$\lambda\mu$-CALCULUS: AN ALGORITHMIC INTERPRETATION OF CLASSICAL NATURAL DEDUCTION

Michel Parigot
Equipe de logique — CNRS UA 753
45-55 5ème étage, Université Paris 7
2 place jussieu, 75251 PARIS Cedex 05, FRANCE
e-mail: parigot@logique.jussieu.fr

1 INTRODUCTION

This paper presents a way of extending the paradigm "proofs as programs" to classical proofs. The system we use is derived from the general Free Deduction system presented in [3].

Usually when considering proofs as programs, one has only in mind some kind of intuitionistic proofs. There is an obvious reason for that restriction: only intuitionistic proofs are contructive, in the sense that from the proof of an existential statement, one can get a witness of this existential statement. But from the programming point of view, constructivity is only needed for Σ^0_1-statements, for which classical and intuitionistic provability coincide. This means that, classical proofs are also candidates for being programs. In order to use them as programs, one has two tasks to achieve:

(i) to find a system in which one can extract directly a program from a classical proof (and not by means of a translation to intuitionistic logic), and

(ii) to understand the algorithmic meaning of classical constructions.

The system we will consider is a natural deduction system with multiple conclusions, we will call it Classical Natural Deduction (the one with the absurdity rule being called Usual Natural Deduction). It is a particular subsystem of Free Deduction (FD) with inputs fixed to the left, chosen for its simplicity: it can be seen as a simple extension of intuitionistic natural deduction, whose algorithmic interpretation is very well known. In this context, the contribution of classical constructs to programming appears clearly: they correspond to control operators added to functional languages, like *call/cc* in Scheme. In both contexts, the role of the classical constructs is the same: they allow to take shorter routes in the construction of a *proof/program*.

The link between control operators and classical constructs has first been made by T. Griffin in [1], where he proposes to type the C operator of Felleisen, with the type $\neg\neg A \to A$. The system he obtains is not satisfactory from the logical point of view: the reduction is in fact a reduction strategy and the type assigned to C doesn't fit in general the reduction rule for C. C. Murthy further analysed the connections

between control operators, classical constructs and translations from classical logic to intuitionistic logic (see [4]).

The difficulties met in trying to use $\neg\neg A \to A$ (or the classical absurdity rule) as a type for control operators is not really due to classical logic, but much nore to the deduction system in which it is expressed. It is not easy to find a satisfactory notion of reduction in usual natural deduction because of the restriction to one conclusion which forbids the most natural transformations of proofs (they often generate proofs with more than one conclusion). Of course, as a by-product of our work, we can get possible adequate reductions for usual natural deduction, but none of them can be called "the " canonical one.

Classical natural deduction has precisely been chosen in order to avoid these problems (it is the simplest subsystem of free deduction which is closed under reduction) and the algorithmic calculus extracted from it. As a consequence, it enjoys the theoretical properties of intuitionistic systems such as confluence and strong normalisation. The extracted pure calculus, called $\lambda\mu$-calculus, is an extension of λ-calculus which satisfies confluence too, and preserves the type during reduction. As expected it allows to reproduce control operators, but in a way which is not exactly the usual one. *Call/cc* is simulated in a manner which allows to reduce at any place in the terms (independently of a strategy).

One important question, which is only sketched at the end of this paper, is the one of results. This question appears if one consider the logical system not only as a type system but directly as a computational system in which results are also proofs (of a data type). In the intuitionistic case there is a uniqueness property of results. In classical logic the uniqueness property fails (uniqueness contradicts, in some sense, confluence). But we can show that there is an operator (a typed $\lambda\mu$-term) which allows to compute the "intuitionistic" result among the "classical" ones, without additional computational cost. This means that, as far as computation is concerned, classical logic works as well as intuitionistic logic and that intuitionistic logic is only useful to communicate results.

Acknowledgements.
I thank J.L. Krivine and C. Raffalli for their stimulating and penetrating comments on this work.

