Technical Report

UCAM-CL-TR-365
ISSN 1476-2986

Number 365

Computational types from a logical perspective I

P.N. Benton, G.M. Bierman, V.C.V. de Paiva

May 1995

ISSN 1476-2986

Computational Types from a Logical Perspective I

P.N. Benton* G.M. Bierman† V.C.V. de Paiva‡

Computer Laboratory
University of Cambridge
New Museums Site
Cambridge CB2 3QG
U.K.

Abstract

Moggi's computational lambda calculus is a metalanguage for denotational semantics which arose from the observation that many different notions of computation have the categorical structure of a strong monad on a cartesian closed category. In this paper we show that the computational lambda calculus also arises naturally as the term calculus corresponding (by the Curry-Howard correspondence) to a novel intuitionistic modal propositional logic. We give natural deduction, sequent calculus and Hilbert-style presentations of this logic and prove a strong normalisation result.

1 Introduction

The computational lambda calculus was introduced by Moggi as a metalanguage for denotational semantics which more faithfully models real programming language features such as non-termination, differing evaluation strategies, non-determinism and side-effects than does the ordinary simply typed lambda calculus [17, 18]. The starting point for Moggi's work is an explicit semantic distinction between *computations* and *values*. If A is an object which interprets the values of a particular type, then $T(A)$ is the object which models computations of that type A. For example, to model non-termination we might take A to be some complete partial order (cpo) and $T(A)$ to be the lifted cpo $A_\perp$.

For a wide variety of notions of computation, the unary operation $T(\cdot)$ turns out to have the categorical structure of a *strong monad* on an underlying cartesian closed category of values. This observation, which was also made by Spivey in the special case of computations which can raise exceptions [22], suggests a more unified and abstract view of programming languages. Having unearthed this common structure, we hope firstly to be able to design general purpose metalanguages and logics for reasoning about a range of programming language features and secondly to be able to modularise the semantics of complicated languages by studying the ways in which different monads can be combined. Work along these lines has been done by Crole [5] and Pitts [19]. The computational lambda calculus is the syntactic theory which expresses this semantic idea of notions of

*Supported by an EPSRC Research Fellowship and the EC Esprit BRA 8130 LOMAPS.

†Supported by an EPSRC Research Fellowship.

‡Supported by EPSRC grant GR/G53279.

computation as monads – it corresponds to cartesian closed categories with strong monads in just the same way that the simply typed lambda calculus with products corresponds to cartesian closed categories.

Whilst Moggi's work was initially aimed at structuring the semantics of programming languages, it has also (by a rather pleasing interplay of theory and practice) had a considerable impact on the pragmatics of writing functional programs. Wadler and others have shown that monads provide an elegant way to structure functional programs which perform naturally imperative operations, such as dealing with updatable state or engaging in interactive input/output [24, 25, 12].

This paper looks at (an extension of) Moggi's computational lambda calculus from a logical perspective. Using the Curry-Howard correspondence 'the other way round' we derive a logic which we term *CL-logic.* This consists of (propositional) intuitionistic logic plus an S4-style modal *possibility* operator $\Diamond$ corresponding to the computation type constructor. On a purely intuitive level, and particularly if one thinks about non-termination, there is certainly something appealing about the idea that a computation of type A represents the possibility of a value of type A.

CL-logic is interesting in its own right, and appears to have been discovered independently at least three times. Soon after completing an early draft of this work, we found that Curry had briefly considered just such a system in the late 50s [6]. More recently, and quite independently of Moggi's work, Fairtlough and Mendler have come up with sequent calculus and Hilbert-style presentations of CL-logic in the context of hardware verification [9].

2 Computational Lambda Calculus

The computational lambda calculus, which Moggi refers to as $\lambda\mathrm{ML}_T$, is a typed lambda calculus whose types are closed under terminal object, binary products, function spaces and the computation type constructor T. For the purposes of this paper, we will consider immediately a slight extension which also includes coproduct types. The natural deduction typing rules for this version of $\lambda\mathrm{ML}_T$ are shown in Figure 1.

Intuitively, if e is a value then $\mathsf{val}(e)$ is the trivial computation that immediately evaluates to e. The let construct allows a computation to be evaluated to a value within the context of another computation: $(\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f)$ denotes the computation which first evaluates e to some value $c : A$ and then proceeds to evaluate $f[c/x]$.

The equational theory of $\lambda\mathrm{ML}_T$ comprises the usual $\beta\eta$ equalities of the simply typed lambda calculus with coproducts, together with the following three extra axioms:

$$\begin{aligned}
\mathsf{let}\, x \Leftarrow (\mathsf{val}(e))\, \mathsf{in}\, f &= f[e/x] && (1)\\
\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, (\mathsf{val}(x)) &= e && (2)\\
\mathsf{let}\, x' \Leftarrow (\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f)\, \mathsf{in}\, g &= \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, (\mathsf{let}\, x' \Leftarrow f\, \mathsf{in}\, g) && (3)
\end{aligned}$$

Here are some examples of different notions of computation, all of which fit this general scheme:

Non-Determinism. Take $T(A) \stackrel{\text{def}}{=} \mathbb{P}(A)$ with

$$\begin{aligned}
\mathsf{val}(e) &\stackrel{\text{def}}{=} \{e\}\\
(\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f) &\stackrel{\text{def}}{=} \bigcup_{x \in e} f.
\end{aligned}$$

$$\frac{}{\Gamma, x{:}\, A \vdash x{:}\, A} \qquad \frac{}{\Gamma \vdash *{:}\,1}$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.e : A \to B} \qquad \frac{\Gamma \vdash e : A \to B \qquad \Gamma \vdash f : A}{\Gamma \vdash e\,f : B}$$

$$\frac{\Gamma \vdash e : A \qquad \Gamma \vdash f : B}{\Gamma \vdash (e, f) : A \times B} \qquad \frac{\Gamma \vdash e : A \times B}{\Gamma \vdash \mathsf{fst}(e) : A} \quad \frac{\Gamma \vdash e : A \times B}{\Gamma \vdash \mathsf{snd}(e) : B}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \mathsf{inl}(e) : A + B} \qquad \frac{\Gamma \vdash e : B}{\Gamma \vdash \mathsf{inr}(e) : A + B}$$

$$\frac{\Gamma \vdash e : A + B \qquad \Gamma, x : A \vdash f : C \qquad \Gamma, y : B \vdash g : C}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \to f \mid \mathsf{inr}(y) \to g : C}$$

$$\frac{\Gamma \vdash e : TA \qquad \Gamma, x : A \vdash f : TB}{\Gamma \vdash \mathsf{let}\ x \Leftarrow e\ \mathsf{in}\ f : TB}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \mathsf{val}(e) : TA}$$

Figure 1: Natural Deduction Presentation of the Computational Lambda Calculus

Exceptions. Take $T(A) \stackrel{\text{def}}{=} 1 + A$ with

$$\begin{aligned} \mathsf{val}(e) &\stackrel{\text{def}}{=} \mathsf{inr}(e) \\ (\mathsf{let}\ x \Leftarrow e\ \mathsf{in}\ f) &\stackrel{\text{def}}{=} \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(\,*\,) \to \mathsf{inl}(\,*\,) \mid \mathsf{inr}(x) \to f. \end{aligned}$$

Continuations. Take $T(A) \stackrel{\text{def}}{=} (A \to R) \to R$ with

$$\begin{aligned} \mathsf{val}(e) &\stackrel{\text{def}}{=} \lambda k{:}\, A \to R.k\ e \\ (\mathsf{let}\ x \Leftarrow e\ \mathsf{in}\ f) &\stackrel{\text{def}}{=} \lambda k{:}\, B \to R.e\ (\lambda x{:}\, A.f\ k). \end{aligned}$$

In each case, not only do the constructs have the right types, but the three equations above are also easily seen to hold.

