Math. Struct. in Comp. Science (1998), *vol.* 8, *pp.* 637–669. Printed in the United Kingdom
© 1998 Cambridge University Press

An environment machine for the $\lambda\mu$-calculus

PHILIPPE DE GROOTE

INRIA-Lorraine, 615 rue du Jardin Botanique – B.P. 101,
F 54602 Villers-lès-Nancy Cedex – FRANCE

E-mail: Philippe.de.Groote@loria.fr

Received 23 December 1996; revised 18 December 1997

We introduce a natural deduction-like formalisation of Parigot's $\lambda\mu$-calculus. From this, we derive an environment machine that allows any well-typed $\lambda\mu$-term to be reduced to its weak head normal form. The soundness and completeness of the machine is proved.

1. Introduction

The goal of this paper is to demonstrate how Parigot's $\lambda\mu$-calculus (Parigot 1992) may act as a correct foundation for functional programming enriched with control operators.

The $\lambda\mu$-calculus is an extension of the λ-calculus that provides classical logic with an algorithmic interpretation. The extension is twofold: on the one hand, new syntactic constructs (μ-*abstraction* and *naming*) are given in order to encode classical proofs; on the other hand, the calculus is supplied with new notions of reduction in order to give algorithmic content to the double-negation rule of classical logic.

Since Griffin's pioneering work (Griffin 1990), based on Felleisen's theory (Felleisen *et al.* 1987; Felleisen and Hieb 1992), it is known how the notion of control in functional programming is related to classical logic through a correspondance akin to the Curry–Howard isomorphism. This analogy, however, is not sufficient to accept the $\lambda\mu$-calculus as a foundation for the notion of control. Indeed, most of the control operators existing in functional programming have been introduced for pragmatic reasons related to the way the functional programs are actually interpreted. Therefore, we must also justify the $\lambda\mu$-calculus on such a pragmatic basis before accepting it as an appropriate foundation. This is what we achieve in this paper by developing an environment machine for the *typed* $\lambda\mu$-calculus.

The problem we have to solve may be explained as follows. The existing control operators obey computation rules that are akin to those used in Krivine (1994). This rule, using the notation of the $\lambda\mu$-calculus, may be written as follows:

$$(\mu\alpha.\, M)\, A_0 \ldots A_n \;\rightarrow\; M[\alpha := \lambda f.\, f\, A_0 \ldots A_n]. \qquad (*)$$

Such a computation rule does not correspond to an actual notion of reduction (in the sense of Barendregt (1984)): first, it is not compatible with the term formation rules (the

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

rule may only be applied at the outermost level); second, it does not satisfy, in general, the subject reduction property. The equivalent of (∗) in the $\lambda\mu$-calculus is the following rule:

$$(\mu\alpha.\, M)\, A_0 \ldots A_n \;\rightarrow\; \mu\alpha.\, M[\alpha := \lambda f.\, [\alpha]\,(f\, A_0 \ldots A_n)], \qquad (**)$$

which is compatible and satisfies the subject reduction property for any type. However, the implementation of this last rule on an environment machine is not straightforward. The difficulty is that the outermost μ-abstraction occurring in the left-hand side of (∗) does not disappear in its right-hand side. Consequently, a straightforward implementation of (∗∗) would require contractions of redexes within the bodies of μ-abstractions, which supposes the implementation of some renaming mechanism in order to avoid clashes between variables. In order to circumvent this problem, we prove that Reduction Rule (∗∗) may be correctly implemented using computation rules akin to (∗).

The paper is organised as follows. The next section is an introduction to the $\lambda\mu$-calculus. This introduction is original in various respects. The syntax that we use is less restrictive than the one introduced by Parigot. This, together with new notions of reduction, provides the calculus with a better handling of negation. We also use intuitionistic sequents (*i.e.*, sequents whose consequents consist of only one formula) instead of Parigot's classical ones. This allows the $\lambda\mu$-calculus to be seen as a classical natural deduction system *à la* Prawitz. Finally, we give a uniform treatment of the different notions of reduction of the calculus, by expressing them in terms of the usual substitution together with *linear β-contractions*.

In Section 3, we briefly review the Krivine machine. We first explain how it works when using λ-terms written in the usual syntax. Then we present the version of the machine that operates on terms written in de Bruijn's nameless notation.

In Section 4, we discuss the notion of weak head reduction in the $\lambda\mu$-calculus. Roughly speaking, weak head reduction strategies for the λ-calculus do not contract β-redexes under the λ-abstractions. By analogy, one could think that weak head reduction strategies for the $\lambda\mu$-calculus should not contract redexes under the μ-abstractions. We explain why this view is wrong, and we define the proper notion of weak head normal form for the $\lambda\mu$-calculus.

Section 5 provides the $\lambda\mu$-calculus with de Bruijn indices. This allows a calculus of explicit substitution, based on $\lambda\sigma_{\Uparrow}$ (Curien *et al.* 1996), to be defined. This is carried out in Section 6, which is rather technical.

Finally, in Section 7, we derive an abstract machine (which we call the μK-machine) from the weak head reduction strategy introduced in Section 4. We prove its correctness, and its completeness in the sense that it allows the weak head normal form of any term to be computed. Then, a careful analysis of our completness proof leads us to a simplification of our machine. This gives rise to a realistic implementation of the $\lambda\mu$-calculus, where μ-abstraction and naming correspond, respectively, to saving and to restoring the *current continuation*. In fact, the resulting machine is akin to the ones described in Streicher and Reus (submitted), with the difference that it is proved to correctly implement typed reductions that satisfy the subject reduction property at any type.

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

2. The $\lambda\mu$-calculus

2.1. *Context-free syntax*

The presentation of the $\lambda\mu$-calculus that we give here differs from the original one. We introduce a notion of *cotype* and use intuitionistic sequents instead of classical ones. With this presentation, the $\lambda\mu$-calculus may be seen as a natural deduction system and its rules appear as instances of natural deduction rules due to Prawitz and Gentzen. It also allows us to simulate the different notions of substitution that we need by using the usual substitution together with linear β-reductions. Consequently, it allows us to give a uniform treatment to the different notions of reduction that we use.

We also adopt a slightly more liberal syntax than the original one. This relaxation of the syntax, which is discussed in Subsection 2.6, is related to the handling of negation and was first introduced in de Groote (1994b).

The terms of the $\lambda\mu$-calculus are built from two disjoint alphabets of variables: the set of λ-variables, and the set of μ-variables. The context-free syntax of the language is given by the following grammar:

$$T \quad ::= \quad x \mid (\lambda x.\, T) \mid (T\ T) \mid (\mu\alpha.\, T) \mid [\alpha]\, T,$$

where x ranges over λ-variables and α ranges over μ-variables. A $\lambda\mu$-term of the form $\mu\alpha.\, T$ is called a *μ-abstraction*, and a $\lambda\mu$-term of the form $[\alpha]\, T$ is called a *named term*. The operator μ is a binding operator. Therefore, the free occurrences of a μ-variable α in T become bound in $\mu\alpha.\, T$. We write $\mathrm{FV}(M)$ for the set of free λ and μ-variables of the $\lambda\mu$-term M.

The usual α-conversion relation extends naturally to the case of bound μ-variables. In order to be protected from clashes between free and bound variables, we adopt Barendregt's variable convention (Barendregt 1984) for μ-variables as well as for λ-variables[†].

2.2. *Typing rules*

The type system of the $\lambda\mu$-calculus in Parigot (1992) amounts to second-order classical logic. In this paper, for the sake of simplicity, we restrict our presentation to the propositional case (*i.e.*, simply typed $\lambda\mu$-calculus). This limitation does not affect the generality of our results. Indeed the presence of first and second-order quantifiers does not require new specific notions of reduction but simply allows more terms to be typed. Therefore, since our abstract machine reduces untyped terms (*i.e.*, typable terms but without typing information), it also works in the second-order setting. In fact, taking first and second-order quantification into account would only complicate the strong-normalisation and subject-reduction proofs.

The set of *types*, which is built on an alphabet of atomic propositions, is given by the following grammar:

$$\tau \quad ::= \quad \bot \mid A \mid (\tau \to \tau),$$

[†] In Section 5, we provide a formal treatment of the notion of free and bound occurrences in terms of de Bruijn indices.