2 CLASSICAL NATURAL DEDUCTION

2.1 Rules of classical natural deduction

Formulas are constructed with the logical operators $\neg$ (negation), $\rightarrow$ (implication), $\forall$ (first and second order universal quantifier). The greek letters $\Gamma, \Delta, \Pi, \Sigma$ denote sets of formulas; latin letters denote formulas. Sequents $\Gamma \vdash \Delta$ are interpreted as usual in sequent calculus. The deduction rules are the following (where y (resp. Y) is not free in the conclusion of the first order (resp. second order) introduction rule for $\forall$).

$$A \vdash A$$

$$\frac{\Pi, A \vdash B, \Sigma}{\Pi \vdash A \rightarrow B, \Sigma}\ {\rightarrow}i \qquad \frac{\Gamma \vdash A \rightarrow B, \Delta \quad \Gamma' \vdash A, \Delta'}{\Gamma, \Gamma' \vdash B, \Delta, \Delta'}\ {\rightarrow}e$$

$$\frac{\Pi, A \vdash \Sigma}{\Pi \vdash \neg A, \Sigma}\ \neg i \qquad \frac{\Gamma \vdash \neg A, \Delta \quad \Gamma' \vdash A, \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \neg e$$

$$\frac{\Pi \vdash A[y/x], \Sigma}{\Pi \vdash \forall x A, \Sigma}\ \forall i \qquad \frac{\Gamma \vdash \forall x A, \Delta}{\Gamma \vdash A[t/x], \Delta}\ \forall e$$

$$\frac{\Pi \vdash A[Y/X], \Sigma}{\Pi \vdash \forall X A, \Sigma}\ \forall i \qquad \frac{\Gamma \vdash \forall X A, \Delta}{\Gamma \vdash A[T/X], \Delta}\ \forall e$$

The formulas explicitely mentionned in the rules are called *active*. The one which bears the connective is called the *main formula* of the rule. Weakening is managed implicitely: non-occuring active formulas are allowed in the premises of the rules.

Comments.
(i) For simplicity, we deal only with the connectives whose (intuitionistic) algorithmic interpretation is done in lambda-calculus, but the system and its algorithmic interpretation extend to the other connectives.
(ii) The deduction system is presented with sequents, but can of course be also presented as a deduction system of (multiple) formulas; here are for instance the corresponding rules for $\rightarrow$:

$$\frac{\begin{array}{c}[A]^1 \\ \cdots \\ B, \Sigma\end{array}}{A \rightarrow B, \Delta}\ ^1 \qquad \frac{A \rightarrow B, \Delta \quad A, \Delta'}{B, \Delta, \Delta'}$$

(iii) Proofs of usual natural deduction are easely translated in this system: one replaces each axiom $\neg A \vdash \neg A$ which is used in an absurdity rule by the following piece of proof

$$\frac{A \vdash A}{\vdash \neg A, A}$$

2.2 On the choice of the rules

Classical natural deduction is a subsystem of free deduction with inputs fixed to the left. Here are, for instance, the corresponding rules of free deduction for $\rightarrow$ and $\neg$ where the missing premisses (which are here restricted to axioms) are inside boxes.

$$\frac{\boxed{A \rightarrow B \vdash A \rightarrow B} \quad \Pi, A \vdash B, \Sigma}{\Pi \vdash A \rightarrow B, \Sigma} \qquad \frac{\Gamma \vdash A \rightarrow B, \Delta \quad \Gamma' \vdash A, \Delta' \quad \boxed{B \vdash B}}{\Gamma, \Gamma' \vdash B, \Delta, \Delta'}$$

$$\frac{\boxed{\neg A \vdash \neg A} \quad \Pi, A \vdash \Sigma}{\Pi \vdash \neg A, \Sigma} \qquad \frac{\Gamma \vdash \neg A, \Delta \quad \Gamma' \vdash A, \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

In addition to this choice of inputs, which corresponds to the usual functional view of proofs, some simplifications have been done to make the system as close as possible to intuitionistic natural deduction and therefore, the classical constructs easier to understand:

(i) The rules for $\rightarrow$ are the usual ones, i.e. those of the unified version where the introduction rule has two active formulas A and B in the premise; we could instead take the decomposed version with two introduction rules, which would generate a decomposed version of β-reduction (the unified version is in a certain sense an optimisation of the decomposed one).