A simple fact about $\lambda\mathrm{ML}_T$, which we shall use later, is that substitution is well-typed:

Lemma 1 (Substitution) *If* $\Gamma \vdash e : A$ *and* $\Delta, x : A \vdash f : B$ *then* $\Gamma, \Delta \vdash f[e/x] : B$. □

3 Propositional CL-Logic

In this section we use the Curry-Howard correspondence to derive a logic from Moggi's original presentation of $\lambda\mathrm{ML}_T$. We shall also consider sequent calculus and axiomatic formulations of the same logic.

3.1 Natural deduction formulation of CL-logic

Using the Curry-Howard correspondence [15], we can simply take Moggi's original presentation (given in Figure 1) and erase the terms to produce a logic. Each type constructor corresponds to a logical connective as follows:

Constructor	Connective
1	$\top$
$\times$	$\wedge$
$\rightarrow$	$\supset$
$+$	$\vee$
T	$\Diamond$

Hence we derive the logic, called propositional CL-logic, given in 'tree-form' in Figure 2 and in sequent form in Figure 3.

$$\dfrac{\begin{matrix}[A^x]\\ \vdots\\ B\end{matrix}}{A \supset B}\,(\supset_{\mathcal{I}})_x \qquad \dfrac{\begin{matrix}\vdots\\ A \supset B\end{matrix} \quad \begin{matrix}\vdots\\ A\end{matrix}}{B}\,(\supset_{\mathcal{E}})$$

$$\dfrac{\begin{matrix}\vdots\\ A\end{matrix} \quad \begin{matrix}\vdots\\ B\end{matrix}}{A \wedge B}\,(\wedge_{\mathcal{I}}) \qquad \dfrac{\begin{matrix}\vdots\\ A \wedge B\end{matrix}}{A}\,(\wedge_{\mathcal{E}}) \quad \dfrac{\begin{matrix}\vdots\\ A \wedge B\end{matrix}}{B}\,(\wedge_{\mathcal{E}})$$

$$\dfrac{\begin{matrix}\vdots\\ A\end{matrix}}{A \vee B}\,(\vee_{\mathcal{I}}) \quad \dfrac{\begin{matrix}\vdots\\ B\end{matrix}}{A \vee B}\,(\vee_{\mathcal{I}}) \qquad \dfrac{\begin{matrix}\vdots\\ A \vee B\end{matrix} \quad \begin{matrix}[A^x]\\ \vdots\\ C\end{matrix} \quad \begin{matrix}[B^y]\\ \vdots\\ C\end{matrix}}{C}\,(\vee_{\mathcal{E}})_{xy}$$

$$\dfrac{\begin{matrix}\vdots\\ A\end{matrix}}{\Diamond A}\,(\Diamond_{\mathcal{I}}) \qquad \dfrac{\begin{matrix}\vdots\\ \Diamond A\end{matrix} \quad \begin{matrix}[A^x]\\ \vdots\\ \Diamond B\end{matrix}}{\Diamond B}\,(\Diamond_{\mathcal{E}})_x$$

Figure 2: Tree-Style Natural Deduction Formulation of CL-logic

3.2 Sequent calculus for CL-logic

We can use the well-known correspondence between natural deduction and sequent calculus proof systems, to systematically derive a sequent calculus formulation of CL-logic. This is given in Figure 4.

Somewhat surprisingly, after completing this research, we discovered that this kind of possibility modality had been considered thirty-five years ago by Curry [6]. However, Curry was dismissive of this formulation:

$$\frac{}{\Gamma, A \vdash A}\ \textit{Identity} \qquad \frac{}{\Gamma \vdash \top}\ (\top_{\mathcal{I}})$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ (\supset_{\mathcal{I}}) \qquad \frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ (\supset_{\mathcal{E}})$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (\wedge_{\mathcal{I}}) \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ (\wedge_{\mathcal{E}}) \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ (\wedge_{\mathcal{E}})$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (\vee_{\mathcal{I}}) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (\vee_{\mathcal{I}})$$

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ (\vee_{\mathcal{E}})$$

$$\frac{\Gamma \vdash \Diamond A \qquad \Gamma, A \vdash \Diamond B}{\Gamma \vdash \Diamond B}\ (\Diamond_{\mathcal{E}})$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \Diamond A}\ (\Diamond_{\mathcal{I}})$$

Figure 3: Natural Deduction Formulation of CL-logic

> "The referee has pointed out that for certain kinds of modality it [the introduction rule for $\Diamond$] is not acceptable even then, because it allows the proof of
>
> $$\Diamond A, \Diamond B \vdash \Diamond(A \wedge B)$$
>
> He has proposed a theory of possibility more strictly dual to that of necessity. Although this theory looks promising it will not be developed here".

It is easy to see that

$$\Diamond A, \Diamond B \vdash \Diamond(A \wedge B)$$

(which is more typical of necessity modalities) *is* provable in our logic:

$$\cfrac{\cfrac{\cfrac{\cfrac{A, B \vdash A \qquad A, B \vdash B}{A, B \vdash A \wedge B}\ (\wedge_{\mathcal{R}})}{A, B \vdash \Diamond(A \wedge B)}\ (\Diamond_{\mathcal{R}})}{A, \Diamond B \vdash \Diamond(A \wedge B)}\ (\Diamond_{\mathcal{L}})}{\Diamond A, \Diamond B \vdash \Diamond(A \wedge B)}\ (\Diamond_{\mathcal{L}})$$

If $\Diamond A$ is to be interpreted as "A is possible" and if we have negation in our logic, we will have the following "paradox": $\Diamond A \wedge \Diamond \neg A \vdash \Diamond(A \wedge \neg A)$ (the antecedent might be true while the consequent appears always to be false!). Clearly this theorem is undesirable in a logic trying to capture the general notion of possibility, but whilst our choice of notation is therefore questionable, the logic is certainly consistent.

$$\frac{}{\Gamma, A \vdash A}\ \mathit{Identity} \qquad \frac{}{\Gamma \vdash \top}\ (\top_{\mathcal{R}})$$

$$\frac{\Gamma \vdash B \qquad B, \Gamma \vdash C}{\Gamma \vdash C}\ \mathit{Cut}$$

$$\frac{\Gamma, A \vdash C}{\Gamma, A \wedge B \vdash C}\ (\wedge_{\mathcal{L}}) \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \wedge B \vdash C}\ (\wedge_{\mathcal{L}}) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (\wedge_{\mathcal{R}})$$

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}\ (\vee_{\mathcal{L}}) \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (\vee_{\mathcal{R}}) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (\vee_{\mathcal{R}})$$

$$\frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \supset B \vdash C}\ (\supset_{\mathcal{L}}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ (\supset_{\mathcal{R}})$$

$$\frac{\Gamma, A \vdash \Diamond B}{\Gamma, \Diamond A \vdash \Diamond B}\ \Diamond_{\mathcal{L}} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \Diamond A}\ (\Diamond_{\mathcal{R}})$$

Figure 4: Sequent Calculus for CL-logic

3.3 Hilbert system for CL-logic

To complete our logical analysis of CL-logic, in this subsection we shall give a Hilbert-style presentation. It is the norm to present modal logics using such a system (for example, see Goré's thesis [13]) and certainly the unusual set of axioms which we shall propose can be seen as further evidence for its peculiarity as a modal logic.

