J. Functional Programming **8** (2): 177–193, March 1998. Printed in the United Kingdom
© 1998 Cambridge University Press

Computational types from a logical perspective

P. N. BENTON
Persimmon IT Inc., Cambridge, UK

G. M. BIERMAN
Gonville and Caius College, Cambridge, UK

V. C. V. DE PAIVA
School of Computer Science, University of Birmingham, Birmingham, UK

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 strong normalisation and confluence results.

Capsule Review

This is a short, concise and well-written paper that addresses Moggi's Computational Lambda Calculus (CLC) from an interesting but little explored perspective.

The authors take the CLC, extend it with coproducts and apply the Curry–Howard correspondence to obtain a propositional intuitionistic modal logic which they call CL-logic. This logic has independent interest but little has so far appeared on its proof theory or applications.

The authors give equivalent axiomatic, sequent calculus and natural deduction presentations of CL-logic and give an indication of a simple proof of cut-elimination for the sequent calculus. As the main technical contribution of the paper, they give an elegant proof of strong normalisation for natural deduction proofs in CL-logic via a translation into the simply typed lambda calculus with coproducts. They go on to show how the three characteristic equations of the CLC arise naturally from their proof-theoretical analyses.

They round off the paper by considering Cartesian closed categories with finite coproducts and strong monads as appropriate categorical models of CL-logic and tabulating the model construction.

1 Introduction

The computational lambda calculus was introduced by Moggi (1989; 1991) 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.

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 (1990) in the special case of computations which can raise exceptions, 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 (1992) and Pitts (1990). The computational lambda calculus is the syntactic theory which expresses this semantic idea of notions of 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 (Wadler, 1990; Wadler, 1992; Gordon, 1994).

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 a curious possibility-like modality $\Diamond$, 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 (1952) had briefly considered just such a system in the early 1950s. More recently, and with completely different motivations (hardware verification), Fairtlough and Mendler (1995) have come up with sequent calculus and Hilbert-style presentations of CL-logic.

2 Computational Lambda Calculus

The Computational Lambda Calculus (CLC), 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 shall 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.

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

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

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

$$\frac{\Gamma \vdash e{:}A}{\Gamma \vdash \mathsf{inl}_{A+B}(e){:}A + B} \qquad \frac{\Gamma \vdash e{:}B}{\Gamma \vdash \mathsf{inr}_{A+B}(e){:}A + B} \qquad \frac{\Gamma \vdash e{:}0}{\Gamma \vdash \nabla_A(e){:}A}$$

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

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

Fig. 1. Natural deduction presentation of the Computational Lambda Calculus.

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 $T(A) \stackrel{\text{def}}{=} \wp(A)$ with

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

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

$$\begin{aligned}
\mathsf{val}(e) &\stackrel{\mathrm{def}}{=} \mathsf{inr}_{1+A}(e)\\
(\mathsf{let}\, x \Leftarrow e \,\mathsf{in}\, f) &\stackrel{\mathrm{def}}{=} \mathsf{case}\; e \;\mathsf{of}\; \mathsf{inl}(*) \to \mathsf{inl}_{1+A}(*) \mid \mathsf{inr}(x) \to f.
\end{aligned}$$

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

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

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 $\Gamma, x : A \vdash f : B$ then $\Gamma \vdash f[e/x] : B$.

3 Propositional CL-logic

In this section we use the Curry–Howard correspondence to derive a natural deduction logic from Moggi's original presentation of $\lambda \mathrm{ML}_T$. We shall also consider sequent calculus and axiomatic formulations of the same logic and will sometimes subscript turnstiles with one of N, S or H to indicate which system is meant.

3.1 Natural deduction formulation of CL-logic

Using the Curry–Howard correspondence (Howard, 1980), 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	$\top$
0	$\bot$
$\times$	$\wedge$
$\to$	$\supset$
$+$	$\vee$
T	$\Diamond$

Hence we derive the logic, called propositional CL-logic, given in 'sequent-style' natural deduction form in figure 2.

3.2 Sequent calculus for CL-logic

We can use the well-known correspondence between natural deduction and sequent calculus proof systems to derive systematically a sequent calculus formulation of CL-logic. This is given in figure 3.