(ii) The choice of the inputs to the left allows to kill every premise having a left active formula. In the present system the ones which are killed are precisely the ones which are killed in intuitionistic natural deduction. Other choices could lead to interesting new algorithmic properties; a good candidate is the unrestricted elimination rule for $\rightarrow$:

$$\frac{\Gamma \vdash A \rightarrow B, \Delta \quad \Gamma' \vdash A, \Delta' \quad \Gamma'', B \vdash \Delta''}{\Gamma, \Gamma', \Gamma'' \vdash \Delta, \Delta', \Delta''}$$

The resulting system is a natural deduction system with multiple conclusions. Several such systems have been proposed in the past for proof-theoretic investigations[1] but, as far as I know, they have never been studied from the algorithmic point of view.

2.3 Cuts and their elimination

The algorithmic interpretation of the system will follow the cut-elimination procedure. Cuts are understood as "obstacles" to the subformula property. As in free deduction, one distinguishes between logical cuts and structural cuts. One has a *logical cut* when the main formula of an elimination rule R_1 is active in the preceding rule R_2 and R_2 is an introduction rule; one has a *structural cut* when the main formula of an elimination rule R_1 is not active in the preceding rule R_2.

Logical cuts are nothing else but the usual cuts of intuitionistic natural deduction. The specificity of classical natural deduction appears with strutural cuts: because

[1] see C. Celluci, *Existential Instantiation and Normalisation in Sequent Natural Deduction, Annals of Pure and Applied Logic* (to appear), for a discussion of these systems and references.

conclusions contain more than one formula, the active formula of the conclusion of a rule is not necessary active as formula of the premise of the next rule.

Logical and structural cuts have corresponding reduction rules. We give here the reduction rules for the connective $\rightarrow$, which are the important ones from the computational point of view (the rules are analogous for the other logical operators). In these rules, d_i are names for proofs indicated by a line of dots.

Logical reduction:

$$\dfrac{\dfrac{\dfrac{A \vdash A}{\vdots\ d_1}}{\dfrac{\Gamma_1, A \vdash B, \Delta_1}{\Gamma_1 \vdash A \rightarrow B, \Delta_1}} \qquad \dfrac{\vdots\ d_2}{\Gamma_2 \vdash A, \Delta_2}}{\Gamma_1, \Gamma_2 \vdash B, \Delta_1, \Delta_2}$$

reduces to

$$\dfrac{\dfrac{\vdots\ d_2}{\Gamma_2 \vdash A, \Delta_2}}{\dfrac{\vdots\ d_1}{\Gamma_1, \Gamma_2 \vdash B, \Delta_1, \Delta_2}}$$

The resulting proof d is obtained by replacing in d_1 the occurences of the axiom $A \vdash A$, by the proof d_2 with conclusion $\Gamma_2 \vdash A, \Delta_2$; the conclusion of d is $\Gamma_1, \Gamma_2 \vdash B, \Delta_1, \Delta_2$ up to weakenings. If there are several occurences of $A \vdash A$ in d_1, the subproof d_2 is duplicated; if there is no occurence of $A \vdash A$ in d_1, the subproof d_2 is erased.

Structural reduction:

$$\dfrac{\dfrac{\vdots\ d_1}{\Gamma_1 \vdash A \rightarrow B, \Delta_1} \qquad \dfrac{\vdots\ d_2}{\Gamma_2 \vdash A \rightarrow B, \Delta_2} \qquad \dfrac{\vdots\ d_3}{\Gamma_3 \vdash A, \Delta_3}\ R}{\Gamma_2, \Gamma_3 \vdash B, \Delta_2, \Delta_3}$$

reduces to

$$\dfrac{\dfrac{\dfrac{\vdots\ d_1}{\Gamma_1 \vdash A \rightarrow B, \Delta_1} \qquad \dfrac{\vdots\ d_3}{\Gamma_3 \vdash A, \Delta_3}\ R}{\Gamma_1, \Gamma_3 \vdash B, \Delta_1, \Delta_3}}{\vdots\ d_2}}{\Gamma_2, \Gamma_3 \vdash B, \Delta_2, \Delta_3}$$

The resulting proof d is obtained by replacing inductively in d_2 each subproof d_1 whose conclusion contains an active occurence of $A \rightarrow B$, by the proof obtained by applying R directly to the conclusions of d_1 and d_3; the conclusion of d is $\Gamma_2, \Gamma_3 \vdash B, \Delta_2, \Delta_3$ up to weakenings. If there are several such subproofs d_1, the subproof d_3 is duplicated; if there is no such subproof d_1, the subproof d_3 is erased.