Clearly the non-modal rules are characterized by any set of axioms for ordinary intuitionistic logic. We shall take those proposed by Van Dalen [7]. For the two modal rules, we shall follow the technique given by Hodges [14] for deriving axioms. Essentially, for those logical rules which have no side conditions or restrictions, the so-called *pure* logical rules, the axioms can be "read off" from the natural deduction formulation. Recalling the two modal rules from Figure 3:

$$\frac{\Gamma \vdash A}{\Gamma \vdash \Diamond A}\ (\Diamond_{\mathcal{I}}) \qquad \frac{\Gamma \vdash \Diamond A \qquad \Gamma, A \vdash \Diamond B}{\Gamma \vdash \Diamond B}\ (\Diamond_{\mathcal{E}})$$

we see that both these rules are pure.[1] The introduction rule suggests an axiom of the form $A \supset \Diamond A$ whereas the elimination rule suggests an axiom $\Diamond A \supset ((A \supset \Diamond B) \supset \Diamond B)$. We give the Hilbert system in Figure 5.

We see at once that our system enjoys a deduction theorem:

Proposition 2 (Deduction Theorem) *If* $\Gamma, A \vdash_H B$ *then* $\Gamma \vdash_H A \supset B$. □

It should be noted that Fairtlough and Mendler [9] have also (independently) proposed a Hilbert-style presentation of CL-logic (which they dub PLL, for *Propositional Lax Logic*),

[1]This should be compared to, for example, Intuitionistic **S4** [4].

Axioms

$A \supset A$
$A \supset B \supset A$
$(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$

$A \supset (B \supset A \wedge B)$
$A \wedge B \supset A$
$A \wedge B \supset B$

$A \supset A \vee B$
$B \supset A \vee B$
$(A \supset C) \wedge (B \supset C) \supset (A \vee B \supset C)$

$A \supset \Diamond A$
$\Diamond A \supset ((A \supset \Diamond B) \supset \Diamond B)$

Rules

$$\frac{}{\Gamma, A \vdash A}\ \textit{Identity}$$

$$\frac{}{\Gamma \vdash A}\ \textit{Axiom}\ (A \text{ one of the axioms above})$$

$$\frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \textit{Modus Ponens}$$

Figure 5: Hilbert System for CL-logic

although they give three axioms for the modality, which they write as $\bigcirc$ since it has aspects of both possibility and necessity:

$$\begin{array}{rcl} R & : & A \supset \bigcirc A \\ M & : & \bigcirc\bigcirc A \supset \bigcirc A \\ F & : & (A \supset B) \supset (\bigcirc A \supset \bigcirc B) \end{array}$$

This alternative axiomatisation of the logic is equivalent to that which we have given in Figure 5 in that the axioms of each system are theorems in the other.

4 Proof Normalisation

We now turn to the question of how to normalise natural deduction proofs in CL-logic, that is, what equalities we want to have on deductions. The normalisation procedure will be described in terms of the logic as presented in Figure 2; each kind of normalisation step then induces a reduction rule on the corresponding terms. One of the advantages of our logical approach is that the three equalities on $\lambda\mathrm{ML}_T$ terms which we gave earlier arise as natural proof-theoretic consequences of the process of normalisation (or cut-elimination) in CL-logic.

4.1 Principal Reductions

The basic kind of normalisation step on natural deduction proofs comes from removing 'detours' which arise when a logical connective is introduced and then immediately eliminated. We consider only the modality introduction/elimination pair as the others are standard (see, for example, [10]).

If we have $(\Diamond_{\mathcal{I}})$ followed by $(\Diamond_{\mathcal{E}})$ then the derivation looks like this:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ A\end{matrix}}{\Diamond A}(\Diamond_{\mathcal{I}}) \qquad \begin{matrix}[A]\\ \vdots\\ \Diamond B\end{matrix}}{\Diamond B}(\Diamond_{\mathcal{E}})$$

which normalises to

$$\begin{matrix}\vdots\\ [A]\\ \vdots\\ \Diamond B\end{matrix}$$

These principal reductions are known as β-rules, and a derivation which contains no possible applications of β-rules is said to be in β-normal form.

4.2 Commuting Conversions

Natural deduction systems can also give rise to a secondary form of normalisation step. These occur when the system contains elimination rules which have a minor premiss (Girard calls this a 'parasitic formula'). In general, when we have such a rule, we want to be able to commute the last rule in the derivation of the minor premiss down past the rule, or to move the application of a rule to the conclusion of the elimination up past the elimination rule into to the derivation of the minor premiss. The only important cases are moving eliminations up or introductions down. Such transformations are called *commuting conversions.* The restriction on the form of the conclusion of our $(\Diamond_{\mathcal{E}})$ rule (it must be modal) means that the rule gives rise to only one commuting conversion:

- A deduction of the form

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Diamond A\end{matrix} \qquad \begin{matrix}[A]\\ \vdots\\ \Diamond B\end{matrix}}{\Diamond B}(\Diamond_{\mathcal{E}}) \qquad \begin{matrix}[B]\\ \vdots\\ \Diamond C\end{matrix}}{\Diamond C}(\Diamond_{\mathcal{E}})$$

  commutes to

$$\dfrac{\begin{matrix}\vdots\\ \Diamond A\end{matrix} \qquad \dfrac{\begin{matrix}[A]\\ \vdots\\ \Diamond B\end{matrix} \qquad \begin{matrix}[B]\\ \vdots\\ \Diamond C\end{matrix}}{\Diamond C}(\Diamond_{\mathcal{E}})}{\Diamond C}(\Diamond_{\mathcal{E}})$$

Clearly we need also commuting conversions for the disjunctions, but these are standard and described in several places (e.g [11]). The only new case is the commutation of disjunction against the modality:

- The deduction

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ A\vee B\end{array}\qquad \begin{array}{c}[A]\\ \vdots\\ \Diamond C\end{array}\qquad \begin{array}{c}[B]\\ \vdots\\ \Diamond C\end{array}}{\Diamond C}\,(\vee_{\mathcal{E}})\qquad \begin{array}{c}[C]\\ \vdots\\ \Diamond D\end{array}}{\Diamond D}\,(\Diamond_{\mathcal{E}})$$

  commutes to

$$\dfrac{\begin{array}{c}\vdots\\ A\vee B\end{array}\qquad \dfrac{\begin{array}{c}[A]\\ \vdots\\ \Diamond C\end{array}\qquad \begin{array}{c}[C]\\ \vdots\\ \Diamond D\end{array}}{\Diamond D}\,(\Diamond_{\mathcal{E}})\qquad \dfrac{\begin{array}{c}[B]\\ \vdots\\ \Diamond C\end{array}\qquad \begin{array}{c}[C]\\ \vdots\\ \Diamond D\end{array}}{\Diamond D}\,(\Diamond_{\mathcal{E}})}{\Diamond D}\,(\vee_{\mathcal{E}})$$

We say that a derivation to which there are no possible applications of a commuting conversion is in c-normal form. A derivation to which neither a β-rule nor a commuting conversion is applicable is said to be in βc-normal form.

