A Curry-Howard Correspondence for a Canonical Classical Natural Deduction

Extending $\lambda\mu$ with First-Class Continuations

Alexander J. Summers

Imperial College London

Abstract. We present an extension of Parigot's $\lambda\mu$-calculus, which we call $\nu\lambda\mu$-calculus. Our motivation is to achieve a Curry-Howard correspondence with a canonical system of classical natural deduction, close to the original formulation due to Gentzen. At the same time, we aim for a notion of reduction which generalises those which exist already in the literature. We choose to employ both the implication and negation connectives in the logic. From a computational point of view, this involves introducing a separate binder for constructing continuations: we represent continuations as distinct first-class citizens.
We discuss a generalisation of the μ-reduction rules of existing calculi, and give new motivations for the reductions in terms of the type system. We state a principal typing property for the calculus, and show that control operators from the literature may be represented by the canonical $\nu\lambda\mu$-terms of the appropriate types. As a consequence of this approach, we discover that a version of Felleisen's $\mathcal{F}$ operator is a better candidate than $\mathcal{C}$ for a control operator corresponding to double-negation elimination in the logic. We compare with existing calculi and show that we can represent and generalise both the $\lambda\mu$ and $\lambda\mu\tilde{\mu}$-calculi, preserving typings and reductions.

1 Introduction

The Curry-Howard correspondence presents a strong relationship between the typed λ-calculus and a standard system of intuitionistic implicative natural deduction. Since the seminal paper by Griffin [10], relating the control operator of Felleisen's $\lambda\mathcal{C}$-calculus [8] to the double negation elimination rule of classical natural deduction, many calculi have been developed with a view to achieving a Curry-Howard correspondence with a classical logic.

Although the original Curry-Howard correspondence was with a Gentzen-style logic (a natural deduction calculus), efforts to produce an analogous result for a classical logic have often involved a departure from Gentzen's formalisms. Taking the $\lambda\mu$-calculus of Parigot [12] as a well-studied example, its standard presentation relates to a logical formalism partly in a natural deduction style but with multiple conclusions and structural rules for manipulating them. In [13], Parigot explains how this logic can be related back to a usual (single conclusion) formalisation of natural deduction, in which negation and bottom ($\perp$) occur within sequents, as well as implication. However, the system inhabited is not a standard one for these connectives: in fact we will show that it can be seen as a restriction of the standard natural deduction calculus of this kind.

1.1 Generalising the Curry-Howard Correspondence

From a programming perspective, it is usually desirable to work with *untyped* calculi. It might seem that the move to an untyped setting eliminates any possibility of talking about a Curry-Howard correspondence, particularly since there will usually exist (untypeable) terms which do not represent proofs. However, we believe that an analogous idea exists for untyped calculi: that the typeable fragment of the calculus has a Curry-Howard correspondence with a logic, while the reduction rules of the calculus are exactly the untyped versions of those which make sense for the logic. For example, we would like to regard *untyped* λ-calculus as a calculus with a Curry-Howard correspondence, in this way.

1.2 Natural Deduction for Classical Logic

We give here for reference a fairly-standard set of Gentzen-style natural deduction rules for classical logic with the $\rightarrow$, $\neg$ and $\perp$ connectives.

Definition 1 (Classical Natural Deduction with $\rightarrow\ \neg\ \perp$). *Formulas (ranged over by A,B) are defined by the following grammar: $A, B\ ::=\ \perp \mid \varphi \mid \neg A \mid A{\rightarrow}B$ (in which φ ranges over an infinite set of atomic formulae).*

$$\frac{}{\Gamma, A \vdash A}\,(ax) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A{\rightarrow}B}\,({\rightarrow}\mathcal{I}) \qquad \frac{\Gamma \vdash A{\rightarrow}B \quad \Gamma \vdash A}{\Gamma \vdash B}\,({\rightarrow}\mathcal{E})$$

$$\frac{\Gamma, \neg A \vdash \perp}{\Gamma \vdash A}\,(PC) \qquad \frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A}\,(\neg\mathcal{I}) \qquad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp}\,(\neg\mathcal{E})$$

Gentzen [9] describes that classical natural deduction is obtained by taking the intuitionistic introduction/elimination rules for the connectives, and adding a rule with "special status" (not fitting the introduction/elimination pattern) to make the logic classical. In our case, the (PC) rule fulfils this role. Note that we omit the standard rule for $(\perp\mathcal{E})$ since it is subsumed by the (PC) rule.

1.3 Structure of the Paper

In Section 2 we recall the definitions of the $\lambda\mu$-calculus, and compare its logical counterpart as a natural deduction system with that of Definition 1. We identify various restrictions in place, and explain how these can be lifted. In Section 3 we introduce the $\nu\lambda\mu$-calculus, and give a detailed discussion of the design choices relating to the μ-reductions. Section 4 presents an investigation of possible representations of control operators in the calculus. In Section 5 we show that we can encode the $\lambda\mu\tilde{\mu}$ calculus of Curien and Herbelin [4] in the $\nu\lambda\mu$ calculus, preserving reductions and typings.

2 The $\lambda\mu$-calculus

The $\lambda\mu$-calculus was introduced by Parigot in [12], and has been extensively studied as a calculus relating to classical logic. We recall here the basic definitions.

Definition 2 **($\lambda\mu$ Syntax).** [1] *The syntax of $\lambda\mu$-terms is defined over two distinct infinite sets of variables (one of Roman letters $x, y, \ldots$ and one of Greek letters $\alpha, \beta, \ldots$) by the following syntax:*

$$M, N := x \mid \lambda x.M \mid M\ N \mid [\alpha]M \mid \mu\alpha.M$$

The reduction rules of the $\lambda\mu$-calculus rely on an additional special notion of substitution. $M\langle[\beta](M'\ N)/[\alpha]M'\rangle$ denotes the replacement of all subterms of M of the form $[\alpha]M'$ with the corresponding term $[\beta](M'\ N)$ (this is referred to as *structural substitution* in the literature). We assume in this paper all substitutions to be capture-free.