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

where $\bot$ is the distinguished atomic proposition that stands for *absurdity* and A ranges over atomic propositions. We also introduce the set of *cotypes* by saying that $\tau^\bot$ is a cotype whenever τ is a type.

The type system of the $\lambda\mu$-calculus is then defined by means of a sequent calculus. The sequents are of the form

$$\Gamma \vdash T : \tau,$$

where τ is a type, T is a $\lambda\mu$-term and Γ is the typing context. Such typing contexts are defined as sets of declarations of the form $(x : \sigma)$ or $(\alpha : \rho^\bot)$, where x is a λ-variable, α is a μ-variable, σ is a type and $\rho^\bot$ is a cotype, and where each λ- or μ-variable is declared at most once. The typing rules are the following:

Logical rules

$$\Gamma, x : \tau \vdash x : \tau \quad (\textsc{id})$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.\, M : \sigma \to \tau} \ (\textsc{abs}) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\, N : \tau} \ (\textsc{app})$$

Naming rules

$$\frac{\Gamma, \alpha : \tau^\bot \vdash M : \tau}{\Gamma, \alpha : \tau^\bot \vdash [\alpha]\, M : \bot} \ (\textsc{name}) \qquad \frac{\Gamma, \alpha : \tau^\bot \vdash M : \bot}{\Gamma \vdash \mu\alpha.\, M : \tau} \ (\textsc{muabs})$$

Notice that Axiom (ID) and Rule (APP) allow for implicit *weakening* and implicit *contraction*, respectively. This includes weakening and contraction of cotypes, which amounts to right weakening and contraction. In fact, it is not difficult to see how the above type system correspond to classical logic. Intuitively, cotypes correspond to negated formulas. With this interpretation in mind, naming corresponds to an instance of Gentzen's elimination of negation (Gentzen 1955), and μ-abstraction to Prawitz's double negation rule (Prawitz 1965). That is, using the cotype notation:

$$\frac{\begin{array}{cc} & \vdots \\ A^\bot & A \end{array}}{\bot} \qquad\qquad \frac{\begin{array}{c} [A^\bot] \\ \vdots \\ \bot \end{array}}{A}$$

Nevertheless, it is important to note that the negation corresponding to the cotypes is not a connective and that cotypes may not appear within subformulas. To summarise, the $\lambda\mu$-calculus deals with two sorts of negation:

— the usual negation, which is a connective – this negation is defined as $\neg\tau \equiv \tau \to \bot$;
— a meta-negation, which corresponds to the cotypes – this negation, which is not a connective, is involutive *de facto*.

This distinction between two notions of negation, which is not mandatory (see Rehof and Sørensen (1994)), gives an interesting structure to the calculus. Indeed, Parigot's $\lambda\mu$-calculus may be seen as a natural deduction system dealing with two kinds of hypotheses:

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

usual hypotheses that are handled intuitionistically (these hypotheses correspond to λ-variables), and negative hypotheses, for which *reductio ad absurdum* is allowed (these hypotheses correspond to μ-variables). Let us illustrate this by giving a proof of $\neg\neg A \to A$:

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

2.3. *Main reductions*

The computational principle of the λ-calculus is the β-reduction relation:

$$(\lambda x.\, M)\, N \to_{\beta} M[x := N], \qquad (\beta)$$

where $M[x := N]$ denotes the usual capture-avoiding substitution. Consequently, β-reduction is also the main reduction rule of the $\lambda\mu$-calculus. Nevertheless, because of the double negation rule, β-reduction is not sufficient. Consider the following proof scheme:

$$\dfrac{\dfrac{[\alpha : (A \to B)^{\perp}] \qquad \dfrac{\begin{array}{c}[x : A] \\ \vdots \\ M : B\end{array}}{\lambda x.\, M : A \to B}\ \textsc{abs}}{\begin{array}{c}[\alpha]\, (\lambda x.\, M) : \perp \\ \vdots \\ (\ldots [\alpha]\, (\lambda x.\, M) \ldots) : \perp\end{array}}}{(\mu\alpha.\, \ldots [\alpha]\, (\lambda x.\, M) \ldots) : A \to B} \qquad \begin{array}{c}\vdots \\ N : A\end{array}}{(\mu\alpha.\, \ldots [\alpha]\, (\lambda x.\, M) \ldots)\, N : B}\ \textsc{app}$$

This proof contains a *hidden redex*, namely $(\lambda x.\, M)\, N$. Indeed, the above proof is not normal in the sense that it does not satisfy the subformula property, which is due to the fact that an introduction rule (ABS) is followed by an elimination rule (APP). However, because of the use of a double negation rule, this elimination rule does not immediately follow the corresponding introduction rule. Consequently, the redex $(\lambda x.\, M)\, N$ does not actually appear in the proof, and this is why we say that it is hidden.

In order to turn hidden redexes into actual ones, some reduction rule other than β is needed. This new reduction rule, called μ by Parigot, allows the above proof scheme to be

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

reduced as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x:A]}{\vdots}}{M:B}}{\lambda x.M:A\to B}\ \text{ABS} \qquad \dfrac{\vdots}{N:A}}{(\lambda x.M)\,N:B}\ \text{APP}}{[\alpha:B^{\perp}]}}{[\alpha]((\lambda x.M)\,N):\perp}$$
$$\vdots$$
$$\dfrac{(\ldots[\alpha]((\lambda x.M)\,N)\ldots):\perp}{(\mu\alpha.\ldots[\alpha]((\lambda x.M)\,N)\ldots):B}$$

Informally, the notion of μ-reduction may be explained as follows. Roughly speaking, in a μ-abstraction $\mu\alpha.M$ of type $A\to B$, only the subterms named by α are necessarily of type $A\to B$. Hence, when such a μ-abstraction is applied to an argument N of type A, this argument must be passed on to the subterms named by α.

In order to formalise the notion of μ-reduction, we introduce some auxiliary definitions. First, we temporarily extend the syntax of the $\lambda\mu$-calculus by generalising the notion of a named term. The grammar that we consider is the following:

$$\begin{array}{lcl} T & ::= & x \mid (\lambda x.T) \mid (T\ T) \mid (\mu\alpha.T) \mid [\alpha]\,T \mid [N]\,T, \\ N & ::= & \lambda^{\circ} f.T, \end{array}$$

where any *name* $\lambda^{\circ} f.T$ is such that T contains one and only one free occurrence of the variable f. For instance, $\lambda^{\circ} f.f\,x$ is a name, while $\lambda^{\circ} f.x$ and $\lambda^{\circ} f.f\,(f\,x)$ are not.

Second, we give typing rules for the new terms:

$$\dfrac{\Gamma, f:\tau \vdash M:\perp}{\Gamma \vdash \lambda^{\circ} f.M:\tau^{\perp}}\ (\text{ABS}^{\circ}) \qquad \dfrac{\Gamma \vdash \lambda^{\circ} f.M:\tau^{\perp} \qquad \Gamma \vdash N:\tau}{\Gamma \vdash [\lambda^{\circ} f.M]\,N:\perp}\ (\text{NAME}^{\circ})$$

where, in Rule ABS$^{\circ}$ the condition that there is a unique occurrence of f in M must be respected.

Finally, we introduce an auxiliary reduction rule that amounts to linear β-reduction:

$$[\lambda^{\circ} f.M]\,N \to_{\beta^{\circ}} M[f:=N]. \qquad (\beta^{\circ})$$

It is important to note that this β°-reduction relation is linear because of the unique occurrence condition for f. Consequently, it is obvious that the terms of the extended calculus are strongly β°-normalisable. Moreover, the β°-reduction relation is confluent and the β°-normal forms of the extended terms correspond to pure $\lambda\mu$-terms. This allows us to keep to the syntax of Subsection 2.1 by working with equivalence classes of extended terms modulo β°-equivalence. From now on, the symbol $\equiv$ that stands for syntactic equivalence will denote strict syntactic identity modulo α and β°-conversion. Therefore, the extended syntax and the notion of β°-reduction must not be seen as a real extension of

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

the $\lambda\mu$-calculus but rather as a meta-linguistic method – akin to the notion of substitution, for instance – that will be useful in formalising different notions of reduction[†].