4.3 Reduction Rules for Terms

Both the principal reductions and the commuting conversions on derivations induce corresponding reduction steps on the terms of $\lambda\mathrm{ML}_T$ in the usual way (α-conversion is necessary to avoid variable capture in some commuting conversions as well as in β-reductions). These reduction rules on terms are shown in Figure 6. Note particularly that two of the three equations for $\lambda\mathrm{ML}_T$ which we listed on page 2 appear as reduction rules, and that these were essentially forced just by the shape of the introduction and elimination rules in the logic. The remaining equation is the η (uniqueness) rule for the computation type constructor, which we will discuss in Section 5.3.

Proposition 3 (Subject Reduction) *If* $\Gamma \vdash e : A$ *and* $e \rightarrow_{\beta c} e'$ *then* $\Gamma \vdash e' : A$

Proof. Induction and Lemma 1. □

4.4 Strong normalisation

In this section we examine the process of normalisation on natural deduction proofs in CL-logic (or, equivalently, the reduction process on terms of $\lambda\mathrm{ML}_T$), and show that it always terminates. This strong normalisation result is stronger than many others in that it applies to the full $\rightarrow_{\beta c}$ reduction relation, rather than just to the $\rightarrow_{\beta}$ relation. We will find it convenient to work with the term calculus $\lambda\mathrm{ML}_T$, rather than the logic, simply for reasons of space.

Strong normalisation proofs usually use variants of Tait's reducibility method [23]; the extension of Tait's method to commuting conversions as well as β-reductions is due to

$$
\begin{array}{lll}
(\lambda x.u)v & \rightarrow_{\beta} & u[v/x] \\[1ex]
\mathsf{fst}(u,v) & \rightarrow_{\beta} & u \\[1ex]
\mathsf{snd}(u,v) & \rightarrow_{\beta} & v \\[1ex]
\mathsf{case}\ \mathsf{inl}(e)\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f \mid \mathsf{inl}(y) \rightarrow g & \rightarrow_{\beta} & f[e/x] \\[1ex]
\mathsf{case}\ \mathsf{inr}(e)\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f \mid \mathsf{inl}(y) \rightarrow g & \rightarrow_{\beta} & g[e/y] \\[1ex]
\mathsf{let}\ x \Leftarrow \mathsf{val}(u)\ \mathsf{in}\ v & \rightarrow_{\beta} & v[u/x] \\[1ex]
\mathsf{fst}(\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f \mid \mathsf{inr}(y) \rightarrow g) & \rightarrow_{c} & \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow \mathsf{fst}(f) \mid \mathsf{inr}(y) \rightarrow \mathsf{fst}(g) \\[1ex]
\mathsf{snd}(\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f \mid \mathsf{inr}(y) \rightarrow g) & \rightarrow_{c} & \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow \mathsf{snd}(f) \mid \mathsf{inr}(y) \rightarrow \mathsf{snd}(g) \\[1ex]
(\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f \mid \mathsf{inr}(y) \rightarrow g)\ h & \rightarrow_{c} & \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f\ h \mid \mathsf{inr}(y) \rightarrow g\ h \\[1ex]
\begin{array}{l} \mathsf{case}\ (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f \mid \mathsf{inr}(y) \rightarrow g) \\ \quad \mathsf{of}\ \mathsf{inl}(w) \rightarrow h \mid \mathsf{inr}(z) \rightarrow k \end{array} & \rightarrow_{c} & \begin{array}{l} \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow \mathsf{case}\ f\ \mathsf{of}\ \mathsf{inl}(w) \rightarrow h \\ \qquad\qquad\qquad\qquad\qquad\quad \mid \mathsf{inr}(z) \rightarrow k \\ \qquad\quad \mid \mathsf{inr}(y) \rightarrow \mathsf{case}\ g\ \mathsf{of}\ \mathsf{inl}(w) \rightarrow h \\ \qquad\qquad\qquad\qquad\qquad\quad \mid \mathsf{inr}(z) \rightarrow k \end{array} \\[1ex]
\begin{array}{l} \mathsf{let}\ z \Leftarrow (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f \\ \qquad\qquad\qquad \mid \mathsf{inr}(y) \rightarrow g)\ \mathsf{in}\ h \end{array} & \rightarrow_{c} & \begin{array}{l} \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow \mathsf{let}\ z \Leftarrow f\ \mathsf{in}\ h \\ \qquad\quad \mid \mathsf{inr}(y) \rightarrow \mathsf{let}\ z \Leftarrow g\ \mathsf{in}\ h \end{array} \\[1ex]
\mathsf{let}\ x \Leftarrow (\mathsf{let}\ y \Leftarrow u\ \mathsf{in}\ v)\ \mathsf{in}\ f & \rightarrow_{c} & \mathsf{let}\ y \Leftarrow u\ \mathsf{in}\ (\mathsf{let}\ x \Leftarrow v\ \mathsf{in}\ f)
\end{array}
$$

Figure 6: Reduction rules for terms

Prawitz [20]. It is possible to use Prawitz's technique to give a proof of strong normalisation for $\lambda\mathrm{ML}_T$ (the first draft of this paper contained such a proof), but the proof is long, complicated and unenlightening. Instead, we will use a translation argument like that previously used by Benton to show strong normalisation for the linear term calculus [2].

If we wish to show strong normalisation for a language $\mathcal{L}_1$, and we already know that strong normalisation holds for another language $\mathcal{L}_2$, then it suffices to exhibit a translation $(\cdot)^{\circ}\colon \mathcal{L}_1 \rightarrow \mathcal{L}_2$ such that $\forall e, f \in \mathcal{L}_1 . e \rightarrow_1 f \;\Rightarrow\; e^{\circ} \rightarrow_2^{+} f^{\circ}$, where $\rightarrow_1$ and $\rightarrow_2$ are the one-step reduction relations in $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. This is because any infinite reduction sequence in $\mathcal{L}_1$ would then induce an infinite reduction sequence in $\mathcal{L}_2$, contradicting strong normalisation for that language. Here we will take $\mathcal{L}_1$ to be the computational lambda calculus, with the $\rightarrow_{\beta c}$ reduction relation obtained by taking the precongruence closure of the rules shown in Figure 6, and $\mathcal{L}_2$ to be the simply typed lambda calculus with coproducts and the usual $\rightarrow_{\beta c}$ reduction relation. Note that $\mathcal{L}_2$ is just the largest sublanguage of $\mathcal{L}_1$ which does not contain the T type constructor or either of the let and val constructs.