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

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

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

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (\vee_{\mathscr{I}}) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (\vee_{\mathscr{I}}) \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ (\bot_{\mathscr{E}})$$

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

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

Fig. 2. Natural deduction formulation of CL-logic.

$$\frac{}{\Gamma, A \vdash A}\ \textit{Identity} \qquad \frac{\Gamma \vdash B \qquad B, \Gamma \vdash C}{\Gamma \vdash C}\ \textit{Cut}$$

$$\frac{}{\Gamma, \bot \vdash A}\ (\bot_{\mathscr{L}}) \qquad \frac{}{\Gamma \vdash \top}\ (\top_{\mathscr{R}})$$

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

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

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

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

Fig. 3. Sequent calculus for CL-logic.

Proposition 2

The sequent calculus and natural deduction presentations of CL-logic are equivalent, i.e.

$$\Gamma \vdash_N A \quad \Leftrightarrow \quad \Gamma \vdash_S A$$

Proof
This follows by inductively defining proof translations in both directions between the two systems. One natural deduction can correspond to many different sequent proofs. $\square$

Having arrived at CL-logic via the computational lambda calculus, we were somewhat surprised to discover that this kind of possibility modality had actually been considered over forty years ago by Curry (1952). However, Curry was dismissive of this formulation:

"The referee has pointed out that for certain kinds of modality it [the introduction rule for $\Diamond$] is not acceptable ... 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, and this has some surprising consequences. If $\Diamond A$ is to be understood as 'A is possible' and, furthermore, we were to try to add negation to our logic, then the following sequent

$$\Diamond A \wedge \Diamond\neg A \vdash \Diamond(A \wedge \neg A)$$

would be provable, though intuitively wrong (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 we have presented is certainly consistent.

3.3 *Hilbert system for CL-logic*

To complete our presentation of CL-logic, we give a Hilbert-style formulation, shown in figure 4. This is the most common way of presenting modal logics and the modality axioms given here, though slightly unusual, were derived from the natural deduction formulation of the logic following a procedure given by Hodges (1983). The following two results about our Hilbert system follow by straightforward induction:

Proposition 3 (*Deduction Theorem*)
If $\Gamma, A \vdash_H B$ then $\Gamma \vdash_H A \supset B$.

Proposition 4
The natural deduction and axiomatic presentations of CL-logic are equivalent, i.e.

$$\Gamma \vdash_N A \qquad \Leftrightarrow \qquad \Gamma \vdash_H A$$

It should be noted that Fairtlough and Mendler (1995) have also (independently) proposed a Hilbert-style presentation of CL-logic (which they dub PLL, for *Propositional Lax Logic*), although they give three axioms[1] for the modality, which they

[1] The categorically minded reader will recognise these axioms as the unit, multiplication and functoriality axioms for a monad.

Axioms

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

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

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

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

Rules

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

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

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

Fig. 4. Hilbert system for CL-logic.

write as $\bigcirc$ since it has aspects of both possibility and necessity. These are

$$\begin{array}{lll} \mathbf{R} & : & A \supset \bigcirc A, \\ \mathbf{M} & : & \bigcirc\bigcirc A \supset \bigcirc A \ \textit{and} \\ \mathbf{F} & : & (A \supset B) \supset (\bigcirc A \supset \bigcirc B) \end{array}$$

This alternative axiomatisation of the logic is equivalent to that given in figure 4 in that the axioms of each system are theorems in the other.

4 Normalisation

We now turn to the question of how to normalise proofs in CL-logic, that is, what equalities we want to have on proofs. Both the natural deduction and the sequent calculus formulations of the logic have an associated normalisation procedure, and we shall consider each of these in turn. As we shall show, one of the major advantages of our logical approach to computational types is that Moggi's three term equalities for $\lambda\mathrm{ML}_T$ are not arbitrary – they arise as natural proof-theoretic consequences of the normalisation (or cut-elimination) process.

4.1 Normalisation of natural deductions

The principal kind of normalisation step on natural deductions is a β reduction. This consists of removing the 'detour' which arises 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, Gallier (1993)).

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

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ A\end{matrix}}{\Diamond A}\,(\Diamond_{\mathscr{I}}) \qquad \begin{matrix}[A]\\ \vdots\\ \Diamond B\end{matrix}}{\Diamond B}\,(\Diamond_{\mathscr{E}}) \qquad \text{which normalises to} \qquad \begin{matrix}\vdots\\ [A]\\ \vdots\\ \Diamond B\end{matrix}$$

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 premise 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 $(\Diamond_{\mathscr{E}})$ rule (it must be modal) means that the rule gives rise to only one commuting conversion, *viz.*

- A deduction of the form

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

  commutes to

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

Clearly, the disjunction and falsity elimination rules also introduce commuting conversions but these follow the usual pattern which is well-described in several places (e.g. Girard, 1989).