Definition 3 **($\lambda\mu$ Reductions).** *The reductions of the $\lambda\mu$-calculus are defined by the following rules:*

$$\begin{array}{lrcll}
(\beta) & (\lambda x.M)\ N & \to & M\langle N/x\rangle & \\
(\mu) & (\mu\alpha.M)\ N & \to & \mu\beta.M\langle[\beta](M'\ N)/[\alpha]M'\rangle & \\
(\mu r) & [\beta]\mu\alpha.M & \to & M\langle\beta/\alpha\rangle & \\
(\mu\eta) & \mu\alpha.[\alpha]M & \to & M & \textit{if } \alpha \notin M
\end{array}$$

Since we wish to discuss the logic underlying the $\lambda\mu$-calculus, we will also recall the basic type-assignment rules. We omit the rules for quantifiers, since they are not relevant to this work, and obscure the logic which corresponds with the syntax constructs.

Definition 4 **(Type Assignment for $\lambda\mu$).**

$$\frac{}{\Gamma, x:A \vdash_{\lambda\mu} x:A \mid \Delta}\,(ax) \qquad \frac{\Gamma, x:A \vdash_{\lambda\mu} M:B \mid \Delta}{\Gamma \vdash_{\lambda\mu} \lambda x.M : A\to B \mid \Delta}\,(\to\mathcal{I})$$

$$\frac{\Gamma \vdash_{\lambda\mu} M:A\to B \mid \Delta \quad \Gamma \vdash_{\lambda\mu} N:A \mid \Delta}{\Gamma \vdash_{\lambda\mu} M\ N : B \mid \Delta}\,(\to\mathcal{E})$$

$$\frac{\Gamma \vdash_{\lambda\mu} M:\bot \mid \alpha:A, \Delta}{\Gamma \vdash_{\lambda\mu} \mu\alpha.M : A \mid \Delta}\,(\mu) \qquad \frac{\Gamma \vdash_{\lambda\mu} M:A \mid \Delta}{\Gamma \vdash_{\lambda\mu} [\alpha]M : \bot \mid \alpha:A, \Delta}\,(n)$$

The logic underlying this type system is *not* an example of a standard natural deduction calculus: the inference rules corresponding to the constructs $\mu\alpha.M$ and $[\alpha]M$ are presented as structural rules, which allow one to manipulate a collection of conclusions.These rules do not fit into the introduction/elimination scheme usual for natural deduction rules.[2] The original intention of the natural deduction style (which was to follow natural argument as much as possible) no longer applies in an obvious way to

[1] In the original presentation of [12], the syntax of the λ-calculus was extended only with terms of the form $\mu\alpha.[\beta]M$. Various subsequent work (for example [6, 3, 11, 17]) has involved separating these as constructs; we adopt this approach.

[2] Ong and Stewart [11] present the typing rules for these two constructs as an introduction/elimination pair for the connective $\bot$. However, these rules do not form such a pair in the sense made explicit by Prawitz [16].

this logic. On the other hand, Parigot shows how to relate the logic back to a usual presentation of natural deduction in [12], in which negation occurs within types, as well as implication and bottom ($\bot$). This is achieved by replacing each multiple-conclusion sequent $\Gamma \vdash_{\lambda\mu} M : A \mid \Delta$ by the single-conclusion sequent $\Gamma, \neg\Delta \vdash M : A$, in which $\neg\Delta = \{\alpha : \neg A \mid \alpha : A \in \Delta\}$. Under this transformation, the type-assignment rules become the following:

$$\frac{}{\Gamma, \neg\Delta, x : A \vdash x : A}\,(ax) \qquad \frac{\Gamma, \neg\Delta, x : A \vdash M : B}{\Gamma, \neg\Delta \vdash \lambda x.M : A \to B}\,(\to\mathcal{I})$$

$$\frac{\Gamma, \neg\Delta \vdash M : A \to B \quad \Gamma, \neg\Delta \vdash N : A}{\Gamma, \neg\Delta \vdash M\, N : B}\,(\to\mathcal{E})$$

$$\frac{\Gamma, \neg\Delta, \alpha : \neg A \vdash M : \bot}{\Gamma, \neg\Delta \vdash \mu\alpha.M : A}\,(\mu) \qquad \frac{\Gamma, \neg\Delta \vdash M : A}{\Gamma, \neg\Delta, \alpha : \neg A \vdash [\alpha]M : \bot}\,(n)$$

In this way, the $\lambda\mu$-calculus can be seen to have a Curry-Howard correspondence with a restricted version of the natural deduction system of Definition 1:

1. Assumptions are divided into two ‘classes’ (the two classes of variables in $\lambda\mu$). In this discussion, we will refer to these as ‘usual’ and ‘special’ assumptions.
2. The (PC) rule is restricted to bind only ‘special’ assumptions in its premise (this corresponds to the μ-binding of Greek variables).
3. The $(\neg\mathcal{E})$ rule is restricted to allow only axioms to occur as the first (major) premise, and these axioms may only feature ‘special’ assumptions (this corresponds to only allowing Greek variables to occur in the position of α in $[\alpha]M$).
4. ‘Special’ assumptions may not be used in any other way (Greek variables do not occur in the position of Roman variables).
5. The $(\neg\mathcal{I})$ rule is removed (no syntax construct is present to ‘inhabit’ this rule).

These restrictions do not seem very intuitive from the point of view of the logic. Ariola and Herbelin argue in [1] that the $\lambda\mu$ calculus corresponds with ‘minimal classical logic’[3]. They define an extension of $\lambda\mu$, adding a special syntax construct $[tp]M$, where *tp* acts as a ‘continuation constant’. In logical terms, the new construct corresponds with an explicit $(\bot\mathcal{E})$, and they then show that full classical provability is achieved. It seems surprising that the addition of the $(\bot\mathcal{E})$ rule to the logic provides any additional strength in terms of provability, since this rule is (in a standard natural deduction setting) subsumed by the (PC) rule, which is already inhabited by the μ-binding construct. This apparent inconsistency stems from the fact that the presentation of the type system of $\lambda\mu$ used in their work is quite different from a usual natural deduction presentation. In terms of *provability*, the apparent ‘gap’ in the original system could be resolved simply by interpreting an empty stoup as a stoup with type $\bot$ inside. In fact, this is essentially the approach used in [17, 3].

[3] Minimal classical logic is defined in [1] as minimal logic extended with Pierce’s law $((A\to B)\to A)\to A$ but without the rule $(\bot\mathcal{E})$ (which is not derivable in this logic).