Proposition 4 *$\mathcal{L}_2$ is strongly normalising.*

Proof. This is the result proved by Prawitz [20].[2] □

The translation $(\cdot)^\circ$ is simply to instantiate the generic monad operations of the computational lambda calculus with the specific case of the exceptions monad which we mentioned earlier. We start by defining the translation of $\mathcal{L}_1$ types to $\mathcal{L}_2$ types:

$$\begin{aligned}
G^\circ &\overset{\text{def}}{=} G \quad \text{(for } G \text{ a base type)}\\
1^\circ &\overset{\text{def}}{=} 1\\
(A \mathbin{\triangle} B)^\circ &\overset{\text{def}}{=} A^\circ \mathbin{\triangle} B^\circ \quad (\text{for } \triangle \in \{\times, +, \rightarrow\})\\
(TA)^\circ &\overset{\text{def}}{=} 1 + A^\circ
\end{aligned}$$

Next we define the translation of $\mathcal{L}_1$ terms to $\mathcal{L}_2$ terms:

$$\begin{aligned}
*^\circ &\overset{\text{def}}{=} * & x^\circ &\overset{\text{def}}{=} x\\
(\lambda x{:}\, A.e)^\circ &\overset{\text{def}}{=} \lambda x{:}\, A^\circ.e^\circ & (e\,f)^\circ &\overset{\text{def}}{=} e^\circ\, f^\circ\\
(\mathsf{fst}(e))^\circ &\overset{\text{def}}{=} \mathsf{fst}(e^\circ) & (\mathsf{snd}(e))^\circ &\overset{\text{def}}{=} \mathsf{snd}(e^\circ)
\end{aligned}$$

$$(e, f)^\circ \overset{\text{def}}{=} (e^\circ, f^\circ)$$

$$\begin{aligned}
(\mathsf{inl}(e))^\circ &\overset{\text{def}}{=} \mathsf{inl}(e^\circ) & (\mathsf{inr}(e))^\circ &\overset{\text{def}}{=} \mathsf{inr}(e^\circ)
\end{aligned}$$

$$\begin{aligned}
(\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f \mid \mathsf{inr}(y) \rightarrow g)^\circ &\overset{\text{def}}{=} \mathsf{case}\ e^\circ\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow f^\circ \mid \mathsf{inr}(y) \rightarrow g^\circ\\
(\mathsf{val}(e))^\circ &\overset{\text{def}}{=} \mathsf{inr}(e^\circ)\\
(\mathsf{let}\ x \Leftarrow e\ \mathsf{in}\ f)^\circ &\overset{\text{def}}{=} \mathsf{case}\ e^\circ\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) \mid \mathsf{inr}(x) \rightarrow f^\circ
\end{aligned}$$

The following two lemmas are both easy inductions:

Lemma 5 *If* $\Gamma \vdash_{\lambda \mathrm{ML}_T} e{:}\, A$ *then* $\Gamma^\circ \vdash_{\mathcal{L}_2} e^\circ{:}\, A^\circ$. □

Lemma 6 *The* $(\cdot)^\circ$ *translation commutes with substitution: for any terms* e, f *of* $\lambda\mathrm{ML}_T$ *and for any variable* x, $(e[f/x])^\circ = e^\circ[f^\circ/x]$. □

Proposition 7 *For any terms* e, f *of the computational lambda calculus, if* $e \rightarrow_{\beta c} f$ *then* $e^\circ \rightarrow^{+}_{\beta c} f^\circ$.

Proof. This is proved by induction on the derivation of the fact that $e \rightarrow_{\beta c} f$. The induction cases are the rules (which we have not explicitly given) which make $\rightarrow_{\beta c}$ into a precongruence and these all follow trivially from the compositional nature of the translation. Similarly, most of the axioms in Figure 6 are easy to deal with. For example

- In the case of the reduction

  $$e = (\lambda x{:}\, A.u)\ v \ \rightarrow_\beta\ u[v/x] = f$$

  we have

  $$e^\circ = (\lambda x{:}\, A^\circ.u^\circ)\ v^\circ \ \rightarrow_\beta\ u^\circ[v^\circ/x] = f^\circ$$

  where the last equality follows from Lemma 6.

[2]Actually, Prawitz considers the commuting conversions associated with existential quantification as well as with disjunction.

The interesting cases are the one $\rightarrow_\beta$ and two $\rightarrow_c$ reductions involving the computation type:

- In the case of the reduction

$$e = (\mathsf{let}\ x \Leftarrow \mathsf{val}(u)\ \mathsf{in}\ v) \quad \rightarrow_\beta \quad v[u/x] = f$$

  we have

$$e^\circ = (\mathsf{case}\ \mathsf{inr}(u^\circ)\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(x) \rightarrow v^\circ) \quad \rightarrow_\beta \quad v^\circ[u^\circ/x] = f^\circ$$

- In the case $e \rightarrow_c f$ where

$$\begin{array}{rcl} e & = & \mathsf{let}\ w \Leftarrow (\mathsf{case}\ s\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow t |\ \mathsf{inr}(y) \rightarrow u)\ \mathsf{in}\ v \\ f & = & \mathsf{case}\ s\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow (\mathsf{let}\ w \Leftarrow t\ \mathsf{in}\ v) |\ \mathsf{inr}(y) \rightarrow (\mathsf{let}\ w \Leftarrow u\ \mathsf{in}\ v) \end{array}$$

  we have

$$\begin{array}{rcl} e^\circ & = & \mathsf{case}\ (\mathsf{case}\ s^\circ\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow t^\circ |\ \mathsf{inr}(y) \rightarrow u^\circ)\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(w) \rightarrow v^\circ \\ & \rightarrow_c & \mathsf{case}\ s^\circ\ \mathsf{of}\ \mathsf{inl}(x) \rightarrow (\mathsf{case}\ t^\circ\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(w) \rightarrow v^\circ) \\ & & \qquad |\ \mathsf{inr}(y) \rightarrow (\mathsf{case}\ u^\circ\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(w) \rightarrow v^\circ) \\ & = & f^\circ \end{array}$$

- In the case $e \rightarrow_c f$ where

$$\begin{array}{rcl} e & = & \mathsf{let}\ x \Leftarrow (\mathsf{let}\ y \Leftarrow t\ \mathsf{in}\ u)\ \mathsf{in}\ v \\ f & = & \mathsf{let}\ y \Leftarrow t\ \mathsf{in}\ (\mathsf{let}\ x \Leftarrow u\ \mathsf{in}\ v) \end{array}$$

  we have

$$\begin{array}{rcl} e^\circ & = & \mathsf{case}\ (\mathsf{case}\ t^\circ\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(y) \rightarrow u^\circ)\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(x) \rightarrow v^\circ \\ & \rightarrow_c & \mathsf{case}\ t^\circ\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow (\mathsf{case}\ \mathsf{inl}(z)\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(x) \rightarrow v^\circ) \\ & & \qquad |\ \mathsf{inr}(y) \rightarrow (\mathsf{case}\ u^\circ\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(x) \rightarrow v^\circ) \\ & \rightarrow_\beta & \mathsf{case}\ t^\circ\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) \\ & & \qquad |\ \mathsf{inr}(y) \rightarrow (\mathsf{case}\ u^\circ\ \mathsf{of}\ \mathsf{inl}(z) \rightarrow \mathsf{inl}(z) |\ \mathsf{inr}(x) \rightarrow v^\circ) \\ & = & f^\circ \end{array}$$

  as required.