Comments. The logical reduction given here coincides exactly with the one of free deduction. The structural reduction takes into account the particular form of the deductions in classical natural deduction.

2.4 Algorithmic interpretation

The algorithmic interpretation of classical natural deduction is based on the cut-elimination procedure, as in the case of intuitionistic natural deduction (where it is expressed as a typed lambda-calculus). For that purpose, formulas are replaced by indexed formulas. In this way the previous reduction rules become precisely defined and, because one deals with sets of indexed formulas, the contraction rule is managed implicitely (formulas with the same index are automatically contracted). In the intuitionistic case the indexes are only needed for the left formulas of the sequents (the hypotheses), because the right formula (the conclusion) is unique. In the present case indexes are required on both side: left formulas receive indexes $x, y, z, \ldots$ (called λ-variables), right formulas receive indexes $\alpha, \beta, \gamma, \ldots$ (called μ-variables), and the binding mechanisms of these variables allow to recognize in a sequent which one is concerned by the current rule.

The algorithmic calculus is a little bit simplified (but not changed) if one keeps a non-indexed *current formula* to the right of the sequent, which denotes the active formula of a premise. This current formula is managed as in usual natural deduction, i.e. the active formula of a conclusion is the current formula unless otherwise stated: the binding mechanism of μ-variables becomes thus the transformation of an indexed formula into the current formula. In this way intuitionistic proofs are treated exactly as usual in the more general context of classical proofs. Of course, because indexing is not systematic, extra rules for indexing are required (we call them naming rules). These rules replace classical absurdity rule of usual natural deduction.

The algorithmic interpretation of classical natural deduction generates a pure calculus, called *$\lambda\mu$-calculus*, which extends λ-calculus in a simple manner.

2.5 Usual natural deduction

Some difficulties for finding a satisfactory reduction notion for usual natural deduction (with the classical absurdity rule) appear clearly when it is embedded in Free Deduction. The introduction and elimination rules for $\neg$ are restrictions to left inputs of the left and right rules of Free Deduction:

$$\frac{\boxed{\neg A \vdash \neg A} \quad \Pi, A \vdash}{\Pi \vdash \neg A} \qquad\qquad \frac{\Gamma \vdash \neg A \quad \Gamma' \vdash A}{\Gamma, \Gamma' \vdash}$$

Because of the restriction to at most one formula in the conclusion, these rules become incomplete for classical logic; completeness is achieved with the classical absurdity rule, which is another restriction of the left rule of Free Deduction:

$$\frac{\Gamma, \neg A \vdash \quad \boxed{A \vdash A}}{\Gamma \vdash A}$$

This rule generates cuts of a new kind: instead of having only left rules (introductions) directly followed by right rules (eliminations), one can have a right rule (elimination) followed by a left rule (absurdity rule), with some other rules between the two.

$$\dfrac{\dfrac{\dfrac{\neg A \vdash \neg A \qquad \overset{\cdots\cdots d_1}{\Gamma \vdash A}}{\Gamma, \neg A \vdash}}{\overset{\cdots\cdots\cdots d_2}{\Delta, \neg A \vdash}}}{\Delta \vdash A}$$

The natural way of eliminating this kind of cut would be to compose directly d_2 and d_1, but if d_2 is not trivial, this creates sequents in d_2 having two formulas in the conclusion (the problem is that usual natural deduction is not closed under structural reduction).

3 $\lambda\mu$-CALCULUS

3.1 Pure $\lambda\mu$-calculus

$\lambda\mu$-calculus has two kinds of variables: the λ-variables $x, y, z, ...$, and the μ-variables $\alpha, \beta, \gamma, ...$ One way of understanding $\lambda\mu$-calculus is to consider it as an extension of λ-calculus where one has the possibility to name arbitrary subterms by μ-variables and to abstract on these names: this means that an operation can be applied directly to subterms of a term, and not only to the whole term as in λ-calculus.