Although completeness from a provability perspective is a definite requirement in order to consider a calculus to represent full classical logic, it is not entirely sufficient. Since we interpret "proofs as programs", it is the *proofs* that give us our computational objects, and the proof reductions which essentially specify the possible computational behaviour. Therefore, in order to speak about a Curry-Howard correspondence with full classical logic, as well as ensuring that all valid formulas are provable we should be concerned that all 'interesting' proofs of these formulas are represented.

In this paper, we consider each of the five restrictions identified above, and make appropriate additions and alterations to the $\lambda\mu$-calculus so that they can be lifted, with the aim of restoring a Curry-Howard correspondence with a Gentzen-style natural deduction system. In this way, we obtain a calculus still much in the spirit of the $\lambda\mu$-calculus, but with a richer and more expressive syntax:

1. The two classes of variables are collapsed into a single set of (Roman) variables.
2. μ-binders now bind the usual term variables: terms of the form $\mu x.M$ are allowed.
3. Terms of the form $[M]N$ are allowed (i.e. there is no restriction on the term M).
4. Greek variables no longer occur in the syntax at all.
5. A syntax construct inhabiting the $(\neg\mathcal{I})$ rule is added, which involves a third kind of binder. We write these new terms as $\nu x.M$ (in which x is bound)[4].

3 The $\nu\lambda\mu$-calculus

Definition 5 (Syntax). *The syntax of the $\nu\lambda\mu$-calculus is defined over the set of variables $x, y, z, \ldots$ as follows:*

$$M, N \ := \ x \mid \lambda x.M \mid M\,N \mid \mu x.M \mid \nu x.M \mid [M]N$$

The typeable fragment of the syntax gives a term representation for the classical natural deduction system of Definition 1. This can be seen by the following type assignment system for the calculus.

Definition 6 (Type assignment for $\nu\lambda\mu$-calculus). *Types are defined over an infinite set of atomic types $\varphi_1, \varphi_2, \ldots$ by the following syntax: $A, B \ := \ \bot \mid \varphi \mid \neg A \mid A \to B$. Γ represents a finite set of statements $\{x : A, \ldots\}$ in which no variable may occur more than once. We write $\Gamma, x : A$ to mean $\Gamma \cup \{x : A\}$, respecting this restriction.*
We write $\Gamma \vdash M : A$ to mean that there is a derivation using the rules below with this statement as the last line. The type assignment rules are as follows:

$$\frac{}{\Gamma, x : A \vdash x : A}\ (ax) \qquad \frac{\Gamma, x : \neg A \vdash M : \bot}{\Gamma \vdash \mu x.M : A}\ (PC)$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ (\to\mathcal{I}) \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}\ (\to\mathcal{E})$$

$$\frac{\Gamma, x : A \vdash M : \bot}{\Gamma \vdash \nu x.M : \neg A}\ (\neg\mathcal{I}) \qquad \frac{\Gamma \vdash M : \neg A \quad \Gamma \vdash N : A}{\Gamma \vdash [M]N : \bot}\ (\neg\mathcal{E})$$

[4] We are aware that the notation ν for a binder is already overloaded in the literature, but have not found a satisfactory alternative, and hope this does not cause confusion.

We can also define a principal typing algorithm (in terms of the usual notion of substitution on atomic types) for the calculus, which generalises the standard result for the λ-calculus.

Proposition 1 (Principal typings for $\nu\lambda\mu$). *There exists an algorithm which, given any $\nu\lambda\mu$ term M, either fails (in which case M is not typeable at all) or produces a pair $\langle\Gamma,A\rangle$ such that:*

1. $\Gamma \vdash M : A$
2. *If $\Gamma' \vdash M : A'$ then there is a substitution S such that $S(\Gamma) \subseteq \Gamma'$ and $S(A) = A'$*

Negation is interpreted as the type for continuations; the type $\neg A$ represents a continuation, which expects an argument of type A, but does not return anything. One could compare this with the type $A\to\bot$. The reduction rules (λ) and (ν) below are the standard logical rules for the $\to$ and $\neg$ connectives[5].

$$\begin{array}{ll} (\lambda) & (\lambda x.M)\, N \;\to\; M\langle N/x\rangle \\ (\nu) & [\nu x.M]N \;\to\; M\langle N/x\rangle \end{array}$$

A ν-bound term provides an explicit representation for constructing a continuation. Terms of the form $[M]N$ represent the application of the continuation M to the argument N.

It is natural to ask why we represent negation explicitly in the type language, instead of using a type $A\to\bot$ instead. This is because the ability to distinguish a continuation from a function in terms of the (untyped) syntax is a useful feature when defining the μ-reductions. In particular, a μ-reduction in a function application behaves differently from a μ-reduction in a continuation application, as will become clear in the forthcoming discussions. Historically, the separation of continuations from functions has also been seen as desirable in more practical settings, for example in Standard ML[7]:"Another way of typing continuations, and the one currently adopted in Standard ML of New Jersey, is to abandon the view that continuations are functions in the ordinary sense …In practice, it is useful to be able to easily distinguish the invocation of a continuation from the application of a function."

It will facilitate the discussions of μ-binding to be able to explicitly describe the context in which a μ-bound term occurs; by this we mean a surrounding term with a 'hole', as is described by the following definition:

Definition 7 (Contexts). *Contexts C are defined using the $\nu\lambda\mu$-syntax, and the special symbol $\bullet$ used to denote the (unique) 'hole' in the term:*

$$C ::= \bullet \mid C\, M \mid M\, C \mid \lambda x.C \mid [C]M \mid [M]C \mid \nu x.C \mid \mu x.C$$

We write $C\{M\}$ to denote the insertion of the term M into the 'hole' of C.

We note that there is a close relationship between contexts of type $\bot$ and ν-bound terms in our syntax. In fact, for any context C of type $\bot$ and with a 'hole' of type A, the term $\nu x.C\{x\}$ (where x is chosen to be a fresh variable) is of type $\neg A$: the hole in the context is abstracted to form an explicit continuation.

[5] The rule (λ) is of course the usual (β) rule of the λ-calculus; we rename it here only because *both* of the rules here are similar to (β).

3.1 μ-reductions

We aim now to describe a new intuition for the meaning of μ-bound terms in our syntax, and to use this to derive a set of reduction rules, which will turn out to generalise those existing in the literature.

Parigot notes in [12] that if one wishes to avoid the special structural substitutions of the form $M\langle[\beta](M'\ N)/[\alpha]M'\rangle$, one can employ the following rule instead of that included in Definition 3 which, up to an extra reduction, has the same effect.