With all the above apparatus, the notion of μ-reduction becomes straightforward to define:

$$(\mu\alpha.\, M)\, N \to_{\mu} \mu\beta.\, M[\alpha := \lambda^{\circ} f.\, [\beta]\,(f\, N)]. \qquad (\mu)$$

2.4. *Auxiliary reductions*

The main notions of reduction of the $\lambda\mu$-calculus are β and μ: the β-reduction relation implements the computational principle of intuitionistic logic while the μ-reduction relation gives an algorithmic interpretation to the double negation rule.

In addition to these two reduction relations, we will consider three other notions of reduction whose computational contents are less important. These notions of reduction do not allow hidden β-redexes to become apparent. Nevertheless, they are helpful because they allow useless inferences to be eliminated.

A first auxiliary notion of reduction is Parigot's renaming (Parigot 1992). Consider the following proof scheme:

$$\dfrac{\dfrac{\vdots}{\Gamma,\, \beta : \tau^{\perp},\, \alpha : \tau^{\perp} \vdash M : \perp}}{\dfrac{\Gamma,\, \beta : \tau^{\perp} \vdash \mu\alpha.\, M : \tau}{\Gamma,\, \beta : \tau^{\perp} \vdash [\beta]\,(\mu\alpha.\, M) : \perp}}$$

The operation of naming is, in some sense, the inverse of the operation of μ-abstraction. It is therefore useless to apply an operation of naming on a μ-abstraction. The renaming relation allows such detours to be eliminated. Let us write M^{β}_{α} for the $\lambda\mu$-term obtained by replacing, in M, each free occurrence of α by β. The above proof scheme may be reduced to the following:

$$\dfrac{\vdots}{\Gamma,\, \beta : \tau^{\perp} \vdash M^{\beta}_{\alpha} : \perp.}$$

The second notion of reduction that we introduce in this section is what we call *elimination of absurd weakening*. This notion of reduction is related to the possible cotype declarations of the form $(\alpha : \perp^{\perp})$. Such declarations are clearly useless. In a classical sequent calculus such as Gentzen's LK, they would correspond to absurd conclusions obtained by right weakening. Consider the following proof scheme:

$$\dfrac{\dfrac{\vdots}{\Gamma,\, \alpha : \perp^{\perp} \vdash M : \perp}}{\Gamma \vdash \mu\alpha.\, M : \perp} \qquad (*)$$

From a logical point of view, the last inference of this proof scheme has no effect since one infers $\perp$ from $\perp$. Nevertheless, we may not reduce the above proof scheme simply to

[†] The main advantage of our approach is that we are able to define the different reduction relations that we study by using only the usual notion of substitution. Consequently, we get for free calculi of explicit substitutions for the $\lambda\mu$-calculus, reminiscent of Audebaud's (Audebaud 1994).

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

$\Gamma \vdash M : \bot$ because of the possible free occurrences of α in M. However, each time α occurs free in M, it has been introduced by a naming rule of the following form:

$$\frac{\Delta, \alpha : \bot^{\bot} \vdash N : \bot}{\Delta, \alpha : \bot^{\bot} \vdash [\alpha] N : \bot} \qquad (**)$$

Inference rules such as (**) are also useless, and may simply be skipped. Let us write M_{α} for the term obtained by removing, from M, each free occurrence of α (that is, replacing each subterm of the form $[\alpha] N$ by N). Then, Proof (*) may be reduced as follows:

$$\begin{array}{c} \vdots \\ \Gamma \vdash M_{\alpha} : \bot \end{array}$$

Finally, the third auxiliary notion of reduction we shall use is reminiscent of the notion of η-reduction in the λ-calculus. Consider the following derivation:

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ \Gamma, \alpha : \tau^{\bot} \vdash M : \tau\end{array}}{\Gamma, \alpha : \tau^{\bot} \vdash [\alpha] M : \bot}}{\Gamma \vdash \mu\alpha.[\alpha] M : \tau}$$

Now, if the declaration $(\alpha : \tau^{\bot})$ is irrelevant in assigning τ to M (that is, if α does not occur free in M), the above derivation may be reduced to the following one:

$$\begin{array}{c} \vdots \\ \Gamma \vdash M : \tau \end{array}$$

We have given a proof-theoretic motivation to three notions of reduction: *renaming* (ρ, for short), *elimination of absurd weakening* (ε) and an η-like reduction that we will call θ. Now we have to define these notions of reduction at the level of untyped terms, since we do not want the notion of type to play any dynamic part when evaluating a $\lambda\mu$-term.

Roughly speaking, elimination of absurd weakening amounts to stripping off the occurrences of a μ-variable from a $\lambda\mu$-term. Nonetheless, this operation does not make any sense if a given typing constraint is not satisfied: the μ-variable that is stripped off must be of cotype $\bot^{\bot}$. Consequently, the problem in defining the notion of ε-reduction in the untyped setting is to satisfy the typing constraint without stating an explicit typing condition. Now, observe that any well-typed μ-abstraction $\mu\alpha. M$ of type τ is such that the subterm M is of type $\bot$ and the μ-variable α is of cotype $\tau^{\bot}$. Therefore, whenever a term of the form $\mu\alpha.\mu\beta. M$ is well-typed, the subterm $\mu\beta. M$ must be of type $\bot$ and, consequently, the μ-variable β must be of cotype $\bot^{\bot}$. This observation suggests the following reduction rule[†]:

$$\mu\alpha.\mu\beta. M \to_{\varepsilon} \mu\alpha. M[\beta := \lambda^{\circ} f. f]. \qquad (\varepsilon)$$

On the other hand, the renaming relation makes perfectly good sense when expressed

[†] The ε-rule that we introduce is, in fact, reminiscent of Felleisen's $\mathscr{C}_{idem}$ reduction rule (Felleisen and Hieb 1992).

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

in an untyped framework (indeed, it satisfies the subject reduction property). Hence, a first attempt would be to specify the notion of ρ-reduction as follows:

$$[\alpha](\mu\beta.M) \to M[\beta := \lambda^\circ f.[\alpha]f]. \qquad (***)$$

Unfortunately, the reduction relation obtained by putting ε and (***) together does not satisfy the Church–Rosser property, as is shown by the following counterexample:

$$\begin{array}{lll} [\alpha](\mu\beta.\mu\gamma.M) & \to_\rho & \mu\gamma.M[\beta := \lambda^\circ f.[\alpha]f] \\ [\alpha](\mu\beta.\mu\gamma.M) & \to_\varepsilon & [\alpha](\mu\beta.M[\gamma := \lambda^\circ f.f]). \end{array}$$

The problem is that the ρ-contraction provokes the disappearance of the context $\mu\beta.[\ ]$ that enables the ε-contraction. In order to circumvent this difficulty, we state the ρ-reduction rule as follows:

$$\mu\alpha.[\beta](\mu\gamma.M) \to_\rho \mu\alpha.M[\gamma := \lambda^\circ f.[\beta]f]. \qquad (\rho)$$

Finally, there is no difficulty in stating Rule θ:

$$\mu\alpha.[\alpha]M \to_\theta M \quad (\alpha \notin \mathrm{FV}(M)). \qquad (\theta)$$

2.5. *Confluence, subject reduction and strong normalisation*

Let $\twoheadrightarrow$ stand for the reduction relation induced by the five notions of reduction β, μ, ε, ρ and θ. This relation satisfies the usual properties of confluence, subject reduction and strong normalisation.

Proposition 2.1. (Confluence) Let M, N, O be (untyped) $\lambda\mu$-terms such that $M \twoheadrightarrow N$ and $M \twoheadrightarrow O$. Then there exists a $\lambda\mu$-term P such that $N \twoheadrightarrow P$ and $O \twoheadrightarrow P$.

Proof. The property may be established using a generalisation of the Tait–Martin-Löf method, which is due to Aczel (Klop *et al.* 1993). Details are given in Py (1998). □

In order to establish the subject reduction property for a reduction relation containing β, we need the following substitution lemma, whose proof (in the case of the simply typed λ-calculus) may be found in any text book.