Terms contain **named** and **unnamed** terms defined inductively as follows:
$-x$ is an unnamed term, if x is a λ-variable;
$-\lambda x.u$ is an unnamed term, if x is a λ-variable and u is an unnamed term;
$-(t\ u)$ is an unnamed term, if t and u are unnamed terms;
$-\mu\alpha.e$ is an unnamed term, if e is a named term and α is a μ-variable;
$-[\alpha]t$ is a named term, if t is an unnamed term and α is a μ-variable.
For simplicity, one considers only terms where bounded and free variables are distinct and bounded variables are bounded only one time (this means that one considers terms up to renaming of bounded variables).

The basic **reduction rules** are the following:
logical reduction:

$$(\lambda x.u\ v) \rhd_c u[v/x]$$

structural reduction:

$$(\mu\beta.u\ v) \rhd_c \mu\beta.u[[\beta](w\ v)/[\beta]w]$$

renaming:

$$[\alpha]\mu\beta.u \rhd_c u[\alpha/\beta]$$

where $u[[\beta](w\ v)/[\beta]w]$ is obtained from u by replacing inductively each subterm of the form $[\beta]w$ by $[\beta](w\ v)$.

The one-step reduction relation (denoted $\triangleright_1$) is defined as the compatible closure of this basic reduction relation, and the reduction relation (denoted $\triangleright$) as the reflexive and transitive closure of the one-step reduction relation.

Example. In $\lambda\mu$-calculus (contrary to λ-calculus), there are terms which give always the same result, independently of the number of arguments they are applied to. For instance, the term $\tau = \lambda x.\lambda y.\mu\delta.[\varphi](x\ y)$ is such that, for each $n \in \mathbf{N}$, $(((\tau\ x)\ y)\ z_1...z_n) \triangleright \mu\delta.[\varphi](x\ y)$.

Comment. The operator μ looks like a λ having a potentially infinite number of arguments. The effect of the reduction of $(...(\mu\beta.u\ v_1)...v_n)$ is to add the arguments $v_1, ..., v_n$ to the subterms of u named β, and this independently of the number n of arguments $\mu\beta.u$ is applied to. If the number n of arguments would be known in advance, one could replace $\mu\beta$ by $\lambda x_1...\lambda x_n$ and $[\beta]w$ by $(...(w\ x_1)...x_n)$, with the usual reduction for λ.

Remark. There is a reduction for the operator μ which is similar to η-reduction in λ-calculus:

$$\mu\alpha.[\alpha]u \triangleright_c u \quad \text{if } \alpha \text{ has no free occurence in } u.$$

3.2 Confluence of $\lambda\mu$-calculus

Theorem 1 *In $\lambda\mu$-calculus, reduction is confluent i.e., if $u \triangleright u_1$ and $u \triangleright u_2$, then there exists v such that $u_1 \triangleright v$ and $u_2 \triangleright v$.*

Proof. One proceeds as for λ-calculus. One defines a new reduction relation $\Rightarrow_1$ such that:

(i) $\triangleright$ is the transitive closure of $\Rightarrow_1$ and

(ii) $\Rightarrow_1$ is confluent.

From this two facts one deduces as usual that $\triangleright$ is confluent.

The relation $\Rightarrow_1$ is defined inductively as follows:

(a) $x \Rightarrow_1 x$;

(b) if $u \Rightarrow_1 u'$, then $\lambda x.u \Rightarrow_1 \lambda x.u'$;

(c) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$, then $(u\ v) \Rightarrow_1 (u'\ v')$;

(d) if $u \Rightarrow_1 u'$, then $\mu\alpha.u \Rightarrow_1 \mu\alpha.u'$;

(e) if $u \Rightarrow_1 u'$, then $[\alpha]u \Rightarrow_1 [\alpha]u'$;

(f) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$, then $(\lambda x.u\ v) \Rightarrow_1 u'[v'/x]$;

(g) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$, then $(\mu\alpha.u\ v) \Rightarrow_1 \mu\alpha.u'[[\alpha](w\ v')/[\alpha]w]$;

(h) if $u \Rightarrow_1 u'$, then $[\alpha]\mu\beta.u \Rightarrow_1 u'[\alpha/\beta]$.

It is easy to check that $\triangleright$ is the transitive closure of $\Rightarrow_1$ and one deduces the confluence of $\Rightarrow_1$ from the following facts:

(iii) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$, then $u[v/x] \Rightarrow_1 u'[v'/x]$;

(iv) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$, then $u[[\alpha](w\ v)/[\alpha]w] \Rightarrow_1 u'[[\alpha](w\ v')/[\alpha]w]$;

(v) if $u \Rightarrow_1 u'$, then $u[\alpha/\beta] \Rightarrow_1 u'[\alpha/\beta]$.