Definition 8 (Alternative formulation of $\lambda\mu$ reduction rule (μ)).

$$(\mu x.M)\ N \to \mu y.M\langle\nu z.[y](z\ N)/x\rangle$$

The advantage with this alternative formulation (without structural substitution), is that it works as a reduction rule when x is allowed to be employed in arbitrary positions in M. In fact, the rule of this definition is adopted as a μ-reduction rule for $\nu\lambda\mu$, as we will shortly explain.

In order to illustrate our intended meaning of a term $\mu x.M$, we find it helpful to examine the form which the body M is allowed to (typeably) take. According to the type system, the typing of M must be of the form: $\Gamma, x : \neg A \vdash M : \bot$. That is, M must be a term of type $\bot$, which itself has a free variable x of type $\neg A$. We can view M as requiring a term of type $\neg A$ (that is, a continuation with a 'hole' of type A) to replace the variable x with. Suppose we were to introduce terms of the form $\mu x.M$, and define (arbitrarily, for the time being) that such a term will have type A. Considering the kind of context C in which such a term can be (typeably) placed, it must be a term with a 'hole' of type A itself! If we can abstract this context as a continuation, then this will produce a term suitable to be substituted for the μ-bound x. We regard this to be the role of the μ-binding in the calculus: it is a term which behaves by capturing and abstracting its surrounding context, and binding it to a variable. Note that this idea is closely related to Bierman's abstract machine for the $\lambda\mu$-calculus, as described in [3].

There are some problems to consider with this point of view. Firstly, in talking about the whole context in which a term is inserted, we lose the notion of local, compatible reductions. This problem is solved by making a μ-reduction consume only the immediate context; i.e. one level further out in the syntax. Secondly, if this context is not itself of type $\bot$, then it does not represent a continuation in the way we would like. Suppose the context is of type B, then we now require a continuation of type $\neg B$ before we can obtain $\bot$. This need for a new continuation can be represented by introducing a new μ-binding appropriately. With the idea in mind that μ-binders should propagate outward through the syntax consuming their surrounding contexts, we consider reduction rules which may achieve this. It turns out that the reduction rules which follow from this intuition subsume the existing μ reduction rules in the literature.

Firstly, let us consider the case of μ-binding occurring on either side of a continuation application $[M]N$. This term is of type $\bot$, and as we have discussed previously it is possible to form an abstraction of a context of type $\bot$ by ν-binding over the 'hole'. For example, consider the case of μ-binding on the left of such an application: $[\mu x.M]N$. The immediate context in which the μ-bound term occurs is $[\bullet]N$ (of type $\bot$). By ν-abstracting over the 'hole' in this context, we obtain the continuation $\nu z.[z]N$, which can be inserted in place of x. Therefore we obtain the rule:

$$[\mu x.M]N \to M\langle \nu z.[z]N/x\rangle$$

In the symmetric case, we observe that an even simpler rule can be defined, since the μ-bound term has a continuation directly applied to it:

$$[N]\mu x.M \to M\langle N/x\rangle$$

In the case where μ-binding occurs in a functional application, only a partial continuation can be formed: since the context has a non-bottom type then a further continuation is required to consume this. For example, for a redex of the form $(\mu x.M)\ N$, the immediate context is $\bullet\ N$. We cannot typeably ν-bind this context as it stands, but can instead introduce a new variable y for the remaining continuation required: $\nu z.[y](z\ N)$ represents a 'partial continuation'. In this way, we obtain the well-known reduction rules from $\lambda\mu$ and related work [14, 2, 11]):

$$\begin{array}{rcl} (\mu x.M)\ N & \to & \mu y.M\langle \nu z.[y](z\ N)/x\rangle \\ N\ (\mu x.M) & \to & \mu y.M\langle \nu z.[y](N\ z)/x\rangle \end{array}$$

The case of a term $\mu x.M$ occurring under another ν or μ binder can be seen to be degenerate: since the type of the term must necessarily be $\bot$, the type of the assumption x must be $\neg\bot$, i.e. the term 'seeks' a trivial continuation. In fact, we deal with this case by removing the μ-binder, and substituting for x the canonical term of type $\neg\bot$, being $\nu z.z$.

It turns out that no general reduction rule can be defined to allow a μ-binder to 'escape' a λ binder, without introducing new μ-binders further down in the structure of the term. This leads to non-termination, therefore such a rule must be abandoned. Instead, inspired by the $\lambda\mu\tilde{\mu}$-calculus of Curien and Herbelin [4], we discovered that a better reduction behaviour could be achieved by changing the rule (λ); that is, we modify the original reduction rule of the λ-calculus. We replace the rule (λ) with the following (in which y is a fresh variable):

$$(\lambda') \quad (\lambda x.M)\ N \to \mu y.[\nu x.[y]M]N$$

This rule looks rather alien to the notion of reduction from the λ-calculus, however we observe that in the presence of the rule $(\mu\eta)$, the original reduction rule can still be simulated:

$$\begin{array}{rcll} (\lambda x.M)\ N & \to & \mu y.[\nu x.[y]M]N & (\lambda') \\ & \to & \mu y.[y]M\langle N/x\rangle & (\nu) \\ & \to & M\langle N/x\rangle & (\mu\eta) \end{array}$$

What then, is the advantage of this rule? It turns out that it allows μ-binders to escape λ-binders in the special case where the λ-bound term is applied to an argument:

$$\begin{array}{rcll} (\lambda x.\mu z.M)\ N & \to & \mu y.[\nu x.[y]\mu z.M]N & (\lambda') \\ & \to & \mu y.[\nu x.M\langle y/z\rangle]N & (\mu\neg_2) \\ & = & \mu z.[\nu x.M]N & (\alpha\ \textit{conversion}) \end{array}$$

We therefore adopt this rule, along with $(\mu\eta)$, and this gives a final set of reduction rules as follows:

Definition 9 (Reduction rules for the $\nu\lambda\mu$-calculus).

