by defining translations, shown in Figure 7, between the two languages. First, for $\Gamma \vdash e : \Theta \mid \Pi$, we define $e^{(x)}$ to be a process so that $e^{(x)} \vdash \Gamma^{\perp}; \Pi, x : \Theta$. When e is a command c we write c^{*}.

Next, for $P \vdash \Delta; \Pi$, we define $(P)_{*}$ to be a command so that $\Delta^{\perp} \vdash (P)_{*} \mid \Pi$. The classes of term and continuation variables are collapsed in $\mathrm{CP}^{\pm}$, so we move between them freely in the translations.

The remainder of the proof must show that the four conditions in the definition of equational correspondence hold. But condition (4), which says that all $\lambda\mu$ expressions e satisfy $e = (e^{(\alpha)})_*$, is not quite right for terms, because every $(P)_*$ is a command. Instead we amend the condition to say that for every term t, we have $t = \mu\alpha.(t^{(\alpha)})_*$. □

6 Discussion

6.1 Comparing Classical Type Systems

Unlike the simply-typed λ-calculus for intuitionistic logic, there is no canonical type system for classical logic. Instead, there are a number of candidate type systems corresponding to different well-known formulations of classical logic.

- Classical logic can be formulated as intuitionistic logic plus the property of double negation elimination. The $\lambda_{\mathcal{C}}$-calculus (Griffin, 1990) is a typed λ-calculus with the addition of the control operator $\mathcal{C}$, of type $((A \to \bot) \to \bot) \to A$.
- Classical logic can alternatively be formulated as intuitionistic logic with multiple conclusions, which corresponds to the structure of the $\lambda\mu$-calculus (Parigot, 1992).
- The logic may be in natural deduction or sequent calculus form. Curien and Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus (2000), which is also called System L (Spiwack, 2014), and Wadler's dual calculus (2003) are variations on equivalent sequent calculus formulations of $\lambda\mu$.

These systems are all related by equational correspondences—de Groote (1994) proves the correspondence between $\lambda\mu$ and $\lambda_{\mathcal{C}}$, and Wadler (2005) proves the correspondence between $\lambda\mu$ and the dual calculus. Our work continues in this tradition by indicating that the linear versions of these systems (for example, Spiwack's Linear L, 2014) are also equivalent to $\mathrm{CP}^{\pm}$.

6.2 A Syntax for Tensor Logic

Melliès and Tabareau introduce tensor logic in the setting of game-theoretic semantics for linear logic. While they study the denotational semantics of the logic extensively and provide translations from CLL, Melliès and Tabareau do not give proof terms for either of the two formulations—two-sided or one-sided.

However, proof terms are not hard to imagine in the style of $\lambda\mu$ and $\mathrm{CP}^{\pm}$. The two-sided perspective types a (strict) sublanguage of linear $\lambda\mu$ with negation but without the μ operator. The result is a simply-typed λ-calculus where the return type of every function is $\bot$, as described by Lafont et al. (1993). The one-sided perspective of tensor logic can similarly type a (strict) sublanguage of $\mathrm{CP}^{\pm}$, where each process offers at most one output channel. This formulation is equivalent to Spiwack's Polarized L (2014), and corresponds to a *focused* polarized logic, as opposed to $\mathrm{CP}^{\pm}$, which is unfocused.

6.3 Session Types and Linear Logic

Linear logic has long been explored as a type system for concurrent programming, especially process calculi in the style of the π-calculus (Bellin and Scott, 1994; Beffara, 2006; Kobayashi et al., 1999). Caires and Pfenning (2010) present an elegant interpretation of an (intuitionistic) linearly-typed fragment of the π-calculus by interpreting linear propositions as session types (Honda, 1993; Honda et al., 1998) in a calculus they call π-DILL. The connections between session types and linear logic have been studied in the past (Mazurak and Zdancewic, 2010; Gay and Vasconcelos, 2010) but had not been formally compared. Wadler (2014) makes the connection formal by exhibiting a translation between CP, a classical variant of π-DILL, and GV, a variation of the session-typed calculus of Gay and Vasconcelos (2010). Lindley and Morris (2014, 2015) expand this translation to a bisimulation between CP and GV.