4.2 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. The β-reduction rules on terms are shown in figure 5, whilst the interesting case of the commuting conversions induces the reduction rule

$$\mathsf{let}\, x \Leftarrow (\mathsf{let}\, y \Leftarrow u \,\mathsf{in}\, v)\,\mathsf{in}\, f \quad \to_c \quad \mathsf{let}\, y \Leftarrow u \,\mathsf{in}\, (\mathsf{let}\, x \Leftarrow v \,\mathsf{in}\, f)$$

Note particularly that two of the three equations for $\lambda\mathrm{ML}_T$ which we listed in §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.

$(\lambda x.u)v$	$\to_\beta$	$u[v/x]$
$\mathsf{fst}(u,v)$	$\to_\beta$	u
$\mathsf{snd}(u,v)$	$\to_\beta$	v
$\mathsf{case}\,\mathsf{inl}(e)\,\mathsf{of}\,\mathsf{inl}(x)\to f \mid \mathsf{inl}(y)\to g$	$\to_\beta$	$f[e/x]$
$\mathsf{case}\,\mathsf{inr}(e)\,\mathsf{of}\,\mathsf{inl}(x)\to f \mid \mathsf{inl}(y)\to g$	$\to_\beta$	$g[e/y]$
$\mathsf{let}\, x \Leftarrow \mathsf{val}(u)\,\mathsf{in}\, v$	$\to_\beta$	$v[u/x]$

Fig. 5. β-reduction rules for terms.

Proposition 5 (Subject Reduction)
If $\Gamma \vdash e : A$ and $e \to_{\beta c} e'$ then $\Gamma \vdash e' : A$.

Proof
Induction and Lemma 1. □

4.3 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 $\to_{\beta c}$ reduction relation, rather than just to the $\to_\beta$ 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 (Tait, 1967); the extension of Tait's method to commuting conversions as well as β-reductions is due to Prawitz (1971). 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 (1995b) to show strong normalisation for the linear term calculus.

If we wish to show strong normalisation for a language $\mathscr{L}_1$, and we already know that strong normalisation holds for another language $\mathscr{L}_2$, then it suffices to exhibit a translation $(\cdot)^\circ : \mathscr{L}_1 \to \mathscr{L}_2$ such that $\forall e, f \in \mathscr{L}_1 . e \to_1 f \Rightarrow e^\circ \to_2^+ f^\circ$, where $\to_1$ and $\to_2$ are the one-step reduction relations in $\mathscr{L}_1$ and $\mathscr{L}_2$ respectively. This is because any infinite reduction sequence in $\mathscr{L}_1$ would then induce an infinite reduction sequence in $\mathscr{L}_2$, contradicting strong normalisation for that language. Here we will take $\mathscr{L}_1$ to be the computational lambda calculus, with the $\to_{\beta c}$ reduction relation obtained by taking the precongruence closure of the rules shown in figure 5 (along with the commuting conversions), and $\mathscr{L}_2$ to be the simply typed lambda calculus with coproducts and the usual $\to_{\beta c}$ reduction relation. Note that $\mathscr{L}_2$ is just the largest sublanguage of $\mathscr{L}_1$ which does not contain the T type constructor or either of the let and val constructs.

Proposition 6
$\mathscr{L}_2$, the simply typed lambda calculus with coproducts and the $\to_{\beta c}$ reduction relation, is strongly normalising.

Proof
This was proved by Prawitz (1971). □

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 $\mathscr{L}_1$ types to $\mathscr{L}_2$ types:

$$\begin{aligned}
1^\circ &\overset{\text{def}}{=} 1\\
0^\circ &\overset{\text{def}}{=} 0\\
(A \mathbin{\triangle} B)^\circ &\overset{\text{def}}{=} A^\circ \mathbin{\triangle} B^\circ \quad (\text{for } \triangle \in \{\times, +, \to\})\\
(TA)^\circ &\overset{\text{def}}{=} 1 + A^\circ
\end{aligned}$$