Lemma 2.2. Let Γ be a typing context, σ and τ be types, x be a λ-variable, and M and N be $\lambda\mu$-terms such that $\Gamma, x : \sigma \vdash M : \tau$ and $\Gamma \vdash N : \sigma$. Then $\Gamma \vdash M[x := N] : \tau$.

Similarly, in order to handle the notions of μ, ε, ρ, θ-reduction, we need another substitution lemma, whose easy proof is left to the reader.

Lemma 2.3. Let Γ be a typing context, $\sigma^\perp$ be a cotype, τ be a type, α be a μ-variable, M be a $\lambda\mu$-term and $\lambda^\circ f.N$ be a name such that $\Gamma, \alpha : \sigma^\perp \vdash M : \tau$ and $\Gamma \vdash \lambda^\circ f.N : \sigma^\perp$. Then $\Gamma \vdash M[\alpha := \lambda^\circ f.N] : \tau$.

Note that we need only one substitution lemma for the four notions of μ, ε, ρ, and θ-reduction. This is because all four have been defined using the usual substitution together with the notion of *name*.

Proposition 2.4. (Subject reduction) Let Γ be a typing context, τ be a type, and M and N be $\lambda\mu$-terms such that $M \twoheadrightarrow N$. If $\Gamma \vdash M : \tau$, then $\Gamma \vdash N : \tau$.

Proof. The proof is as usual, using Lemmas 2.2 and 2.3 for the the base cases. □

Finally, we state the strong normalisation property.

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Proposition 2.5. (Strong normalisation) Let Γ be a typing context, τ be a type and M be a $\lambda\mu$-term such that $\Gamma \vdash M : \tau$. Then there is no infinite sequence of $\beta\mu\rho\varepsilon\theta$-contractions starting from M.

Proof. The proposition may be established using standard techniques. Details are given in Appendix A. □

2.6. *Relation to Parigot's original definition*

As we said previously, the syntax that we have adopted is slightly more liberal than the one introduced by Parigot (Parigot 1992). It is immediate that any $\lambda\mu$-term typable according to Parigot's (first-order) system is typable according to ours. Moreover, the notions of μ and ρ-reduction that we have defined correspond exactly to the original ones when dealing with Parigot's original syntax. The converse does not hold. We justify this choice in this subsection.

Parigot defines two syntactic categories of terms: the unamed terms (U) and the named terms (N). The grammar is the following:

$$\begin{array}{lll} U & ::= & x \mid (\lambda x.\, U) \mid (U\, U) \mid (\mu\alpha.\, N), \\ N & ::= & [\alpha]\, U. \end{array}$$

The naming rules given by Parigot are also different from ours because they do not introduce or eliminate the absurd proposition $\bot$. In fact, what they introduce and eliminate amounts to a meta-notion of absurdity, which is denoted by an absence of formula. Hence, using our notations, Parigot's rules are the following:

$$\frac{\Gamma, \alpha : \tau^{\bot} \vdash M : \tau}{\Gamma, \alpha : \tau^{\bot} \vdash [\alpha]\, M :} \ (\textsc{name}) \qquad \frac{\Gamma, \alpha : \tau^{\bot} \vdash M :}{\Gamma \vdash \mu\alpha.\, M : \tau} \ (\textsc{muabs})$$

Consequently, the introduction and elimination of propositional absurdity appear as particular cases of μ-abstraction and naming:

$$\frac{\Gamma \vdash M :}{\Gamma \vdash \mu\beta.\, M : \bot} \ (\bot\text{-}\textsc{intro}) \qquad \frac{\Gamma \vdash M : \bot}{\Gamma \vdash [\gamma]\, M :} \ (\bot\text{-}\textsc{elim})$$

where β does not occur free in M and γ is not declared in Γ, and with the special convention that declarations of the form $(\gamma : \bot^{\bot})$ are not mentioned.

With these conventions, the proof of $\neg\neg A \to A$ that we have given at the end of Subection 2.2 becomes the following:

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

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Parigot's syntax offers the advantage that the notion of ε-reduction is not needed. Indeed, terms of the form $\mu\alpha.\mu\beta.M$ being forbidden, they are replaced by terms of the form $\mu\alpha.[\gamma](\mu\beta.M)$, where γ is a free variable of cotype $\bot^{\bot}$. Consequently, elimination of absurd weakening (that is, ε) may be mimicked by renaming (that is, ρ). However, this apparent economy of concepts induces some odd properties. In the above example, the $\lambda\mu$-term $\lambda y.\mu\alpha.[\gamma](y\,(\lambda x.\mu\beta.[\alpha]x))$ stands for a completed proof. However, it contains a free μ-variable. This phenomenon is due to the fact that the μ-variable that is introduced by the absurdity elimination rule is not declared in the typing context. Consequently, it is condemned to remain free. This is not quite satisfactory because, from a logical point of view, this free μ-variable corresponds to a useless hypothesis that has not been discarded. Moreover, the presence of undeclared free variables becomes a real impediment when representing variables by means of de Bruijn indices.

3. The Krivine abstract machine

3.1. *The K-machine with named variables*

In this section, we review the Krivine abstract machine, which is an environment machine that evaluates λ-terms. We give a first version of it that reduces λ-terms written in the usual concrete syntax. In the next subsection, we give a lower level version of the machine that works on λ-terms in de Bruijn's notation.

The Krivine abstract machine (K-machine, for short) computes weak head normal forms (if any) by contracting head β-redexes. In fact, it can be seen as a call-by-name variant of Landin's SECD machine (Landin 1964).

Roughly speaking, the state (or the dump) of the K-machine is given by a triple $\langle\langle M, \mathscr{E}\rangle, \mathscr{S}\rangle$, where M is the λ-term to be reduced, $\mathscr{E}$ is an environment that assigns values to the free variables of M, and $\mathscr{S}$ is a stack that contains the arguments of M. More formally, let Λ denote the set of the λ-terms and $\mathscr{X}$ the set of λ-variables. The set **Dump** of the states of the K-machine is defined as follows:

$$\begin{aligned} \textbf{Dump} &= \textbf{Closure} \times \textbf{Stack} \\ \textbf{Closure} &= \Lambda \times \textbf{Env} \\ \textbf{Env} &= \mathscr{X} \xrightarrow{\text{fin}} \textbf{Closure} \\ \textbf{Stack} &= \textbf{Closure}^{*}. \end{aligned}$$

Then, the following transition system defines the possible moves of the machine:

(i) $\langle\langle x, \mathscr{E}\rangle, \mathscr{S}\rangle \rightarrow \langle\mathscr{E}(x), \mathscr{S}\rangle$

(ii) $\langle\langle\lambda x.M, \mathscr{E}\rangle, cl :: \mathscr{S}\rangle \rightarrow \langle\langle M, \mathscr{E}[cl/x]\rangle, \mathscr{S}\rangle$

(iii) $\langle\langle M\,N, \mathscr{E}\rangle, \mathscr{S}\rangle \rightarrow \langle\langle M, \mathscr{E}\rangle, \langle N, \mathscr{E}\rangle :: \mathscr{S}\rangle$

where the infix operator '::' denotes the *cons* operation, as in SML.

Intuitively, a closure stands for a term together with a list of substitutions for its free variables. Then, to evaluate a (free) variable consists of performing the corresponding substitution (Transition i). To evaluate an abstraction when the stack of arguments is non-empty consists of contracting a β-redex by adding the corresponding substitution to

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

the environment (Transition ii). Finally, to evaluate an application consists of turning the argument of the application into a closure and in pushing it onto the stack (Transition iii).

3.2. *The K-machine with de Bruijn indices*

When using concrete bound variables, one has to consider λ-terms up to α-congruence, which can be intricate when implementing terms on a machine because it may involve variable renaming. A way to circumvent this problem is to use de Bruijn's nameless notation (de Bruijn 1972), which is a formalism particularly well suited to automatic manipulation. The idea is to represent any bound occurrence of a variable by means of an index. This index, which is a positive integer, corresponds to the number of λ's lying, within the syntactic tree of the term, between the bound occurrence it represents and the corresponding binding λ. For instance, the term $\lambda x.\, x\,(\lambda y.\, x\, y)$ is written as follows using de Bruijn indices: $\lambda\mathbf{0}(\lambda\mathbf{10})$.