□

Corollary 8 (Strong Normalisation) *The calculus $\lambda\mathrm{ML}_T$ with the $\rightarrow_{\beta c}$ reduction relation is strongly normalising.*

Proof. By Propositions 7 and 4. □

5 Cut Elimination

Corresponding to the normalisation process for natural deductions, there is also a simplification process for sequent calculus proofs, which consists of removing applications of the cut-rule. In this section we look at the process of cut-elimination for CL-logic and see that it is closely related to the normalisation process. So as to be able to look at the term reductions induced by cut-elimination, we shall work with the sequent calculus presentation of $\lambda\mathrm{ML}_T$, which we have not so far given explicitly.

We shall use the notation that the application of the *Cut* rule in the following proof is called a (D_1, D_2)-cut.

$$\dfrac{\dfrac{}{\Gamma \vdash e : A}\,D_1 \qquad \dfrac{}{\Delta, x : A \vdash f : B}\,D_2}{\Gamma, \Delta \vdash f[e/x] : B}\,\mathit{Cut}$$

As in our previous work in intuitionistic linear logic [1], we have principal cuts and secondary cuts as well as insignificant cuts.

5.1 Principal Cuts

Principal cuts are the ones where we introduce on the right and on the left of a cut the formula that is being cut. The 'new' case is

$$\dfrac{\dfrac{\Gamma \vdash e : A}{\Gamma \vdash \mathsf{val}(e) : TA}\,T_{\mathcal{R}} \qquad \dfrac{\Delta, x : A \vdash f : TB}{\Delta, z : TA \vdash \mathsf{let}\, x \Leftarrow z \,\mathsf{in}\, f : TB}\,T_{\mathcal{L}}}{\Gamma, \Delta \vdash \mathsf{let}\, x \Leftarrow \mathsf{val}(e) \,\mathsf{in}\, f : TB}\,\mathit{Cut}$$

This derivation reduces to

$$\dfrac{\Gamma \vdash e : A \qquad \Delta, x : A \vdash f : TB}{\Gamma, \Delta \vdash f[e/x] : TB}\,\mathit{Cut}$$

giving us the reduction rule

$$\mathsf{let}\, x \Leftarrow \mathsf{val}(e) \,\mathsf{in}\, f \;\rightarrow\; f[e/x]$$

which is, of course, the same reduction as we obtained from the normalisation process and corresponds to the β-equality axiom given by Moggi.

We also have principal cuts with respect to conjunction, disjunction and implication, but these are standard.

5.2 Secondary Cuts

One case of a secondary cut is the following

$$\dfrac{\dfrac{\Gamma, x : A \vdash e : TB}{\Gamma, w : TA \vdash \mathsf{let}\, x \Leftarrow w \,\mathsf{in}\, e : TB}\,T_{\mathcal{L}} \qquad \dfrac{\Delta, y : B \vdash f : TC}{\Delta, z : TB \vdash \mathsf{let}\, y \Leftarrow z \,\mathsf{in}\, f : TC}\,T_{\mathcal{L}}}{\Gamma, w : TA, \Delta \vdash \mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow w \,\mathsf{in}\, e) \,\mathsf{in}\, f : TC}\,\mathit{Cut}$$

this derivation reduces to

$$\dfrac{\dfrac{\Gamma, x : A \vdash e : TB \qquad \dfrac{\Delta, y : B \vdash f : TC}{\Delta, z : TB \vdash \mathsf{let}\, y \Leftarrow z \,\mathsf{in}\, f : TC}\, T_{\mathcal{L}}}{\Gamma, x : A, \Delta \vdash \mathsf{let}\, y \Leftarrow e \,\mathsf{in}\, f : TC}\, \mathit{Cut}}{\Gamma, w : TA, \Delta \vdash \mathsf{let}\, x \Leftarrow w \,\mathsf{in}\, (\mathsf{let}\, y \Leftarrow e \,\mathsf{in}\, f) : TC}$$

This reduction gives us the equality

$$\mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow w \,\mathsf{in}\, e) \,\mathsf{in}\, f = \mathsf{let}\, x \Leftarrow w \,\mathsf{in}\, (\mathsf{let}\, y \Leftarrow e \,\mathsf{in}\, f)$$

which corresponds to Moggi's third axiom and which we also derived from the normalisation process on natural deductions.

A secondary cut, like

$$\dfrac{\dfrac{\Gamma \vdash e : A}{\Gamma \vdash \mathsf{val}(e) : TA}\, T_{\mathcal{R}} \qquad \dfrac{\Delta, x : TA \vdash f : B}{\Delta, x : TA \vdash \mathsf{val}(f) : TB}\, T_{\mathcal{R}}}{\Gamma, \Delta \vdash \mathsf{val}(f)[\,\mathsf{val}(e)/x] : TB}\, \mathit{Cut}$$

reducing to

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash e : A}{\Gamma \vdash \mathsf{val}(e) : TA}\, T_{\mathcal{R}} \qquad \Delta, x : TA \vdash f : B}{\Gamma, \Delta \vdash f[\,\mathsf{val}(e)/x] : B}\, \mathit{Cut}}{\Gamma, \Delta \vdash \mathsf{val}(f[\,\mathsf{val}(e)/x]) : TB}\, T_{\mathcal{R}}$$

does not introduce new axioms. This is an example of what we call an *insignificant* cut.

Theorem 9 (Cut Elimination) *Given a derivation π of a sequent $\vec{x}{:}\,\Gamma \vdash e{:}\,A$, a derivation π' of $\vec{x}{:}\,\Gamma \vdash e'{:}\,A$ can be found which contains no instances of the* Cut *rule.*

Proof. This was first proved by Curry [6] and is along standard lines. □

5.3 η-equalities

Each of the type constructors of $\lambda\mathrm{ML}_T$ also has an associated η-equality as well as β and commutation equalities. We have already seen that the latter two classes of equalities follow as direct consequences of the proof theory of CL-logic and we now try to explain the η-equalities in the same spirit. This seems to work out most naturally if we use a *multiplicative* (disjoint contexts), rather than an *additive* (shared contexts) formulation of the sequent rules of the logic. Given such a presentation, the traditional η-equality associated with the function-space constructor arises from reducing the derivation

$$\dfrac{\dfrac{x : A \vdash x : A \qquad y : B \vdash y : B}{x : A, f : A \to B \vdash fx : B}\, (\to_{\mathcal{L}})}{f : A \to B \vdash \lambda x.fx : A \to B}\, (\to_{\mathcal{R}})$$

to the derivation

$$\dfrac{}{f : A \to B \vdash f : A \to B}\, \mathit{Identity}$$

We can similarly obtain the η-rules for $\wedge$ and $\vee$ as consequences of (roughly) simplifying a right rule applied to the identity, followed by a left rule to an identity on the compound proposition. Following this general pattern, the η-rule for T can be obtained by reducing the derivation

$$\dfrac{\dfrac{x : A \vdash x : A}{x : A \vdash \mathsf{val}(x) : TA}\,T_{\mathcal{R}}}{z : TA \vdash \mathsf{let}\ x \Leftarrow z\ \mathsf{in}\ \mathsf{val}(x) : TA}\,T_{\mathcal{L}}$$

to the derivation

$$\dfrac{}{z : TA \vdash z : TA}\,\mathit{Identity}$$

and this does indeed yield exactly the second of the three equalities on $\lambda\mathrm{ML}_T$ terms which we listed on page 2. Thus we can now see the full theory of $\beta c\eta$-equality for $\lambda\mathrm{ML}_T$ as a natural consequence of the proof theory of CL-logic.