$$\begin{array}{llcll}
(\lambda') & (\lambda x.M)\ N & \rightarrow & \mu y.[\nu x.[y]M]N & \\
(\nu) & [\nu x.M]N & \rightarrow & M\langle N/x\rangle & \\
(\mu{\rightarrow_1}) & (\mu x.M)\ N & \rightarrow & \mu y.M\langle \nu z.[y](z\ N)/x\rangle & \\
(\mu{\rightarrow_2}) & N\ (\mu x.M) & \rightarrow & \mu y.M\langle \nu z.[y](N\ z)/x\rangle & \\
(\mu\neg_1) & [\mu x.M]N & \rightarrow & M\langle \nu z.[z]N/x\rangle & \\
(\mu\neg_2) & [N]\mu x.M & \rightarrow & M\langle N/x\rangle & \\
(\mu\nu) & \nu y.\mu x.M & \rightarrow & \nu y.M\langle \nu z.z/x\rangle & \\
(\mu\mu) & \mu y.\mu x.M & \rightarrow & \mu y.M\langle \nu z.z/x\rangle & \\
(\mu\eta) & \mu x.[x]M & \rightarrow & M & \textit{if}\ x \notin M
\end{array}$$

As usual, we define our reduction relation $\rightarrow$ to be the reflexive, transitive, compatible closure of the above rules. Although we have now formally replaced the (λ) rule previously discussed, we will treat it as a derived rule in order to shorten the examples we require later. We have a subject reduction property for this set of rules:

Proposition 2 (Subject reduction). *If $\Gamma \vdash M : A$ and $M \rightarrow N$ then $\Gamma \vdash N : A$.*

The reduction rules are non-confluent, with several critical pairs, for example between the rules (ν) and $(\mu\neg_2)$, and between $(\mu{\rightarrow_1})$ and $(\mu{\rightarrow_2})$. We regard this non-confluence as an inherent property of a sufficiently general framework based on classical logic. While we aim to define confluent sub-calculi as future work, for the present we are interested in the general approach. As we shall show later, our calculus is able to simulate a general cut-elimination procedure for classical sequent calculus, which is itself non-confluent.

As an example of the differences between the $\nu\lambda\mu$ and $\lambda\mu$ calculi, we consider terms inhabiting the type $\neg\neg A{\rightarrow}A$. In [1], issues regarding the inhabitation of this type in $\lambda\mu$ are discussed: "In Parigot's style. . . [this type]. . . is represented with the term $\lambda y.\mu\alpha.[\gamma](y\ (\lambda x.\mu\delta.[\alpha]x))$". This term is significantly more complex than might be expected of a canonical term inhabiting this type. This is improved upon by allowing the body of a μ-abstraction to be any term of type $\perp$ [17, 3], permitting a term such as $\lambda y.\mu\alpha.(y\ (\lambda x.[\alpha]x))$ instead. However, the canonical natural deduction proof of $\neg\neg A{\rightarrow}A$ yields the $\nu\lambda\mu$ term $\lambda y.\mu x.[y]x$. This cannot be a $\lambda\mu$ term, because the μ-bound variable is used as a standard term variable. Therefore we have a simpler representation than in other comparable calculi. This term has behaviour similar (although not identical, as we shall see) to the Felleisen's $\mathcal{F}$ operator: when applied to an argument (bound to y), it captures the outlying context (binding it to x), and then passes x to y in a continuation application. The representation of control operators will be discussed further in the next section.

4 Control Operators

In this section we examine possible representations for various control operators from the literature in the $\nu\lambda\mu$-calculus. We show that the representations of these operators can usually be deduced from their type: given a control operator with a type A, the $\nu\lambda\mu$equivalent can be found by inhabiting the simplest proof of the formula A.

We introduce two more-specific notions of context.

Definition 10 (Applicative Contexts). *We define applicative contexts C_A as follows:*

$$C_A \ ::= \ \bullet\, M \ \mid \ M\, \bullet \ \mid \ C_A\, M \ \mid \ M\, C_A$$

We define continuation-delimited applicative contexts C_C *by the following grammar:*

$$C_C \ ::= \ [\bullet]M \ \mid \ [C_A]M \ \mid \ [M]C_A$$

Note that we do not include the empty context as an applicative context, or the context $[M]\bullet$ as a continuation-delimited one. The behaviours described below are different for these special cases, and it is easier to rule them out for the general discussions.

Recall that μ-bound terms propagate outwards through applicative contexts, consuming and binding their context as a continuation. When the level of a continuation application is reached, this behaviour terminates (the μ-binder is removed). We can now give a characterisation of this kind of continued μ reduction by the following operational-style rules:

Proposition 3 (Characterisation of repeated μ-reductions). *The following reductions are derivable in the $\nu\lambda\mu$-calculus:*

$$\begin{array}{l} C_A\{(\mu x.M)\} \rightarrow \mu y.M\langle \nu z.[y]C_A\{z\}/x\rangle \\ C_C\{(\mu x.M)\} \rightarrow M\langle \nu z.C_C\{z\}/x\rangle \end{array}$$

Note that these reductions are certainly not, in general, deterministic. Although the rules above have the feel of an operational semantics, this is definitely not the intention. Instead, we wish to facilitate comparisons with the operational rules associated with control operators. As a space-saving exercise, we will now summarise the behaviour of the control operators we will refer to in this section. Where appropriate according to the types, we have made use of the extra syntax constructs available in $\nu\lambda\mu$.

Name	Syntax	Reduction Behaviour
abort	$\mathcal{A}(M)$	$C_A\{\mathcal{A}(M)\} \rightarrow M$
call/cc	$\mathcal{K}$	$C_A\{(\mathcal{K}\, M)\} \rightarrow C_A\{(M\ (\lambda x.\mathcal{A}(C_A\{x\})))\}$
control	$\mathcal{C}$	$C_A\{(\mathcal{C}\, M)\} \rightarrow [M]\nu x.\mathcal{A}(C_A\{x\})$
prompt	$\#$	$\#(V) \rightarrow V$
$\mathcal{F}$	$\mathcal{F}$	$C\{\#(C_A\{(\mathcal{F}\, M)\})\} \rightarrow C\{\#(([M]\nu x.C_A\{x\}))\}$

As a first control operator to study, we consider the $\mathcal{A}$ ('abort') operator. Since the sequent $\bot \vdash A$ is classically valid, we can find a proof for it in our natural deduction calculus, and inhabit with a $\nu\lambda\mu$-term. In fact, we require essentially a $(\bot\mathcal{E})$ step here, which can be simulated by the (PC) rule: we define $\mathcal{A}_{\nu\lambda\mu}(M) = \mu x.M$ with $x \notin M$[6] To check this has the correct operational behaviour, we make use of Proposition 3:

$$\begin{array}{rcll} C_A\{\mathcal{A}_{\nu\lambda\mu}(M)\} & = & C_A\{(\mu x.M)\} & (\lambda) \\ & \rightarrow & \mu y.M\langle \nu z.[y]C_A\{z\}/x\rangle & (\textit{Proposition 3}) \\ & = & \mu y.M & (x \notin M) \end{array}$$

[6] We choose not to represent $\mathcal{A}_{\nu\lambda\mu}$ as a term itself, but rather as an operator with an argument. We could instead have used the term $\lambda y.\mu x.y$ of type $\bot\rightarrow A$.