More formally, the set of *nameless terms* is defined by the following grammar:

$$T \quad ::= \quad \mathbf{n} \mid (\lambda T) \mid (T\ T)$$

where $\mathbf{n}$ is any positive integer. Then the correspondence ($\Rightarrow$) between the usual concrete syntax and de Bruijn's nameless notation is given by the following system:

$$\Gamma, x \vdash x \Rightarrow \mathbf{0} \qquad \frac{\Gamma \vdash x \Rightarrow \mathbf{n}}{\Gamma,\, y \vdash x \Rightarrow \mathbf{n{+}1}}$$

$$\frac{\Gamma,\, x \vdash M \Rightarrow M'}{\Gamma \vdash \lambda x.\, M \Rightarrow \lambda M'} \qquad \frac{\Gamma \vdash M \Rightarrow M' \qquad \Gamma \vdash N \Rightarrow N'}{\Gamma \vdash M\, N \Rightarrow M'\, N'}$$

where Γ stands for a *sequence* of λ-variables.

The above system illustrates that the de Bruijn indices may be interpreted, from a computational point of view, as addresses in a stack. Indeed, when an index $\mathbf{n}$ stands for a free variable, it corresponds to the n^{th} variable (counting from zero) in the sequence on the left of the turnstile (starting from the right of the sequence).

In the previous section, we defined an environment to be a function from $\mathscr{X}$ to **Closure**, undefined almost everywhere. A common way to represent such functions whose definition domains are finite is by means of finite lists. This observation, together with the remark about the computational interpretation of the de Bruijn indices, explains why environments can be represented as stacks. This leads to a machine where there is no technical difference between environments and stacks.

Let Λ denote the set of de Bruijn's nameless terms. The states of the *K*-machine (with de Bruijn indices) is specified by the following equations:

$$\begin{aligned} \mathbf{Dump} &= \Lambda \times \mathbf{Env} \times \mathbf{Stack} \\ \mathbf{Closure} &= \Lambda \times \mathbf{Env} \\ \mathbf{Env} &= \mathbf{Closure}^* \\ \mathbf{Stack} &= \mathbf{Closure}^*. \end{aligned}$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

The moves of the machine are specified by the following transition system:

(i) $\langle \mathbf{0}, \langle M, \mathscr{E}\rangle :: \mathscr{E}', \mathscr{S}\rangle \to \langle M, \mathscr{E}, \mathscr{S}\rangle$,
(ii) $\langle \mathbf{n+1}, cl :: \mathscr{E}, \mathscr{S}\rangle \to \langle \mathbf{n}, \mathscr{E}, \mathscr{S}\rangle$,
(iii) $\langle \lambda M, \mathscr{E}, cl :: \mathscr{S}\rangle \to \langle M, cl :: \mathscr{E}, \mathscr{S}\rangle$,
(iv) $\langle (MN), \mathscr{E}, \mathscr{S}\rangle \to \langle M, \mathscr{E}, \langle N, \mathscr{E}\rangle :: \mathscr{S}\rangle$.

The striking simplicity of this definition comes mainly from the fact that the K-machine is designed to compute the weak head normal form (if any) of some given λ-term. For this reason, the machine does not perform any reduction or substitution under a λ-abstraction and, consequently, there is no need for any updating of the de Bruijn indices.

Now, in order to adapt the K-machine to the $\lambda\mu$-calculus, we have two tasks to perform: the first is to define the notions of weak head reduction and weak head normal form for the $\lambda\mu$-calculus; the second is to adapt de Bruijn's nameless notation to the $\lambda\mu$-terms. These two tasks are the subjects of the next two sections.

4. Weak head reductions in the $\lambda\mu$-calculus

The weak head reduction strategy $\twoheadrightarrow_h$ in the case of the λ-calculus may be specified by the following formal system:

$$(\lambda x.\, M)\, N \to_h M[x := N] \qquad \frac{M \to_h N}{M\, O \to_h N\, O} \qquad M \twoheadrightarrow_h M \qquad \frac{M \to_h N \quad N \twoheadrightarrow_h O}{M \twoheadrightarrow_h O}$$

This system specifies the two following facts that characterise the weak head reduction strategy:

— redexes occurring in the body of an abstraction are not contracted (absence of Rule ξ);
— redexes occurring in the argument of an application are not contracted either (call-by-name strategy).

Now, in order to adapt this reduction strategy to the $\lambda\mu$-calculus, we must certainly complete the above system with the following axioms:

$$\begin{aligned}
(\mu\alpha.\, M)\, N &\to_h \mu\alpha.\, M[\alpha := \lambda f.\, [\alpha]\,(f\, N)] \\
\mu\alpha.\, \mu\beta.\, M &\to_h \mu\alpha.\, M[\beta := \lambda f.\, f] \\
\mu\alpha.\, [\beta]\,(\mu\gamma.\, M) &\to_h \mu\alpha.\, M[\gamma := \lambda f.\, [\beta]\, f] \\
\mu\alpha.\, [\alpha]\, M &\to_h M \qquad\qquad \alpha \notin \mathrm{FV}(M)
\end{aligned}$$

Then, the question is: should we add inference rules as well in order to allow redexes to be contracted inside μ-abstractions and named terms? At first sight, the answer could be no because of an analogy between λ- and μ-abstraction, on the one hand, and between application and naming, on the other hand. Such an answer, however, does not really make sense as will be demonstrated by a simple example.

One of the properties of the weak head reduction strategy (which explains why it is used in the case of actual programming languages) is that it allows the actual normal form to be computed when the term to be reduced is of atomic type. Now, consider the following $\lambda\mu$-term where $\star$ is a constant of atomic type:

$$\mu\alpha.\, [\alpha]\,((\lambda x.\, x)\,(\mu\beta.\, [\alpha]\,\star)).$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

This term may be reduced as follows:

$$\begin{array}{lll} \mu\alpha.[\alpha]((\lambda x.x)(\mu\beta.[\alpha]\star)) & \to_\beta & \mu\alpha.[\alpha](\mu\beta.[\alpha]\star) \\ & \to_\rho & \mu\alpha.[\alpha]\star \\ & \to_\theta & \star. \end{array}$$

In order to keep the property that the weak head reduction strategy completely reduces any term of atomic type, we must accept the above reduction sequence as belonging to the strategy. But this implies that we allow redexes to be contracted inside μ-abstractions and named terms. Consequently, we have to add the two following rules to the system specifying the weak head reduction strategy:

$$\frac{M \to_h N}{\mu\alpha.M \to_h \mu\alpha.N} \qquad \frac{M \to_h N}{[\alpha]M \to_h [\alpha]N}$$

In the case of the λ-calculus, the weak head reduction strategy is completely specified by the following computation rule:

$$(\lambda x.M)A_0A_1\cdots A_n \to M[x:=A_0]A_1\cdots A_n.$$

We end this section by giving similar rules for the $\lambda\mu$-calculus. These rules, for a reason that we will explain later, are stated in the two contexts $\mu\alpha.[\alpha][\ ]$ and $\mu\alpha.[\ ]$. They are the following:

$$\mu\alpha.[\alpha]((\lambda x.M)A_0A_1\cdots A_n) \to_c \mu\alpha.[\alpha](M[x:=A_0]A_1\cdots A_n) \quad (1)$$

$$\mu\alpha.(\lambda x.M)A_0A_1\cdots A_n \to_c \mu\alpha.M[x:=A_0]A_1\cdots A_n \quad (2)$$

$$\mu\alpha.[\alpha]((\mu\beta.M)A_0A_1\cdots A_n) \to_c \mu\alpha.M[\beta:=\lambda^\circ f.[\alpha](fA_0A_1\cdots A_n)] \quad (3)$$

$$\mu\alpha.(\mu\beta.M)A_0A_1\cdots A_n \to_c \mu\alpha.M[\beta:=\lambda^\circ f.fA_0A_1\cdots A_n]. \quad (4)$$

Computation Rules 1 and 2 amount simply to the usual λ-calculus computation rule stated in the two contexts of interest. On the other hand, Rules 3 and 4 are specific to the $\lambda\mu$-calculus: they involve several μ-reduction steps. In addition, Rule 3 performs one ρ-reduction step while Rule 4 performs one ε-reduction step. We also adopt the convention that Rules 3 and 4 include the particular case where the sequence of terms $A_0A_1\cdots A_n$ is empty (in which case they simply amount to ρ and ε, respectively).