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

$$\begin{array}{rcl@{\qquad}rcl}
*^\circ &\overset{\text{def}}{=}& * & x^\circ &\overset{\text{def}}{=}& x\\
(\lambda x{:}A.e)^\circ &\overset{\text{def}}{=}& \lambda x{:}A^\circ.e^\circ & (e\,f)^\circ &\overset{\text{def}}{=}& e^\circ\, f^\circ\\
(\mathsf{fst}(e))^\circ &\overset{\text{def}}{=}& \mathsf{fst}(e^\circ) & (\mathsf{snd}(e))^\circ &\overset{\text{def}}{=}& \mathsf{snd}(e^\circ)\\
(e,f)^\circ &\overset{\text{def}}{=}& (e^\circ, f^\circ) & (\nabla_A(e))^\circ &\overset{\text{def}}{=}& \nabla_{A^\circ}(e^\circ)\\
(\mathsf{inl}(e))^\circ &\overset{\text{def}}{=}& \mathsf{inl}(e^\circ) & (\mathsf{inr}(e))^\circ &\overset{\text{def}}{=}& \mathsf{inr}(e^\circ)
\end{array}$$

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

The following two lemmas are both easy inductions:

Lemma 7
If $\Gamma \vdash_{\lambda \mathrm{ML}_T} e : A$ then $\Gamma^\circ \vdash_{\mathscr{L}_2} e^\circ : A^\circ$.

Lemma 8
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 9
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 $e \rightarrow_{\beta c} f$ and uses Lemma 8. 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. □

Corollary 10 (*Strong Normalisation*)
The calculus $\lambda \mathrm{ML}_T$ with the $\rightarrow_{\beta c}$ reduction relation is strongly normalising.

Proof
By Propositions 9 and 6. □

4.4 Confluence

Given the property of strong normalisation it is relatively straightforward to show confluence. We employ Newman's lemma (Klop, 1992), which states that if a reduction system is weakly confluent and strongly normalising then it is confluent. We first need the following simple facts.

Lemma 11
1. If $u \rightarrow_{\beta c} u'$ then $u[v/x] \rightarrow^{*}_{\beta c} u'[v/x]$.
2. If $u \rightarrow_{\beta c} u'$ then $v[u/x] \rightarrow^{*}_{\beta c} v[u'/x]$.

We are now able to show weak confluence.

Proposition 12 (*Weak Confluence*)
If $t \rightarrow_{\beta c} t'$ and $t \rightarrow_{\beta c} t''$ then there exists a term s such that $t' \rightarrow^{*}_{\beta c} s$ and $t'' \rightarrow^{*}_{\beta c} s$.

Proof
This is proved by considering all critical pairs. The cases where the redexes are disjoint are trivial. If the redexes overlap, the proof uses Lemma 11 and congruence properties of the reduction relation. □

Corollary 13 (*Confluence*)
The reduction relation $\rightarrow_{\beta c}$ is confluent.

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. We shall not examine the cut elimination procedure in detail here, but merely state the result:

Theorem 14 (*Cut Elimination*)
There is a procedure which, given a sequent calculus proof π of a sequent $\Gamma \vdash A$ in CL-logic, yields a proof π' of the same sequent in which there are no occurrences of the *Cut* rule.

Proof

This was first proved by Curry (1952). The proof is a minor extension of the standard proof for ordinary propositional intuitionistic logic (see Gallier (1993), for example). □

Furthermore, we can annotate sequent calculus proofs with $\lambda\mathrm{ML}_T$ terms – the term associated with a sequent proof is the term annotating the corresponding natural deduction (so different sequent calculus proofs may be annotated with the same term). If one does this then all the term equalities of figure 5 also arise from the cut elimination process (though some cut elimination steps do not affect the term at all).

5 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) presentation of the sequent calculus. Given such a presentation, the traditional η-equality associated with the function-space constructor arises from reducing the derivation

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

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

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

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 in section 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.

It is worth stating here a fact which is essentially folklore for those working in categorical logic, but remains little known in other communities.[2] Reading off the η-rules for the T type and the coproduct from the categorical model yields not just the standard rules but more general rules. For example the more general η-rule for the T type is

$$\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f[\mathsf{val}(x)/z] =_{\eta_T} f[e/z].$$