Remark. $\lambda\mu$-calculus suggests some more reduction rules which are not required for the cut elimination procedure, e.g. the following one, which is symmetrical to the structural reduction rule:

$$(u\ \mu\beta.v) \triangleright_c \mu\beta.v[[\beta](u\ w)/[\beta]w]$$

The addition of such reduction rules allows to have a stronger notion of normal form, but it destroys of course the confluence of the calculus.

3.3 Typed $\lambda\mu$-calculus

Here one deals with sequents such that (i) formulas to the left of $\vdash$ are indexed with λ-variables, (ii) formulas to the right of $\vdash$ are indexed with μ-variables, except at most one formula which is not indexed, (iii) distinct formulas never have the same index. To each sequent in a proof is associated a term of $\lambda\mu$-calculus.

In the following rules, Γ, Π are set of formulas indexed by λ-variables, and Δ, Σ are set of formulas indexed by μ-variables.

Logical rules

$$x : A^x \vdash A$$

$$\frac{u : \Pi, A^x \vdash B, \Sigma}{\lambda x.u : \Pi \vdash A \to B, \Sigma} \qquad \frac{t : \Gamma \vdash A \to B, \Delta \quad u : \Gamma' \vdash A, \Delta'}{(t\ u) : \Gamma, \Gamma' \vdash B, \Delta, \Delta'}$$

$$\frac{u : \Pi \vdash A[y/x], \Sigma}{u : \Pi \vdash \forall x A, \Sigma} \qquad \frac{u : \Gamma \vdash \forall x A, \Delta}{u : \Gamma \vdash A[t/x], \Delta}$$

$$\frac{u : \Pi \vdash A[Y/X], \Sigma}{u : \Pi \vdash \forall X A, \Sigma} \qquad \frac{u : \Gamma \vdash \forall X A, \Delta}{u : \Gamma \vdash A[T/X], \Delta}$$

Naming rules

$$\frac{t : \Pi \vdash A, \Sigma}{[\alpha]t : \Pi \vdash A^\alpha, \Sigma} \qquad \frac{e : \Gamma \vdash A^\alpha, \Delta}{\mu\alpha.e : \Gamma \vdash A, \Delta}$$

Comment. The logical rules are essentially the ones of typed λ-calculus. The structural rules are reproduced implicitely at the level of indexed formulas; the right structural rules are thus connected to the naming rules: the first one includes contractions (the case where A^α already appears in Δ) and the second one, the weakening (the case where A^α does not occur).

As in typed λ-calculus one can define $\neg A$ as $A \to \bot$ and use the previous rules with the following special interpretation of naming for $\bot$: for α a μ-variable, $\bot^\alpha$ is not mentionned; this gives the following rules (where δ has no free occurence in e):

$$\frac{u : \Pi, A^x \vdash \bot, \Sigma}{\lambda x.u : \Pi \vdash \neg A, \Sigma} \qquad \frac{t : \Gamma \vdash \neg A, \Delta \quad u : \Gamma' \vdash A, \Delta'}{(t\ u) : \Gamma, \Gamma' \vdash \bot, \Delta, \Delta'}$$

$$\frac{t : \Pi \vdash \bot, \Sigma}{[\gamma]t : \Pi \vdash \Sigma} \qquad \frac{e : \Gamma \vdash \Delta}{\mu\delta.e : \Gamma \vdash \bot, \Delta}$$

Of course, one can also write direct rules for $\neg$ (they are derivable from the previous ones):

$$\frac{u \ : \ \Pi, A^x \vdash \Sigma}{\lambda x.\mu\delta.u \ : \ \Pi \vdash \neg A, \Sigma} \qquad \frac{t \ : \ \Gamma \vdash \neg A, \Delta \quad u \ : \ \Gamma' \vdash A, \Delta'}{[\gamma](t\,u) \ : \ \Gamma, \Gamma' \vdash \Delta, \Delta'}$$

Remark. The type system for $\lambda\mu$-calculus can be seen as a system with at most one conclusion, each named formula in the conclusions being replaced by its negation in the hypotheses. The operator μ becomes thus in a certain sense the algorithmic content of the classical absurdity rule. In this interpretation, terms corresponds to the case where there is exactly one conclusion.