Because our computation rules are stated in the two contexts $\mu\alpha.[\alpha][\ ]$ and $\mu\alpha.[\ ]$, the μ-abstractor '$\mu\beta$' that occurs in the left-hand sides of both Rules 3 and 4 disappears in their respective right-hand sides. This will appear as a key property when defining the μK-machine.

Rules 1 to 4 are consistent with the weak head reduction strategy as shown by the following proposition.

Proposition 4.1. Let M and N be two $\lambda\mu$-terms such that $M \to_c N$. Then, $M \twoheadrightarrow_h N$.

Proof. The proof is by a straightforward computation. □

The converse is not true. Consider, for instance, the following term:

$$\mu\alpha.[\alpha]([\alpha]((\lambda x.x)\star)).$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

It cannot be reduced using our computation rules. In fact this term is not typable, and the results of Section 7 will provide a weak converse to Proposition 4.1: *any closed typable $\lambda\mu$-term may be reduced to its weak head normal form using the computation rules.*

5. $\lambda\mu$-calculus and de Bruijn indices

To adapt de Bruijn's nameless notation to the $\lambda\mu$-calculus is almost straightforward. The only degree of freedom consists in deciding whether both λ and μ-variables will be represented by the same set of indices or by two separate sets. The first choice, which we adopt here, results in a machine with one environment. On the other hand, the second choice would result in a machine with two distinct environments: one for the λ-variables, and another one for the μ-variables, though in fact, such a difference is inessential.

We give below a formal system that defines both the context-free syntax and the typing relation at the same time.

Logical rules

$$\Gamma, A \vdash \mathbf{0} : A \qquad \frac{\Gamma \vdash \mathbf{n} : A}{\Gamma, B \vdash \mathbf{n{+}1} : A} \qquad \frac{\Gamma \vdash \mathbf{n} : A}{\Gamma, B^{\perp} \vdash \mathbf{n{+}1} : A}$$

$$\frac{\Gamma, A \vdash M : B}{\Gamma \vdash \lambda M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}$$

Naming rules

$$\Gamma, A^{\perp} \vdash [\mathbf{0}] : A^{\perp} \qquad \frac{\Gamma \vdash [\mathbf{n}] : A^{\perp}}{\Gamma, B \vdash [\mathbf{n{+}1}] : A^{\perp}} \qquad \frac{\Gamma \vdash [\mathbf{n}] : A^{\perp}}{\Gamma, B^{\perp} \vdash [\mathbf{n{+}1}] : A^{\perp}}$$

$$\frac{\Gamma, A^{\perp} \vdash M : \perp}{\Gamma \vdash \mu M : A} \qquad \frac{\Gamma \vdash [\mathbf{n}] : A^{\perp} \qquad \Gamma \vdash M : A}{\Gamma \vdash [\mathbf{n}]\,M : \perp}$$

Note that the above system makes explicit the analogy existing between *application* and *naming*.

The relation between the nameless notation and the usual concrete syntax is given by adapting the system of Section 3.2:

$$\Gamma, x \vdash x \Rightarrow \mathbf{0} \qquad \frac{\Gamma \vdash x \Rightarrow \mathbf{n}}{\Gamma,\, y \vdash x \Rightarrow \mathbf{n{+}1}} \qquad \frac{\Gamma \vdash x \Rightarrow \mathbf{n}}{\Gamma,\, \alpha \vdash x \Rightarrow \mathbf{n{+}1}}$$

$$\frac{\Gamma,\, x \vdash M \Rightarrow M'}{\Gamma \vdash \lambda x.\, M \Rightarrow \lambda M'} \qquad \frac{\Gamma \vdash M \Rightarrow M' \qquad \Gamma \vdash N \Rightarrow N'}{\Gamma \vdash M\,N \Rightarrow M'\,N'}$$

$$\Gamma, \alpha \vdash [\alpha] \Rightarrow [\mathbf{0}] \qquad \frac{\Gamma \vdash [\alpha] \Rightarrow [\mathbf{n}]}{\Gamma, x \vdash [\alpha] \Rightarrow [\mathbf{n{+}1}]} \qquad \frac{\Gamma \vdash [\alpha] \Rightarrow [\mathbf{n}]}{\Gamma, \beta \vdash [\alpha] \Rightarrow [\mathbf{n{+}1}]}$$

$$\frac{\Gamma, \alpha \vdash M \Rightarrow M'}{\Gamma \vdash \mu\alpha.\, M \Rightarrow \mu M'} \qquad \frac{\Gamma \vdash [\alpha] \Rightarrow [\mathbf{n}] \qquad \Gamma \vdash M \Rightarrow M'}{\Gamma \vdash [\alpha]\,M \Rightarrow [\mathbf{n}]\,M'}$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

We end this section by stating a proposition that relates the two formal systems given above to the typing system of Section 2.2.

Proposition 5.1. Let Γ be a typing context. Let Γ' be the list of variables declared in Γ, and let Γ'' be the list of corresponding types and cotypes (Γ' and Γ'' correspond, respectively, to the first and the second projections of the components of Γ). Let M be a $\lambda\mu$-term and M' be a nameless $\lambda\mu$-term such that $\Gamma' \vdash M \Rightarrow M'$. Then, $\Gamma \vdash M : A$ if and only if $\Gamma'' \vdash M' : A$.

Proof. The proof is straightforward. □

6. $\lambda\mu$-calculus and explicit substitutions

As we have already stressed, by defining our notions of reduction in terms of the usual substitution together with linear β-reductions, we get calculi of explicit substitutions for free. This section is devoted to the introduction of such a calculus, based on $\lambda\sigma_{\Uparrow}$ (Curien *et al.* 1996). This calculus, which is merely technical, is only needed in the course of the proof of Proposition 7.3 (which is given in Appendix B). Hence, the reader who is not interested in such technicalities may proceed directly to the next section.

We do not provide a full account of the $\lambda\sigma_{\Uparrow}$-calculus. We refer the interested reader to Curien *et al.* (1996). We just give here the syntax of the $\lambda\mu$-term with explicit substitutions, together with the associated rewriting system.

The set of terms is that of the $\lambda\mu$-calculus extended with the construction $M\{s\}$, where s is a substitution. The precise syntax is the following:

$$\begin{array}{lcl} T & ::= & \mathbf{n} \mid (\lambda T) \mid (T\ T) \mid (\mu T) \mid [\mathbf{n}]\, T \mid [N]\, T \mid T\{s\}, \\ N & ::= & (\lambda^{\circ} T) \\ s & ::= & id \mid \uparrow \mid T \cdot s \mid \Uparrow(s) \mid s \circ s. \end{array}$$

Note that the above syntax is based on the extended syntax of Section 2.3. Indeed, to make sense, a caculus of explicit substitutions for the $\lambda\mu$-calculus must also provide an explicit treatment of β°. It is to be understood that λ° stands for a linear abstractor: $(\lambda^{\circ} T)$ is a well-formed name if and only if one and only one de Bruijn index in T refers to the outermost λ°.

We will use Λ_{μ} to denote the set of pure $\lambda\mu$-terms (with de Bruijn indices), and $\Lambda_{\mu\sigma}$ to denote the set of $\lambda\mu$-terms with explicit substitutions. Hence, we have that $\Lambda_{\mu} \subset \Lambda_{\mu v}$.