We have not reached M by this reduction, although the context C_A has been discarded, as expected. In fact, the persistence of the μ-binder in the result avoids the problem with subject reduction which is seen with the operational definition of the behaviour of $\mathcal{C}$ (as is discussed by Griffin [10]). On the other hand, if the context employed were a continuation-delimited one, the full effect of the abort would be seen (the reader may wish to verify that $C_C\{\mathcal{A}_{\nu\lambda\mu}(M)\} \to M$).

Consider next the *call/cc* operator (hereafter denoted by $\mathcal{K}$). The operator $\mathcal{K}$ can be typed with Pierce's Law $(((A{\to}B){\to}A){\to}A)$. By seeking the canonical natural deduction proof matching this type, we obtain the $\nu\lambda\mu$term $\mathcal{K}_{\nu\lambda\mu} = \lambda x.\mu y.[y](x\ (\lambda z.\mu w.[y]z))$. The behaviour of this term in a continuation delimited context can be seen to be exactly that of the original:

$$\begin{array}{ll}
C_C\{(\mathcal{K}_{\nu\lambda\mu}\ M)\} & \\
\ \to C_C\{(\mu y.[y](M\ (\lambda z.\mu w.[y]z)))\} & (\lambda) \\
\ \to [\nu x.C_C\{x\}](M\ (\lambda z.\mu w.[\nu x.C_C\{x\}]z)) & (5.2) \\
\ \to C_C\{(M\ (\lambda z.\mu w.[\nu x.C_C\{x\}]z))\} & (\nu) \\
\ \to C_C\{(M\ (\lambda z.\mu w.C_C\{z\}))\} & (\nu) \\
\ = C_C\{(M\ (\lambda z.\mathcal{A}_{\nu\lambda\mu}(C_C\{z\})))\} & (\nu)
\end{array}$$

Remark 1. Note that this behaviour is seen for a continuation-delimited context C_C: in fact what we have here is a delimited version of the *call/cc* operator, since it only captures the surrounding context up to the next continuation application.

Griffin also studies the $\mathcal{C}$ operator and shows that, in some situations, it can be typed as a double-negation-elimination operator. However, there are some problems with this view, since the typing does not match up with the general reduction rules for the operator. If $\mathcal{C}$ is accepted to have type $\neg\neg A{\to}A$, then subject reduction is violated by the reduction rule in general[7]. The problem is that the right-hand side of the rule is necessarily of type $\bot$, whereas the left-hand side can be of whatever type the context C_A returns. Griffin proposes a solution to this problem by restricting the form of programs, so that the 'top level' can always be sure to have type $\bot$. Despite these difficulties, $\mathcal{C}$ is often considered to be the computational representation of double negation elimination.

In order to find an analogous operator in our calculus, we start by finding a natural deduction proof of $\neg\neg A{\to}A$. The simplest such proof yields the $\nu\lambda\mu$term $\lambda x.\mu y.[x]y$, which has the following behaviour in a continuation-delimited context:

$$\begin{array}{rlll}
C_C\{(\mathcal{C}_{\nu\lambda\mu}\ M)\} & = & C_C\{((\lambda x.\mu y.[x]y)\ M)\} & \\
& \to & C_C\{(\mu y.[M]y)\} & (\lambda) \\
& \to & ([M]y)\langle \nu z.C_C\{z\}/y\rangle & (\textit{Proposition 3}) \\
& \to & [M]\nu z.C_C\{z\} &
\end{array}$$

This does not appear to match the behaviour of the $\mathcal{C}$ operator, since no $\mathcal{A}$ steps occur in the redex. In fact, since we have derived our operator from the canonical proof of $\neg\neg A{\to}A$, this is not surprising: the occurrence $\mathcal{A}$ in $\mathcal{A}(x)$ is in fact of type $\bot{\to}\bot$, and so seems to be a redundant step from the point of view of the proof. This is discussed

[7] Strictly, this eliminates the possibility of seeing $\lambda\mathcal{C}$ as a calculus with a Curry-Howard correspondence, since there are reduction steps which do not correspond to valid proof reductions.

by Ariola and Herbelin in [1]: "...these steps are of type $\bot \to \bot$. Therefore it seems we have a mismatch. While the aborts are essential in the reduction semantics they are irrelevant in the corresponding proof." They criticise the work of Ong and Stewart [11] and of de Groote [5], since these works (which compare $\mathcal{C}$ with variants of $\lambda\mu$) do not include the abort steps in the reduction rules for $\mathcal{C}$.

We observe that the $\mathcal{F}$ operator can also be typed as a double negation elimination operator (i.e. given the type $\neg\neg A \to A$). There is no subject reduction conflict between this typing and the associated reduction rules. Furthermore, the reduction behaviour of the $\nu\lambda\mu$-term $\mathcal{C}_{\nu\lambda\mu}$ above (which we argue is a canonical term inhabiting this type) is actually closer to $\mathcal{F}$ than $\mathcal{C}$. Therefore, we believe that $\mathcal{F}$ is a *better* candidate than $\mathcal{C}$ to provide a Curry-Howard correspondence with a calculus with double-negation elimination. This explains why, in the work of de Groote, Ong and Stewart, it was found that better correspondences could be identified if the abort steps were removed from the reduction rules for $\mathcal{C}$; essentially they were employing a version of the $\mathcal{F}$ operator instead. A more complete investigation of the relationship between these operators and their representations in the $\nu\lambda\mu$-calculus will be the subject of future work.

5 The $\overline{\lambda}\mu\tilde{\mu}$-calculus

In this section, we show that we can encode the $\lambda\mu\tilde{\mu}$-calculus into $\nu\lambda\mu$. The $\lambda\mu\tilde{\mu}$-calculus [4] has a Curry-Howard correspondence with a modified version of Gentzen's LK (sequent calculus for classical logic).