Theorem 2 *The type is preserved during reduction, i.e. if* $t \ : \ \Gamma \vdash A, \Delta$ *is derivable and* $t \triangleright t'$, *then* $t \ : \ \Gamma \vdash A, \Delta$ *is derivable.*

Remark. One can imagine more general notions of structural reduction, for instance the following one, where one shifts an arbitrary piece of term instead of an application only (in this rule, x is supposed to have exactly one occurence in t, and t unnamed):

$$t[(\mu\alpha.u\ v)/x] \triangleright \mu\alpha.u[[\alpha]t[(w\ v)/x]/[\alpha]w].$$

But such transformations preserve the type (i.e. are logically correct) only under certain conditions proofs.

3.4 Examples

In the examples the naming rules are not explicitely mentionned (but easy to recover).

Example. $\neg A \to (A \to \forall XX)$

$$\dfrac{\dfrac{\dfrac{\neg A \vdash \neg A \quad A \vdash A}{\neg A, A \vdash}}{\neg A \vdash A \to \forall XX}}{\vdash \neg A \to (A \to \forall XX)}$$

This proof produces the term $\tau = \lambda x.\lambda y.\mu\delta.[\varphi](x\ y)$ of the § 3.1 .

$$\dfrac{\dfrac{\dfrac{x \ : \ \neg A^x \vdash \neg A \quad y \ : \ A^y \vdash A}{[\varphi](x\ y) \ : \ \neg A^x, A^y \vdash}}{\lambda y.\mu\delta.[\varphi](x\ y) \ : \ \neg A \vdash A \to \forall XX}}{\lambda x.\lambda y.\mu\delta.[\varphi](x\ y) \ : \vdash \neg A \to (A \to \forall XX)}$$

One can remark that τ is also of type $\neg A \to \neg A$, but $(...(((\tau\ x)\ y)\ z_1)...\ z_n)$ is typable with τ of type $\neg A \to (A \to \forall XX)$ but not with τ of type $\neg A \to \neg A$ (the typable term is in this context $(...(\mu\delta.[\varphi]((\tau\ x)\ y)\ z_1)...z_n)$. In this context, $\bot$ and $\forall XX$ do not have exactly the same behaviour.

Example. $\neg\neg A \to A$

$$\dfrac{\dfrac{\neg\neg A \vdash \neg\neg A \quad \dfrac{A \vdash A}{\vdash \neg A, A}}{\neg\neg A \vdash A}}{\vdash \neg\neg A \to A}$$

This proof produces a $\lambda\mu$-term as follows:

$$\dfrac{\dfrac{y\ :\ \neg\neg A^y \vdash \neg\neg A \qquad \dfrac{x\ :\ A^x \vdash A}{\lambda x.\mu\delta.[\alpha]x\ :\vdash \neg A, A^\alpha}}{[\varphi](y\ \lambda x.\mu\delta.[\alpha]x)\ :\ \neg\neg A^y \vdash A^\alpha}}{\lambda y.\mu\alpha.[\varphi](y\ \lambda x.\mu\delta.[\alpha]x)\ :\vdash \neg\neg A \to A}$$

Let $\aleph$ be $\lambda y.\mu\alpha.[\beta](y\ \lambda x.\mu\delta.[\alpha]x)$. When applied to arguments $u, v_1, ..., v_n$, it reduces in the following way:

$(...((\lambda y.\mu\alpha.[\beta](y\ \lambda x.\mu\delta.[\alpha]x)\ u)\ v_1)...v_n)$

$\triangleright(...(\mu\alpha.[\beta](u\ \lambda x.\mu\delta.[\alpha]x)\ v_1)...v_n)$

$\triangleright\mu\alpha.[\beta](u\ \lambda x.\mu\delta.[\alpha](...(x\ v_1)...v_n))$

The term $\aleph$ has a behaviour close to the one of the C operator of Felleisen.