6 Categorical Models

Since the computational lambda calculus was originally derived from categorical considerations [17], we already know that a categorical model is a cartesian closed category (CCC) with a strong monad. For completeness we shall give these categorical definitions.

Definition 1 *A* monad *over a category $\mathcal{C}$ is a triple (T, η, μ), where $T: \mathcal{C} \to \mathcal{C}$ is a functor, and $\eta: Id \dot{\to} T$ and $\mu: T^2 \dot{\to} T$ are natural transformations which satisfy the following diagrams:*

$$\begin{tikzcd}
T^3A \arrow[r, "\mu_{TA}"] \arrow[d, "T\mu_A"'] & T^2A \arrow[d, "\mu_A"] \\
T^2A \arrow[r, "\mu_A"'] & TA
\end{tikzcd}
\qquad
\begin{tikzcd}
TA \arrow[r, "T\eta_A"] \arrow[d, "\eta_{TA}"'] \arrow[dr, "id"'] & T^2A \arrow[d, "\mu_A"] \\
T^2A \arrow[r, "\mu"'] & TA
\end{tikzcd}$$

Definition 2 *A* strong monad *over a category $\mathcal{C}$ with finite products is a monad (T, η, μ), together with a natural transformation $\tau_{A,B}: A \times TB \to T(A \times B)$, such that the following 4 diagrams commute, where λ and α are the evident natural isomorphisms:*

$$\begin{tikzcd}
1 \times TA \arrow[r, "\tau"] \arrow[dr, "\lambda"] & T(1 \times A) \arrow[d, "T(\lambda)"] \\
& TA
\end{tikzcd}
\qquad
\begin{tikzcd}
A \times B \arrow[r, "id \times \eta"] \arrow[dr, "\eta"] & A \times TB \arrow[d, "\tau"] \\
& T(A \times B)
\end{tikzcd}$$

$$\begin{tikzcd}
(A \times B) \times TC \arrow[r, "\alpha"] \arrow[d, "\tau"'] & A \times (B \times TC) \arrow[r, "id \times \tau"] & A \times T(B \times C) \arrow[d, "\tau"] \\
T((A \times B) \times C) \arrow[rr, "T(\alpha)"'] & & T(A \times (B \times C))
\end{tikzcd}$$

$$\begin{tikzcd}
A \times T^2 B \arrow[r, "\tau"] \arrow[d, "id \times \mu"'] & T(A \times TB) \arrow[r, "T(\tau)"] & T^2(A \times B) \arrow[d, "\mu"] \\
A \times TB \arrow[rr, "\tau"'] & & T(A \times B)
\end{tikzcd}$$

Definition 3 *A* CL-model *is a cartesian closed category with binary coproducts and a strong monad.*

Figure 7 outlines the way in which $\lambda\mathrm{ML}_T$ is modelled in a CL-model. As one would expect, any CL-model validates all the η equalities as well as those arising from β-reductions and commuting conversions.[3] The prototypical computer science example of a CL-model is the category of ω-cpos (not necessarily with a bottom) with continuous maps and the lifting monad.

Note how the tensorial strength is needed in the interpretation of the T-elimination rule. This is an instance of a general phenomenon which arises in modelling many different logics (see, for example, [1, 3, 4]) – the interpretations of logical connectives have to behave well with respect to the (tensor) product which is used to represent the multicategorical structure implied by the comma on the left of sequents. This means firstly that there has to be some extra categorical structure relating the interpretation of the connective to the (tensor) product, and secondly that this extra structure has to satisfy some coherence conditions like those above, so that the interpretation in the category of each proof is really uniquely determined. It is also worth remarking that the existence of a strong monad is equivalent to the existence of a *monoidal monad* [8] which would be an alternative way of presenting the models.

7 Conclusions and Future Work

We have shown that Moggi's computational lambda calculus, which was initially arrived at from a purely categorical perspective, with no thoughts of proof theory, actually corresponds via the propositions-as-types analogy to an intuitionistic modal logic. Whilst CL-logic is rather odd from the point of view of traditional work in modal logic (in particular, the modality has aspects of both possibility and necessity), it seems natural and well-behaved. Further evidence that CL-logic is indeed a 'naturally occurring' logic comes from the fact that both Curry [6] and Fairtlough and Mendler have also discovered it [9]. Fairtlough and Mendler's work is particularly interesting because, although they discuss the same logic, their motivation and methodology is rather different from ours. They are interested in the specification and verification of hardware and noted that a general weakened notion of correctness *'correctness up to constraints'* can help one to reason about the real-life behaviour of circuits (e.g. the fact that gates only stabilise some time after their inputs are changed) without having completely to model such low-level details. Their logic PLL seems useful in proving properties of circuits under a number of different notions of constraint, which is very reminiscent of the way in which the computational

[3] It is, of course, also possible to relax some of the categorical definitions so as to obtain a class of models which only model βc-equalities. For example, if we do not insist on the η-equality associated with the modality then instead of a strong monad, we would only require a *right pre-monad* [16].

$$\frac{}{\Gamma \times A \xrightarrow{\pi_2} A}\ \textit{Identity} \qquad \frac{}{\Gamma \xrightarrow{!} 1}\ (1_{\mathcal{I}})$$

$$\frac{\Gamma \times A \xrightarrow{e} B}{\Gamma \xrightarrow{cur(e)} B^A}\ (\to_{\mathcal{I}}) \qquad \frac{\Gamma \xrightarrow{e} B^A \qquad \Gamma \xrightarrow{f} A}{\Gamma \xrightarrow{\langle e,f\rangle} B^A \times A \xrightarrow{\mathrm{ev}} B}\ (\to_{\mathcal{E}})$$

$$\frac{\Gamma \xrightarrow{e} A \qquad \Gamma \xrightarrow{f} B}{\Gamma \xrightarrow{\langle e,f\rangle} A \times B}\ (\times_{\mathcal{I}}) \quad \frac{\Gamma \xrightarrow{e} A \times B}{\Gamma \xrightarrow{e} A \times B \xrightarrow{\pi_1} A}\ (\times_{\mathcal{E}}) \quad \frac{\Gamma \xrightarrow{e} A \times B}{\Gamma \xrightarrow{e} A \times B \xrightarrow{\pi_2} B}\ (\times_{\mathcal{E}})$$