Given the β-rules, these extended rules are sufficient to derive the commuting conversions. For example, the commuting conversion for the T type (as described in section 4.1)

$$\mathsf{let}\, x' \Leftarrow (\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f)\, \mathsf{in}\, g = \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, (\mathsf{let}\, x' \Leftarrow f\, \mathsf{in}\, g)$$

can be derived

$$\begin{array}{ll}
 & \mathsf{let}\, x' \Leftarrow (\mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, f)\, \mathsf{in}\, g \\
\equiv & (\mathsf{let}\, x' \Leftarrow (\mathsf{let}\, x \Leftarrow z\, \mathsf{in}\, f)\, \mathsf{in}\, g)[e/z] \\
=_{\eta_T} & \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, ((\mathsf{let}\, x' \Leftarrow (\mathsf{let}\, x \Leftarrow z\, \mathsf{in}\, f)\, \mathsf{in}\, g)[\mathsf{val}(x)/z]) \\
\equiv & \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, (\mathsf{let}\, x' \Leftarrow (\mathsf{let}\, x \Leftarrow \mathsf{val}(x)\, \mathsf{in}\, f)\, \mathsf{in}\, g) \\
=_{\beta} & \mathsf{let}\, x \Leftarrow e\, \mathsf{in}\, (\mathsf{let}\, x' \Leftarrow f\, \mathsf{in}\, g).
\end{array}$$

This technique can be applied to most connectives which have a parasitic formula. When considering reductions, rather than equality, however, it is still advantageous to separate commuting conversions from η-rules, particularly as the more general equational rules are non-local.

6 Categorical models

Since the computational lambda calculus was originally derived from categorical considerations (Moggi, 1989), we already know that a categorical model is a Cartesian Closed Category (CCC) with a strong monad. For completeness we shall sketch these categorical definitions.

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

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

[2] For example, the more general η-rule for the coproduct is given by Crole (1993).

Definition 2
A *strong monad* over a category $\mathscr{C}$ with finite products is a monad (T, η, μ), together with a natural transformation $\tau_{A,B} : A \times TB \to T(A \times B)$, satisfying 4 coherence conditions (for the details, see Moggi (1989), for example).

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

Figure 6 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. 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.

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

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

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

$$\frac{\Gamma \xrightarrow{e} 0}{\Gamma \xrightarrow{e} 0 \xrightarrow{!} A}\;(0_{\mathscr{E}}) \qquad \frac{\Gamma \xrightarrow{e} A}{\Gamma \xrightarrow{e} A \xrightarrow{inl} A + B}\;(+_{\mathscr{I}}) \qquad \frac{\Gamma \xrightarrow{e} B}{\Gamma \xrightarrow{e} B \xrightarrow{inr} A + B}\;(+_{\mathscr{I}})$$

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

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

Fig. 6. Modelling the computational lambda calculus.

Theorem 15 (*Moggi*)
CL-models provide a sound and complete interpretation of the computational lambda calculus.

7 Kripke models

The semantics most commonly assigned to modal logics are possible-world, or Kripke models. They provide interpretations of provability but not proofs, unlike categorical models. For completeness we provide a definition of a Kripke-style model which is sound and complete for CL-logic. Similar models have been considered by Fairtlough and Mendler (1995).

Definition 4
A *CL-Kripke model* is a tuple $(W, V, \leq, R, \models)$, where W is a non-empty set of possible worlds; V is a map which assigns to every propositional variable p a subset $V(p) \subseteq W$; R and $\leq$ are partial orders on W; $\models$ is a relation between worlds and formulae such that for all $w \in W$

- $w \models p$ iff $w \in V(p)$
- $w \models \top$
- $w \models \bot$ iff $\forall p.w \models p$
- $w \models A \wedge B$ iff $w \models A$ and $w \models B$
- $w \models A \vee B$ iff $w \models A$ or $w \models B$
- $w \models A \supset B$ iff $\forall v \geq w.v \models A$ implies $v \models B$
- $w \models \Diamond A$ iff $\forall v \geq w.\exists u.vRu$ and $u \models A$.

We also require that the two relations are hereditary

- If $w \models p$ and $w \leq v$ then $v \models p$,
- If $w \models A$ and wRv then $v \models A$.

Theorem 16
$\vdash A$ iff $\forall w.w \models A$.

Proof
By standard Henkin constructions (see, for example, Van Dalen, 1986). $\square$

It is an interesting question to ask how these Kripke models are related to the categorical models of the previous section. Indeed there appears to be more than one way to approach the question. One approach is given by Alechina *et al.* (1997), who demonstrate a natural method of finding a CL-Kripke model within a (categorical) CL-model.

8 Conclusions

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 (1952) and Fairtlough and Mendler (1995) have also discovered it. 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 lambda calculus is an useful metalanguage for describing a number of different notions of computation.

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 Benton *et al.* (1992) and Bierman and de Paiva (1996). In fact, there is a close relationship between CL-logic and intuitionistic linear logic. Any linear category (model for intuitionistic linear logic – see Benton *et al.* (1992) and Bierman (1995)), 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. More discussion of the relationship between intuitionistic linear logic and CL-logic may be found in Benton (1995a) and Benton and Wadler (1996), 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.