Definition 11 **($\lambda\mu\tilde{\mu}$ Syntax).** *The syntax of the terms (ranged over by v), contexts (ranged over by e) and commands (ranged over by c) of the $\lambda\mu\tilde{\mu}$-calculus is specified by the following mutually-recursive definitions, in which x, y range over* term variables*, and α, β over* context variables*:*

$$c ::= \langle v|e\rangle \qquad v ::= x \mid (\lambda x.v) \mid (\mu\alpha.c) \qquad e ::= \alpha \mid (v\cdot e) \mid (\tilde{\mu}x.c)$$

The commands $\langle v|e\rangle$ correspond to cuts in the logic, and reduction induces a partial cut-elimination. Not all cuts are redexes in this calculus: the command $\langle x|\alpha\rangle$ is in normal form, for example.

Definition 12 **($\lambda\mu\tilde{\mu}$ Reductions).** *The reduction relation of the $\lambda\mu\tilde{\mu}$-calculus is defined to be the reflexive, transitive, compatible closure of the following rules:*

$$\begin{array}{lrcl} (\to') & \langle \lambda x.v_1|v_2\cdot e\rangle & \to & \langle v_2|\tilde{\mu}x.\langle v_1|e\rangle\rangle \\ (\mu) & \langle \mu\beta.c|e\rangle & \to & c\langle e/\beta\rangle \\ (\tilde{\mu}) & \langle v|\tilde{\mu}x.c\rangle & \to & c\langle v/x\rangle \end{array}$$

Compared with Gentzen's cut-elimination for the sequent calculus, the first of these reduction rules corresponds to one of the two bracketings of the usual logical reduction rule for implication. The dual idea is to allow the argument v_2 to be 'cut with' v_1 first, and then insert the result into the context e. This corresponds to a different bracketing of the two resulting cuts, and is less naturally expressible in the syntax of $\lambda\mu\tilde{\mu}$ because of the lack of an explicit name for the *output* of the function $\lambda x.v_1$. In fact, this cut-elimination step is not simulated by the reductions of $\lambda\mu\tilde{\mu}$.

We show that it is possible to encode the $\lambda\mu\tilde{\mu}$-calculus into $\nu\lambda\mu$, in such a way that reductions and typings are preserved. The key observation is that, while $\lambda\mu\tilde{\mu}$ distinguishes between terms and contexts, in the $\nu\lambda\mu$-calculus there is a construct present to explicitly represent continuations (or contexts), being the ν-binding.

Definition 13 (Encoding $\lambda\mu\tilde{\mu}$). *We encode $\lambda\mu\tilde{\mu}$ by the following mapping (which applies to terms, contexts, and commands alike; one could consider this three mutually-recursive definitions). In the case for encoding a context of the form $v\cdot e$, we assume y to be a fresh variable.*

$$\begin{array}{rclrcl}
\overline{\langle v|e\rangle} &=& [\overline{e}]\overline{v} & & & \\
\overline{x} &=& x & \overline{\alpha} &=& \alpha \\
\overline{\lambda x.v} &=& \lambda x.\overline{v} & \overline{v\cdot e} &=& \nu y.[\overline{e}](y\ \overline{v}) \\
\overline{\mu\alpha.c} &=& \mu\alpha.\overline{c} & \overline{\tilde{\mu}x.c} &=& \nu x.\overline{c}
\end{array}$$

Note that commands are encoded exactly as continuation applications; the context e takes the role of the continuation whereas the term v is the argument to the continuation.

Proposition 4 (Simulation of $\lambda\mu\tilde{\mu}$).

1. *The mapping $\overline{\cdot}$ is an injection.*
2. *(a) For any command c, if $c : \Gamma \vdash_{\overline{\lambda\mu\tilde{\mu}}} \Delta$ then $\Gamma, \neg\Delta \vdash \overline{c} : \bot$.*
   (b) For any term v, if $\Gamma \vdash_{\overline{\lambda\mu\tilde{\mu}}} v : A \mid \Delta$ then $\Gamma, \neg\Delta \vdash \overline{v} : A$.
   (c) For any context e, if $\Gamma \mid e : A \vdash_{\overline{\lambda\mu\tilde{\mu}}} \Delta$ then $\Gamma, \neg\Delta \vdash \overline{e} : \neg A$.
3. *For any $\lambda\mu\tilde{\mu}$ commands c_1,c_2 (or terms, or contexts) , if $c_1 \to c_2$ then $\overline{c_1} \to \overline{c_2}$.*

Proof. 1. By induction on the definition of the encoding.
2. By simultaneous induction on the structures of terms, contexts and commands.
3. We show here only the case for the $(\to')$ rule (the others being simpler). We have:

$$\begin{array}{rcll}
\overline{\langle \lambda x.v_1 | v_2\cdot e\rangle} &=& [\nu y.[\overline{e}](y\ \overline{v_2})]\lambda x.\overline{v_1} & \\
&\to& [\overline{e}]((\lambda x.\overline{v_1})\ \overline{v_2}) & (\nu) \\
&\to& [\overline{e}]\mu z.[\nu x.[z]\overline{v_1}]\overline{v_2} & (\lambda') \\
&\to& [\nu x.[\overline{e}]\overline{v_1}]\overline{v_2} & (\mu\neg_2) \\
&=& \overline{\langle v_2 | \tilde{\mu}x.\langle v_1|e\rangle\rangle} &
\end{array}$$

In [4], the following remark is made: “Without logical or computational loss, one may force the body of a λ-abstraction to have the form $\mu\alpha.c\ldots$”. If this approach were taken, we suggest the rule $(\to')$ could be replaced by the following rule:

$$(\to'')\ \ \langle \lambda x.\mu\alpha.c | v\cdot e\rangle \to \left\{\begin{array}{c} \langle v|\tilde{\mu}x.\langle \mu\alpha.c|e\rangle\rangle \\ \textit{or} \\ \langle \mu\alpha.\langle v|\tilde{\mu}x.c\rangle|e\rangle \end{array}\right\}$$

In this way, the correspondence with the logical cut elimination rule for implication would be restored. However, we note further that this rule is already fully simulated in the $\nu\lambda\mu$-calculus. The first alternative is reachable as demonstrated in the proof above, but the second can also be achieved as follows:

$$\begin{array}{rcll}
\overline{\langle \lambda x.\mu\alpha.c | v\cdot e\rangle} & = & [\nu y.[\overline{e}](y\,\overline{v})]\lambda x.\mu\alpha.\overline{c} & \\
& \to & [\overline{e}]((\lambda x.\mu\alpha.\overline{c})\,\overline{v}) & (\nu) \\
& \to & [\overline{e}]\mu z.[\nu x.[z]\mu\alpha.\overline{c}]\overline{v} & (\lambda') \\
& \to & [\overline{e}]\mu z.[\nu x.\overline{c}\langle z/\alpha\rangle]\overline{v} & (\mu\neg_2) \\
& = & [\overline{e}]\mu\alpha.[\nu x.\overline{c}]\overline{v} & (\alpha\ \textit{conversion}) \\
& = & \overline{\langle \mu\alpha.\langle v | \tilde{\mu}x.c\rangle | e\rangle} &
\end{array}$$

The reductions in $\nu\lambda\mu$ are therefore powerful enough to simulate a stronger notion of reduction than that already present in $\lambda\mu\tilde{\mu}$, and a 'full' cut elimination for the sequent calculus.