Using the formalism of the $\lambda\sigma_{\Uparrow}$-calculus, the notions of β, μ, ε, and ρ-reduction, are specified, respectively, as follows[†].

$$\begin{array}{rcll} (\lambda M)\, N & \rightarrow_{\beta'} & M\{N \cdot id\} & \text{(Beta)} \\ (\mu M)\, N & \rightarrow_{\mu'} & \mu(M\{\lambda^{\circ}[\mathbf{1}]\,(\mathbf{0}\, N\{\uparrow \circ \uparrow\}) \cdot \uparrow\}) & \text{(Mu)} \\ \mu\mu M & \rightarrow_{\varepsilon'} & (\mu M\{\lambda^{\circ}\mathbf{0} \cdot id\}) & \text{(Epsilon)} \\ \mu[\mathbf{n}]\,\mu M & \rightarrow_{\rho'} & (\mu M\{\lambda^{\circ}[\mathbf{n}{+}\mathbf{1}]\,\mathbf{0} \cdot id\}). & \text{(Rho)} \end{array}$$

[†] We do not give any rule corresponding to θ because, on the one hand, it will not be needed in the course of the proof of Proposition 7.3 and, on the other hand, the explicit substitution formalisms are not well suited to η-like rules.

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

The notion of β°-reduction is explicitly handled by the following rule:

$$[\lambda^\circ M]\, N \quad \to_{\beta'^\circ} \quad M\{N \cdot id\}. \qquad \text{(Beta}^\circ\text{)}$$

Finally, the rewriting system $(\to_\sigma)$, which allows the substitutions themselves to be handled is given in Appendix C.

The next three propositions relate the notions of β, μ, ε and ρ-reduction, as introduced in Sections 2.3 and 2.4, to the present calculus of explicit substitution. The three of them may be established by adapting proofs given in Curien *et al.* (1996). We leave it as a tedious but not difficult exercise for the interested reader.

Proposition 6.1. Let r stand for β, μ, ε and ρ, and let r' stand for β', μ', ε' and ρ', respectively. Let M, N be $\lambda\mu$-terms, let $M' \in \Lambda_\mu$, and let Γ be a list of variables such that $\Gamma \vdash M \Rightarrow M'$. If $M \to_r N$, then there exists $N' \in \Lambda_\mu$ such that $M' \twoheadrightarrow_{r'\sigma\beta^\circ} N'$ and $\Gamma \vdash N \Rightarrow N'$.

The rewriting system $(\to_\sigma)$ given in Appendix C is terminating. On the other hand, any $\lambda\mu$-term is strongly β° normalisable. It is not the case, however, that any $M \in \Lambda_{\mu\sigma}$ is strongly $\sigma\beta'^\circ$-normalisable – see Mellies (1995). Nevertheless, it is easy to show that the $\sigma\beta'^\circ$-normal form of any $M \in \Lambda_{\mu\sigma}$ – denoted $\sigma\beta'^\circ(M)$ – exists. Moreover, for any $M \in \Lambda_{\mu\sigma}$, we have $\sigma\beta'^\circ(M) \in \Lambda_\mu$. This property allows us to state a kind of converse to Proposition 6.1.

Proposition 6.2. Let r stand for β, μ, ε and ρ, and let r' stand for β', μ', ε' and ρ', respectively. Let M be a $\lambda\mu$-term, let $M', N' \in \Lambda_{\mu\sigma}$, and let Γ be a list of variables such that $\Gamma \vdash M \Rightarrow \sigma\beta'^\circ(M')$. If $M' \to_{r'} N'$, then there exists a $\lambda\mu$-term N such that $M \twoheadrightarrow_r N$ and $\Gamma \vdash N \Rightarrow \sigma\beta'^\circ(N')$.

It is possible to refine the above proposition by introducing the notion of external contraction. Intuitively, an external reduction is a reduction step that does not take place within a substitution. More precisely, when formally defining a relation R of contraction reduction, one states a congruence rule for each term formation rule. In particular, in the present setting, one must state the following rule:

$$\frac{s_1 \to_R s_2}{T\{s_1\} \to_R T\{s_2\}}$$

By dropping this rule (and consequently, all the rules corresponding to substitution formation rules), one defines the external contraction relation, which is denoted by $\to_R^{\text{ext}}$.

Proposition 6.3. Let r stand for β, μ, ε and ρ, and let r' stand for β', μ', ε' and ρ', respectively. Let M be a $\lambda\mu$-term, let $M', N' \in \Lambda_{\mu\sigma}$, and let Γ be a list of variables such that $\Gamma \vdash M \Rightarrow \sigma\beta'^\circ(M')$. If $M' \to_{r'}^{\text{ext}} N'$, then there exists a $\lambda\mu$-term N such that $M \to_r N$ and $\Gamma \vdash N \Rightarrow \sigma\beta'^\circ(N')$.

We end this section by introducing some notation that will be needed in the rest of the paper.

Let '$\sqcup$' denote the empty sequence and '::' denote the *cons* operation. We define the function

$$\textsc{App} \;:\; \Lambda_{\mu\sigma} \times (\Lambda_{\mu\sigma})^* \to \Lambda_{\mu\sigma}$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

that applies a term to a list of arguments as follows:

(i) $\textsc{App}(M, \sqcup) = M$,
(ii) $\textsc{App}(M, N :: \mathscr{L}) = \textsc{App}(M\,N, \mathscr{L})$.

If s is a substitution, we write $\textsc{App}(M, \mathscr{L}\{s\})$ for the term defined as follows:

(i) $\textsc{App}(M, \sqcup\{s\}) = M$,
(ii) $\textsc{App}(M, (N :: \mathscr{L})\{s\}) = \textsc{App}(M\,(N\{s\}), \mathscr{L}\{s\})$.

The advantage of the latter notation is that it allows us to write the following substitution rule:

$$(\textsc{App}(M, \mathscr{L}))\{s\} \;\twoheadrightarrow_{\sigma}\; \textsc{App}(M\{s\}, \mathscr{L}\{s\}),$$

which generalises Rules (App) of $\lambda\sigma_{\Uparrow}$ (see Appendix C).

7. An environment machine for the $\lambda\mu$-calculus

7.1. *Overview*

The reader who is familiar with applicative control operators (such as *catch and throw*, or *call with current continuation*) will have noticed a strong similarity between such operators and the $\lambda\mu$-calculus. Indeed the intutitive operational interpretation of a μ-redex is as follows: take the list of arguments of the μ-abstraction and pass it on to the subterms named by the corresponding μ-variable. Now, in the K-machine, the list of arguments corresponds to the stack. This suggests the following moves: when encountering a μ-abstraction, save the stack on the environment; when encountering a μ-variable, restore the corresponding stack. These rough ideas lead to a straightforward generalisation of the Krivine abstract machine.

Let Λ_{μ} denote the set of $\lambda\mu$-terms (in concrete syntax), and $\mathscr{A}$ denote the set of μ-variables. Define the following domains and moves:

$$\begin{aligned} \mathbf{Dump} &= \mathbf{Closure} \times \mathbf{Stack} \\ \mathbf{Closure} &= \Lambda_{\mu} \times \mathbf{Env} \\ \mathbf{Env} &= (\mathscr{X} \xrightarrow{\text{fin}} \mathbf{Closure}) \uplus (\mathscr{A} \xrightarrow{\text{fin}} \mathbf{Stack}) \\ \mathbf{Stack} &= \mathbf{Closure}^{*} \end{aligned}$$

(i) $\langle\langle x, \mathscr{E}\rangle, \mathscr{S}\rangle \;\rightarrow\; \langle \mathscr{E}(x), \mathscr{S}\rangle$
(ii) $\langle\langle \lambda x.\, M, \mathscr{E}\rangle, cl :: \mathscr{S}\rangle \;\rightarrow\; \langle\langle M, \mathscr{E}[cl/x]\,\rangle, \mathscr{S}\rangle$
(iii) $\langle\langle M\,N, \mathscr{E}\rangle, \mathscr{S}\rangle \;\rightarrow\; \langle\langle M, \mathscr{E}\rangle, \langle N, \mathscr{E}\rangle :: \mathscr{S}\rangle$
(iv) $\langle\langle \mu\alpha.\, M, \mathscr{E}\rangle, \mathscr{S}\rangle \;\rightarrow\; \langle\langle M, \mathscr{E}[S/\alpha]\,\rangle, \sqcup\rangle$
(v) $\langle\langle [\alpha]\, M, \mathscr{E}\rangle, \sqcup\rangle \;\rightarrow\; \langle\langle M, \mathscr{E}\rangle, \mathscr{E}(\alpha)\rangle$.