6.4 Variations on CP

The process calculus described in Section 2 varies from Wadler's original formulation of CP in a few significant ways. The first is that Wadler's processes only communicate at the top level, never under prefix actions. This means that normal forms consist of any processes that do not begin with a cut. Following Pérez et al. (2014), we allow communication to take place at any point in a process, even if another channel is waiting to synchronize. For example, in the process

$$x().\nu y.\langle y[].0 \mid y().P\rangle,$$

communication takes place over the channel y even though x sends a message first. Operationally, the difference is the exclusion of the structural rule in Equation (2). Wadler's equational theory is thus finer-grained than ours, and does not reach the level of behavioral equivalence.

In the same vein, Wadler does not equate the processes defined by our third class of commuting conversions. Following Bellin and Scott (1994) and Pérez et al. (2014), behavioral equivalence means that the order of non-interfering prefix actions is unobservable, and should be equated.

The treatment of η-expansions for link forwarding is another departure from Wadler's presentation. Like Wadler, we allow link forwarding at arbitrary type for the sake of expressibility. However, just as in the λ-calculus, η-expansion reduces the allowable normal forms by equating behaviorally equivalent processes.

The operational semantics we define for CP in Section 2 extends naturally to $\mathrm{CP}^{\pm}$. It is worth noting, however, that the equational correspondence proved in Section 5 relies on the second class of commuting conversions (CUT-CUT) but not the third (ACTION-ACTION). Therefore it would still be applicable using a semantics closer to Wadler's original presentation (2014).

Finally, Wadler's presentation of CP includes the linear exponentials ! and ?, as well as polymorphism $\forall$ and $\exists$. In this paper we work with the linear "core" of CP, excluding the exponentials for the sake of clarity. To incorporate polymorphism into our presentation, we would need to extend types with type variables.

In that case, communication would eliminate all cuts, but not necessarily links at atomic type.

6.5 How Much is "More or Less"?

In the introduction we made the claim that linear $\lambda\mu$ was "more or less" CP, and we made the statement precise by exhibiting $\text{CP}^{\pm}$. However, we haven't addressed exactly how much $\text{CP}^{\pm}$ is "more or less" CP.

We can easily define an erasure operation $|-|$ on both types and processes that removes all shift operations. Trivially, if $P \vdash \Delta; \Pi$ then $|P| \vdash |\Delta|, |\Pi|$. As Zeilberger (2008) shows for a different polarized term syntax, the completeness of such an erasure can be summarized as follows: If $Q \vdash |\Delta|, |\Pi|$ in CP, then there exists a $\text{CP}^{\pm}$ process P such that $P \vdash \Delta; \Pi$ and $|P| = Q$. In addition, we would require that this "expansion" of Q respects equalities in a similar way to equational correspondences. However, even describing the necessary relationship in detail is beyond the scope of this work.

Meanwhile, $\text{CP}^{\pm}$ offers operational behaviors not expressible in CP. In particular, $\text{CP}^{\pm}$ has the ability to set a particular reduction strategy such as CBV, CBN, or something in between. Once an evaluation strategy is fixed, the types dictate the evaluation of any particular process. The programmer thus has full control over scheduling in $\text{CP}^{\pm}$, while in CP the programmer has no control.

Acknowledgments Thanks to Neel Krishnaswami, Sam Lindley, and the paper's reviewers for their valuable insights. We would also like to thank Phil Wadler for inspiring a fascination with the Curry-Howard correspondence. This material is based in part upon work supported by the NSF Graduate Research Fellowship under Grant No. DGE-1321851 and by NSF Grant No. CCF-1421193.

Bibliography

Beffara, E.: A concurrent model for linear logic. Electronic Notes in Theoretical Computer Science 155, 147–168 (May 2006)

Bellin, G., Scott, P.: On the π-calculus and linear logic. Theoretical Computer Science 135(1), 11–65 (1994)

Bernardy, J.P., Rosn, D., Smallbone, N.: Compiling linear logic using continuations (2014)

Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010 - Concurrency Theory, Lecture Notes in Computer Science, vol. 6269, pp. 222–236. Springer Berlin Heidelberg (2010)

Curien, P.L., Herbelin, H.: The duality of computation. ACM SIGPLAN Notices 35(9), 233–243 (Sept 2000)

Felleisen, M.: The theory and practice of first-class prompts. In: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 180–190. POPL '88, ACM, New York, NY, USA (1988)

Gay, S.J., Vasconcelos, V.T.: Linear type theory for asynchronous session types. Journal of Functional Programming 20, 19–50 (2010)

Girard, J.Y.: Linear logic. Theoretical Computer Science 50(1), 1–101 (1987)