Acknowledgements

We are grateful for the enthusiasm and comments of Phil Wadler and the advice of the referees. This work was completed whilst Benton and de Paiva were at the University of Cambridge Computer Laboratory.

References

Alechina, N., de Paiva, V. C. V. and Ritter, E. (1997) Relating Categorical and Kripke Semantics for Intuitionistic Modal Logics. *Unpublished report*, School of Computer Science, University of Birmingham.

Benton, P. N. (1995a) A mixed linear and non-linear logic: Proofs, terms and models. *Proc. Conference on Computer Science Logic: Lecture Notes in Computer Science 933*. Springer-Verlag. (Expanded version available as Technical Report 352, University of Cambridge Computer Laboratory.)

Benton, P. N. (1995b) Strong normalisation for the linear term calculus. *J. Functional Programming*, **5**(1), 65–80.

Benton, P. N. and Wadler, P. (1996) Linear logic, monads and the lambda calculus. *Proceedings of Symposium on Logic in Computer Science.*

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

Bierman, G. M. (1995) What is a categorical model of intuitionistic linear logic? *Proceedings of*

2nd International Conference on Typed λ-calculi and Applications: Lecture Notes in Computer Science 902, pp. 78–93. Springer-Verlag.

Bierman, G. M. and de Paiva, V. C. V. (1996) Intuitionistic necessity revisited. *Technical Report CSR–96–10*, School of Computer Science, University of Birmingham.

Crole, R. L. (1992) Programming metalogics with a fixpoint type. *PhD thesis*, Computer Laboratory, University of Cambridge. (Revised version available as Technical Report 247.)

Crole, R. L. (1993) *Categories for Types.* Cambridge University Press.

Curry, H. B. (1952) The elimination theorem when modality is present. *J. Symbolic Logic*, **17**(4), 249–265.

Fairtlough, M. and Mendler, M. (1995). An intuitionistic modal logic with applications to the formal verification of hardware. *Proceedings of Conference on Computer Science Logic: Lecture Notes in Computer Science 933*. Springer-Verlag.

Gallier, J. (1993) Constructive logics part I: A tutorial on proof systems and typed λ-calculi. *Theor. Comput. Sci.*, **110**(2), 249–339.

Girard, J.-Y., Lafont, Y. and Taylor, P. (1989) *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science, vol. 7. Cambridge University Press.

Gordon, A. D. (1994) *Functional programming and input/output.* Distinguished Dissertation in Computer Science Series. Cambridge University Press.

Hodges, W. (1983) Elementary predicate logic. In: Gabbay, D. and Guenthner, F., editors, *Handbook of Philosophical Logic. Vol. 1*, Ch. 1, pp. 1–131. Reidel.

Howard, W. A. (1980) The formulae-as-types notion of construction. In: Hindley, J. R. and Seldin, J. P., editors, *To H.B. Curry: Essays on combinatory logic, lambda calculus and formalism.* Academic Press.

Klop, J. W. (1992) Term rewriting systems. In: Abramsky, S., Gabbay, D. M. and Maibaum, T. S. E., editors, *Handbook of Logic in Computer Science*, pp. 1–116. Oxford Science Publications.

Moggi, E. (1989) Computational lambda-calculus and monads. *Proceedings of Symposium on Logic in Computer Science*, pp. 14–23.

Moggi, E. (1991) Notions of computation and monads. *Infor. and Control*, **93**(1), 55–92.

Pitts, A. M. (1990) Evaluation logic. *Technical Report 198*, Computer Laboratory, University of Cambridge.

Prawitz, D. (1971) Ideas and results in proof theory. In: Fenstad, J. E., editor, *Proceedings of 2nd Scandinavian Logic Symposium*, pp. 235–307.

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

Tait, W. W. (1967) Intensional interpretation of functionals of finite type. *J. Symbolic Logic*, **32**, 198–212.

Van Dalen, D. (1986) Intuitionistic Logic. In: Gabbay, D. and Guenthner, F., editors, *Handbook of Philosophical Logic: Vol. 3*, Ch. 4, pp. 225–339. Reidel.

Wadler, P. (1990) Comprehending monads. *Proceedings of Conference on LISP and Functional Programming*, pp. 61–78.

Wadler, P. (1992) The essence of functional programming (invited talk). *Proceedings of Symposium on Principles of Programming Languages.*