A natural question to ask is whether a simple translation back from $\nu\lambda\mu$ to $\lambda\mu\tilde{\mu}$ is possible. We conjecture that one does not exist, since there are not constructs to represent an explicit negation in $\lambda\mu\tilde{\mu}$.

6 Conclusions and Future Work

We have presented the $\nu\lambda\mu$-calculus, and shown how its definitions were derived. The resulting calculus appears to be very flexible, in that it can naturally express many other calculi and programming concepts. The calculus includes $\lambda\mu$ as a subset, but provides a Curry-Howard correspondence with a standard formulation of classical natural deduction. It can also simulate cut elimination in a classical sequent calculus.

One interesting area of future work is the extension of our calculus with an explicit substitution operator; we conjecture that step-by-step sequent calculus cut elimination (for example, the strongly normalising procedure of Urban [19]) could then be simulated in our standard natural deduction setting. We believe to be a new result: the only comparable work we are aware of is in [19], in which cut elimination is related to a non-standard presentation of natural deduction.

Regarding the relationship between the μ-binding of $\nu\lambda\mu$ and existing control operators, we have argued that $\mathcal{C}$ is not the obvious choice for an operator to represent double negation elimination, and that $\mathcal{F}$ is closer to the reduction behaviour we expect. The relationship between our calculus and *delimited* control operators, such as $\mathcal{F}$ should be investigated in more detail. In particular, it seems that continuation applications can be compared with prompts, since they delimit the effect of μ-reductions.

Our calculus incorporates many of the features present in the "symmetric lambda-mu calculus" which Parigot defines in [15]. However, he chooses to make a departure from the realm of Curry-Howard calculi: his calculus relates to classical logic by implicitly identifying the types $\neg\neg A$ and A.

The $\lambda\Delta$-calculus of Rehof and Sørensen [18] is the only other calculus we are aware of with a Curry-Howard correspondence with a Gentzen-style classical natural deduction. However, the reduction rules of their calculus are fairly restricted; we believe them to be subsumed by Parigot's original definitions for $\lambda\mu$.

The restriction of our calculus to confluent subsystems is an important area to investigate. In particular, we would like to discover the differences between a call-by-name restriction of $\nu\lambda\mu$ and the $\lambda\mu$-calculus, and between a call-by-value version and Ong and Stewart's call-by-value $\lambda\mu$ [11]. Proofs of strong normalisation of typeable terms should also be completed for this work.

We would like to thank Steffen van Bakel, Luca Cardelli, Krysia Broda, Ian Hodkinson, Dorian Gaertner, Jayshan Raghunandan and Nathaniel Charlton for helpful and patient discussions on the subject of this paper.

References

1. Zena M. Ariola and Hugo Herbelin. Minimal classical logic and control operators. In *Proc. ICALP '03*, volume 2719 of *LNCS*, pages 871–885. Springer, 2003.
2. Zena M. Ariola, Hugo Herbelin, and Amr Sabry. A proof theoretic foundation of abortive continuations (extended version). to appear.
3. G. M. Bierman. A computational interpretation of the $\lambda\mu$-calculus. In *Proceedings of Symposium on Mathematical Foundations of Computer Science.*, volume 1450 of *Lecture Notes in Computer Science*, pages 336–345. Springer-Verlag, 1998.
4. Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proc. ICFP'00*, pages 233–243. ACM, 2000.
5. Philippe de Groote. On the relation between the lambda-mu-calculus and the syntactic theory of sequential control. In *LPAR '94: Proc. 5th International Conference on Logic Programming and Automated Reasoning*, pages 31–43, London, UK, 1994. Springer-Verlag.
6. Philippe de Groote. Strong normalization of classical natural deduction with disjunction. In *TLCA*, pages 182–196, 2001.
7. Bruce Duba, Robert Harper, and David MacQueen. Typing first-class continuations in ml. In *Proc. POPL '91*, pages 163–173, New York, NY, USA, 1991. ACM Press.
8. M. Felleisen, D. P. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Journal of Theoretical Computer Science*, 52:205–237, 1987.
9. Gerhard Gentzen. Untersuchungen über das logische schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.
10. T. Griffin. The formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90*, pages 47–57. ACM Press, 1990.
11. C.-H. Luke Ong and Charles A. Stewart. A Curry-Howard foundation for functional computation with control. In *Proc. 24th Symp. on Principles of Programming Languages*, pages 215–227. ACM Press, New York, 1997.
12. M. Parigot. An algorithmic interpretation of classical natural deduction. In *Proc. LPAR'92*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.
13. M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *The Journal of Symbolic Logic*, 62(4):1461–1479, December 1997.
14. Michel Parigot. Classical proofs as programs. In *Proc. Third Kurt Gödel Colloquium on Computational Logic and Proof Theory*, pages 263–276, London, UK, 1993. Springer-Verlag.
15. Michel Parigot. On the computational interpretation of negation. In *Proc. 14th Annual Conference of the EACSL*, pages 472–484, London, UK, 2000. Springer-Verlag.
16. Dag Prawitz. *Natural Deduction, A Proof-Theoretical Study*. Almqvist and Wiskell, Stockholm, 1965.
17. W. Py. *Confluence en $\lambda\mu$-calcul*. PhD thesis, Université de Savoie, 1998.
18. N. Rehof and M. Sørensen. The $\lambda\delta$ calculus. *TACS, volume 789 of LNCS pages 516–542.*, 1994.
19. Christian Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, October 2000.