Girard, J.Y.: A new constructive logic: classic logic. Mathematical Structures in Computer Science 1, 255–296 (Nov 1991)

Griffin, T.G.: A formulae-as-type notion of control. In: Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 47–58. POPL '90, ACM, New York, NY, USA (1990)

de Groote, P.: On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In: Pfenning, F. (ed.) Logic Programming and Automated Reasoning, Lecture Notes in Computer Science, vol. 822, pp. 31–43. Springer Berlin Heidelberg (1994)

Honda, K.: Types for dyadic interaction. In: CONCUR'93, pp. 509–523 (1993)

Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Programming Languages and Systems, pp. 122–138 (1998)

Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the π-calculus. ACM Transactions on Programming Languages and Systems 21(5), 914–947 (sep 1999)

Lafont, Y., Reus, B., Streicher, T.: Continuation semantics or expressing implication by negation. Tech. Rep. 9321, Ludwig-Maximilians-Universitt (1993)

Laurent, O.: Polarized proof-nets and $\lambda\mu$-calculus. Theoretical Computer Science 290(1), 161 – 188 (2003)

Laurent, O., Regnier, L.: About translations of classical logic into polarized linear logic. In: Logic in Computer Science. pp. 11–20. IEEE (2003)

Lindley, S., Morris, J.G.: Sessions as propositions. Electronic Proceedings of Theoretical Computer Science 155, 9–16 (Jun 2014)

Lindley, S., Morris, J.G.: A semantics for propositions as sessions. In: Programming Languages and Systems, pp. 560–584 (2015)
Mazurak, K., Zdancewic, S.: Lolliproc: To concurrency from classical linear logic via Curry-Howard and control. SIGPLAN Not. 45(9), 39–50 (Sept 2010)
Melliès, P.A., Tabareau, N.: Resource modalities in tensor logic. Annals of Pure and Applied Logic 161(5), 632–653 (2010), the Third Workshop on Games for Logic and Programming Languages (GaLoP), 5-6 April 2008
Milner, R.: Functions as processes. Mathematical Structures in Computer Science 2, 119–141 (Jun 1992)
Munch-Maccagnoni, G.: Focalisation and classical realisability (version with appendices). In: Gradel, E., Kahle, R. (eds.) 18th EACSL Annual Conference on Computer Science Logic - CSL 09, Lecture Notes in Computer Science, vol. 5771, pp. 409–423. Springer-Verlag (Sept 2009)
Parigot, M.: $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.) Logic Programming and Automated Reasoning, Lecture Notes in Computer Science, vol. 624, pp. 190–201. Springer Berlin Heidelberg (1992)
Paykin, J., Zdancewic, S., Krishnaswami, N.R.: Linear temporal type theory for event-based reactive programming (2015)
Pérez, J.A., Caires, L., Pfenning, F., Toninho, B.: Linear logical relations and observational equivalences for session-based concurrency. Information and Computation 239, 254–302 (2014)
Pfenning, F., Griffith, D.: Polarized substructural session types. In: Pitts, A. (ed.) Foundations of Software Science and Computation Structures, Lecture Notes in Computer Science, vol. 9034, pp. 3–22. Springer Berlin Heidelberg (2015)
Sabry, A., Felleisen, M.: Reasoning about programs in continuation-passing style. LISP and Symbolic Computation 6(3–4), 289–360 (1993)
Selinger, P.: Control categories and duality: on the categorical semantics of the lambda-mu calculus. Mathematical Structures in Computer Science 11, 207–260 (4 2001), http://journals.cambridge.org/article_S096012950000311X
Spiwack, A.: A dissection of L (2014)
Wadler, P.: Call-by-value is dual to call-by-name. In: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming. pp. 189–201. ICFP '03, ACM, New York, NY, USA (2003)
Wadler, P.: Call-by-value is dual to call-by-name reloaded. In: Giesl, J. (ed.) Term Rewriting and Applications, Lecture Notes in Computer Science, vol. 3467, pp. 185–203. Springer Berlin Heidelberg (2005)
Wadler, P.: Propositions as sessions. SIGPLAN Not. 47(9), 273–286 (#sep# 2012)
Wadler, P.: Propositions as sessions. Journal of Functional Programming 24, 384–418 (2014)
Zeilberger, N.: On the unity of duality. Annals of Pure and Applied Logic 153(13), 66 – 96 (2008), special Issue: Classical Logic and Computation (2006)