Example. $((A \to B) \to A) \to A$

$$\dfrac{\dfrac{(A \to B) \to A \vdash (A \to B) \to A \qquad \dfrac{A \vdash A}{\vdash A \to B, A}}{(A \to B) \to A \vdash A}}{\vdash ((A \to B) \to A) \to A}$$

This proof produces a $\lambda\mu$-term as follows:

$$\dfrac{\dfrac{y\ :\ (A \to B) \to A^y \vdash (A \to B) \to A \qquad \dfrac{x\ :\ A^x \vdash A}{\lambda x.\mu\delta.[\alpha]x\ :\vdash A \to B, A^\alpha}}{(y\ \lambda x.\mu\delta.[\alpha]x)\ :\ (A \to B) \to A^y \vdash A, A^\alpha}}{\lambda y.\mu\alpha.[\alpha](y\ \lambda x.\mu\delta.[\alpha]x)\ :\vdash ((A \to B) \to A) \to A}$$

Let κ be $\lambda y.\mu\alpha.[\alpha](y\ \lambda x.\mu\delta.[\alpha]x)$. When applied to arguments $u, v_1, ..., v_n$, it reduces in the following way:

$(...((\lambda y.\mu\alpha.[\alpha](y\ \lambda x.\mu\delta.[\alpha]x\ )\ u)\ v_1)...v_n)$

$\triangleright(...(\mu\alpha.[\alpha](u\ \lambda x.\mu\delta.[\alpha]x)\ v_1)...v_n)$

$\triangleright\mu\alpha.[\alpha](...((u\ \lambda x.\mu\delta.[\alpha](...(x\ v_1)...v_n))\ v_1)...v_n)$

The term κ has a behaviour close to the one of the *call/cc* operator of the Scheme programming language.

3.5 Further properties of $\lambda\mu$-calculus

We briefly sketch two important topics, that we cannot develop here.

3.5.1 Data types

In intuitionistic natural deduction one has a uniqueness property of results. Suppose for instance that natural numbers are defined by the following second order formula

$$Nx\ :=\ \forall X(X0 \to (\forall y(Xy \to Xsy) \to Xx))$$

which means semantically that x is a natural number if and only if x belongs to each set X containing 0 and closed under the successor function s. For each n, there is a

unique intuitionistic cut-free proof of Ns^n0, and the corresponding term of lambda-calculus can be considered as an adequate represention of the natural number n (see [5]). In classical natural deduction, the uniqueness property fails (for each $n > 0$, there is an infinite number of cut-free proofs of Ns^n0) and classical proofs generates "false" natural numbers: $\lambda x.\lambda f.(f\ \mu\delta.[\varphi]x)$ is for instance a false representation of 1.

But one can show that there is an "output" operator ϕ_N which allows to recover the intuitionistic cut-free proof among the classical ones. Instead of reducing a proof d of Nx, one reduces ϕ_N applied to d (of course, using the rules of reduction of classical logic). For each data type, one can find an output operator which works for the proofs of this data type. Output operators are strongly connected with the memorization operators of Krivine [3].

3.5.2 An abstract $\lambda\mu$-machine

$\lambda\mu$-calculus can be executed by an abstract machine in the same way as λ-calculus. In the case of an environnement based machine (for left-most reduction), with an environnement and a stack, one has only to add two instructions to the ones for λ-calculus: to save the current stack in α (when the code is $\mu\alpha$) and to restore the stack α (when the code is $[\alpha]$).

REFERENCES

[1] T. GRIFFIN, A formulae-as-types notion of control, Proc. POPL, 1990.

[2] W.A. HOWARD, The formulae as types notion of construction. Manuscript 1969. In: To H.B. CURRY: Essays on combinatory logic, λ-calculus and formalism; Seldin, Hindley (eds.), Academic Press 1980.

[3] J.L. KRIVINE, Opérateurs de mise en mémoire et traduction de Gödel. Archive for Mathematical Logic, 30(1990).

[4] C. MURTHY, Classical proofs as programs: how, when, and why. Manuscript, 1991.

[5] M. PARIGOT & J.L. KRIVINE, Programming with proofs. Presented at SCT'87. In EIK 26 (1990).

[6] M. PARIGOT, Free deduction: an analysis of "computations" in classical logic. 2nd Russian Conference on Logic Programming, 1991 (to appear in LNAI).

[7] D. PRAWITZ, Natural deduction. Almqvist&Wiksell, 1965.