$$\frac{\Gamma \xrightarrow{e} A}{\Gamma \xrightarrow{e} A \xrightarrow{inl} A + B}\ (+_{\mathcal{I}}) \qquad \frac{\Gamma \xrightarrow{e} B}{\Gamma \xrightarrow{e} B \xrightarrow{inr} A + B}\ (+_{\mathcal{I}})$$

$$\frac{\Gamma \xrightarrow{e} A + B \qquad \Gamma \times A \xrightarrow{f} C \qquad \Gamma \times B \xrightarrow{g} B}{\Gamma \xrightarrow{\langle id,e\rangle} \Gamma \times (A + B) \cong (\Gamma \times A) + (\Gamma \times B) \xrightarrow{[f,g]} C}\ (+_{\mathcal{E}})$$

$$\frac{\Gamma \xrightarrow{e} A}{\Gamma \xrightarrow{e} A \xrightarrow{\eta} TA}\ (T_{\mathcal{I}}) \quad \frac{\Gamma \xrightarrow{e} TA \qquad \Gamma \times A \xrightarrow{f} TB}{\Gamma \xrightarrow{\langle id,e\rangle} \Gamma \times TA \xrightarrow{\tau} T(\Gamma \times A) \xrightarrow{Tf} T^2B \xrightarrow{\mu} TB}\ (T_{\mathcal{E}})$$

Figure 7: Modelling the computational lambda calculus

lambda calculus is an useful metalanguage for describing a number of different notions of computation. Fairtlough and Mendler's work is essentially concerned with provability, rather than proofs, and they give a novel (and complete) semantics for the logic using a non-standard class of Kripke models. We hope to investigate further just how these Kripke models are related to the general categorical models we have considered.

Our logical reconstruction of $\lambda\mathrm{ML}_T$ shows how the equational axioms which were initially imposed on the calculus are actually *consequences* of the proof theory of the logic. We have also extended the class of interesting constructive logics for which there is a perfect three-way correspondence between logic, term calculus and categorical models. This is part of an ongoing project of ours, see [1, 4, 3]. In fact, there is a close relationship between CL-logic and intuitionstic linear logic. Any linear category (model for intuitionistic linear logic, see [1]), gives rise to a CL-model as a subcategory of the category of algebras for the ! comonad. Whilst this is interesting, not all CL-models arise in this way because the monad part of the model is always a *commutative* strong monad (equivalently, a *symmetric monoidal monad*). More discussion of the relationship between intuitionistic linear logic and CL-logic may be found in [3], but there is still scope for further work looking at whether, for example, CL-logic is more closely associated with a non-commutative variant of intuitionistic linear logic.

The strong normalisation proof we have given is extremely straightforward, particularly by comparison with the proof from first principles which it replaces. A similar, but more subtle, translation argument (into System F) has also been used to prove strong normalisation for the linear term calculus [2]. There are almost certainly a number of other calculi where this technique can profitably be applied.

There are a several natural extensions to CL-logic yet to be explored. We should like to consider the addition of first-order quantifiers to CL-logic. From a proof theoretical viewpoint, these present little difficulty, but from a categorical viewpoint, they would seem to suggest a move to a hyperdoctrine model (see, for example, [21]). We should also like to investigate a *linear* computational lambda calculus, where we have computational types over the linear lambda calculus [1].

References

[1] P.N. Benton, G.M. Bierman, V.C.V. de Paiva, and J.M.E. Hyland. Term assignment for intuitionistic linear logic. Technical Report 262, Computer Laboratory, University of Cambridge, August 1992.

[2] P. N. Benton. Strong normalisation for the linear term calculus. *Journal of Functional Programming*, 5(1):65–80, January 1995.

[3] P. N. Benton. A mixed linear and non-linear logic: proofs, terms and models. In *Selected Papers from Computer Science Logic 1994, Kazimierz, Poland*, Springer-Verlag LNCS 933, 1995. Full version available as Technical Report 352, University of Cambridge Computer Laboratory, October 1994.

[4] G.M. Bierman and V.C.V. de Paiva. Intuitionistic necessity revisited. In *Proceedings of Logic at Work Conference. Amsterdam, Holland*, December 1992.

[5] R.L. Crole. *Programming Metalogics With a Fixpoint Type.* PhD thesis, Computer Laboratory, University of Cambridge, 1992. Revised version available as Technical Report 247.

[6] H.B. Curry. The elimination theorem when modality is present. *The Journal of Symbolic Logic*, 17(4):249–265, January 1957.

[7] D. Van Dalen. *Intuitionistic Logic*, volume 3 of *Handbook of Philosophical Logic*, chapter 4, pages 225–339. Reidel, 1986.

[8] S. Eilenberg and G.M. Kelly. Closed categories. In *Proceedings of Conference on Categorical Algebra, La Jolla*, 1966.

[9] M. Fairtlough and M. Mendler. An intuitionistic modal logic with applications to the formal verification of hardware. In *Selected Papers from Computer Science Logic 1994, Kazimierz, Poland*, Springer-Verlag LNCS 933, 1995. Also available as Technical Report ID-TR: 1994-133, Department of Computer Science, Technical University of Denmark, January 1994.

[10] J. Gallier. Constructive logics part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110(2):249–339, March 1993.

[11] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.

[12] A.D. Gordon. *Functional Programming and Input/Output.* PhD thesis, Computer Laboratory, University of Cambridge, February 1993. Available as Computer Laboratory Technical Report 285.

[13] R.P. Goré. *Cut-free Sequent and Tableau Systems for Propositional Normal Modal Logics.* PhD thesis, Computer Laboratory, University of Cambridge, 1992. Available as Computer Laboratory Technical Report 257.

[14] W. Hodges. Elementary predicate logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic. Volume 1*, chapter 1, pages 1–131. D. Reidel, 1983.

[15] W. A. Howard. The formulæ-as-types notion of construction. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 1980.

[16] R. Kieburtz, B. Agapiev, and J. Hook. Three monads for continuations. In *Proceedings of ACM Workshop on Continuations*, pages 39–48, 1992.

[17] E. Moggi. Computational lambda-calculus and monads. In *Proceedings of Symposium on Logic in Computer Science*, pages 14–23, June 1989.

[18] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[19] A.M. Pitts. Evaluation logic. Technical Report 198, Computer Laboratory, University of Cambridge, August 1990.

[20] D. Prawitz. Ideas and results in proof theory. In J.E. Fenstad, editor, *Proceedings of Second Scandinavian Logic Symposium*, pages 235–307, 1971.

[21] R.A.G. Seely. Hyperdoctrines, natural deduction and the Beck condition. *Zeitschrift für Math. Logik*, 29:505–542, 1983.

[22] M. Spivey. A functional theory of exceptions. *Science of Computer Programming*, 14(1):25–42, June 1990.

[23] W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32, 1967.

[24] P. Wadler. Comprehending monads. In *Proceedings of Conference on LISP and Functional Programming*, pages 61–78. ACM, June 1990.

[25] P. Wadler. The essence of functional programming. Invited talk. In *19th Symposium on Principles of Programming Languages.* ACM, January 1992.