The above machine corresponds actually to the abstract machine that we shall derive from the computation rules of Section 4. However, as we explained in the introduction of this paper, the reduction implemented by Moves (iv) and (v) seems unsound at first sight: it satisifies the subject reduction property only at type $\bot$, while the consistency of classical logic implies that programs (*i.e.*, closed terms) of type $\bot$ cannot exist. Consequently, in order to prove the correctness of our abstract machine, some detour will be needed.

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

7.2. *The μK-abstract machine: preliminary definition*

As we explained in Section 3.2, the simplicity of the K-machine comes from the fact that the redexes occurring within the body of an abstraction are not reduced.

Unfortunately, with the $\lambda\mu$-calculus the situation is not as simple. Indeed, when computing the weak-head normal form of a $\lambda\mu$-term, it may be necessary to perform reductions and substitutions under a μ-abstraction. This is a consequence of the following fact: when contracting a μ-redex, the μ-abstraction involved in this redex does not disappear.

Nevertheless, since we are only interested in the evaluation of closed expressions, we may use the computation rules of Section 4. Consider, for instance, Rule 3. If we concentrate only on the redex (and thus forget the context in which it occurs), Rule 3 seems to suggest the following reduction relation:

$$(\mu\beta.\, M)\, A_0\, A_1 \cdots A_n \;\succ\; M[\beta := \lambda f.\, [\alpha]\, (f\, A_0\, A_1 \cdots A_n)]. \qquad (*)$$

This relation $\succ$, however, is not a notion of reduction because it is only correct when the left-hand side occurs in the context $\mu\alpha.\, [\alpha]\, [\ ]$ while the right-hand side occurs in the context $\mu\alpha.\, [\ ]$. Therefore a machine implementing Transition $(*)$ must also record some contextual information. Therefore, we will provide our abstract machine with an additional component, namely, a Boolean state variable that will indicate whether the content of the machine must be interpreted in the context $\mu[\mathbf{0}]\, [\ ]$ or $\mu[\ ]$.

The next step in the design of the μK-machine is to observe that the substituted term in $(*)$, that is,

$$\lambda f.\, [\alpha]\, (f\, A_0\, A_1 \cdots A_n),$$

is completely characterised by the list of arguments $A_0, A_1, \ldots, A_n$, which corresponds to the content of the stack of the machine. So, as we suggested in Section 7.1, when the machine encounters a μ-abstraction, it must save the contents of the stack in the environment. Conversely, when a named term is encountered, the stack saved in the environment should be restored. However, there are two possible uses of the stack that correspond, respectively, to Rules 3 and 4 of Section 4. In the first case, the stack must be interpreted as

$$\lambda^{\circ} f.\, [\alpha]\, (f\, A_0\, A_1 \cdots A_n),$$

while in the second case, it must be interpreted as

$$\lambda^{\circ} f.\, f\, A_0\, A_1 \cdots A_n.$$

This interpretation depends of the value of the Boolean state variable at the time the stack is saved. Consequently, it is also necessary to save this value.

These ideas are formalized in the following definition.

Definition 7.1. (μK-machine – preliminary definition) The set **Dump** of the states of the μK-machine is defined by the following equations:

$$\begin{aligned} \mathbf{Dump} &= \Lambda_{\mu} \times \mathbf{Env} \times \mathbf{Stack} \times 2 \\ \mathbf{Env} &= ((\mathbf{Stack} \times 2) \cup \mathbf{Closure})^{*} \\ \mathbf{Stack} &= \mathbf{Closure}^{*} \\ \mathbf{Closure} &= \Lambda_{\mu} \times \mathbf{Env}. \end{aligned}$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Let $M, N \in \Lambda_\mu$; $\mathscr{E}, \mathscr{E}' \in \mathbf{Env}$; $\mathscr{S} \in \mathbf{Stack}$; $i \in 2$; $ccl \in (\mathbf{Stack} \times 2) \cup \mathbf{Closure}$; $cl \in \mathbf{Closure}$. The moves of the μK-machine are specified by the following transition system:

(i) $\langle \mathbf{0}, \langle M, \mathscr{E} \rangle :: \mathscr{E}', \mathscr{S}, i \rangle \rightarrow \langle M, \mathscr{E}, \mathscr{S}, i \rangle$,
(ii) $\langle \mathbf{n+1}, ccl :: \mathscr{E}, \mathscr{S}, i \rangle \rightarrow \langle \mathbf{n}, \mathscr{E}, \mathscr{S}, i \rangle$,
(iii) $\langle \lambda M, \mathscr{E}, cl :: \mathscr{S}, i \rangle \rightarrow \langle M, cl :: \mathscr{E}, \mathscr{S}, i \rangle$,
(iv) $\langle (MN), \mathscr{E}, \mathscr{S}, i \rangle \rightarrow \langle M, \mathscr{E}, \langle N, \mathscr{E} \rangle :: \mathscr{S}, i \rangle$,
(v) $\langle \mu M, \mathscr{E}, \mathscr{S}, i \rangle \rightarrow \langle M, \langle \mathscr{S}, i \rangle :: \mathscr{E}, \sqcup, 1 \rangle$,
(vi) $\langle [\mathbf{n}]M, \mathscr{E}, \sqcup, i \rangle \rightarrow \langle \mathbf{n}, \mathscr{E}, \langle M, \mathscr{E} \rangle, i \rangle$,
(vii) $\langle \mathbf{0}, \langle \mathscr{S}, 0 \rangle :: \mathscr{E}', \langle M, \mathscr{E} \rangle, 1 \rangle \rightarrow \langle M, \mathscr{E}, \mathscr{S}, 0 \rangle$,
(viii) $\langle \mathbf{0}, \langle \mathscr{S}, 1 \rangle :: \mathscr{E}', \langle M, \mathscr{E} \rangle, 0 \rangle \rightarrow \langle M, \mathscr{E}, \mathscr{S}, 0 \rangle$,
(ix) $\langle \mathbf{0}, \langle \mathscr{S}, 1 \rangle :: \mathscr{E}', \langle M, \mathscr{E} \rangle, 1 \rangle \rightarrow \langle M, \mathscr{E}, \mathscr{S}, 1 \rangle$.

The intuitive meanings of the above moves are as follows. Moves (i) to (iv) correspond exactly to the moves of the K-machine. Move (v) corresponds to the storage of the stack together with the value of the Boolean state variable into the environment. The value of the Boolean state variable is 0 when the content of the dump must be interpreted in the context $\mu[\mathbf{0}][\ ]$, and 1 when it must be interpreted in the context $\mu[\ ]$. Moves (vi) to (ix) correspond to the restoration of the associated stack when encountering a μ-variable. *A priori* there should be four cases according to the Boolean value saved with the stack, on the one hand, and the value of the Boolean state variable in the dump, on the other hand. Nevertheless, as we shall see, one of these cases cannot occur.

7.3. *Correctness and completeness*

In order to prove that our intuition is legitimate, we must prove that the moves of the μK-machine implement correctly the weak head reduction strategy. To this end, we define an unloading function $\mathrm{D}[\![\cdot]\!]$, which transforms a dump into a $\lambda\mu$-term with explicit substitutions. The idea is, as usual: the environment is interpreted as a substitution, and the stack as a list of arguments.

Definition 7.2. (Unloading function) The unloading function $\mathrm{D}[\![\cdot]\!] : \mathbf{Dump} \rightarrow \Lambda_{\mu\sigma}$ is defined by the following clauses:

(i) $\mathrm{D}[\![\langle M, \mathscr{E}, \mathscr{S}, 0 \rangle]\!] = \mu[\mathbf{0}]\,\textsc{App}(M\{\overline{\mathscr{E}}\}, \underline{\mathscr{S}})$,

(ii) $\mathrm{D}[\![\langle M, \mathscr{E}, \mathscr{S}, 1 \rangle]\!] = \mu\,\textsc{App}(M\{\overline{\mathscr{E}}\}, \underline{\mathscr{S}})$,

where the auxiliary functions used in this definition are defined below.
The function $\underline{\cdot} : \mathbf{Stack} \rightarrow (\Lambda_{\mu\sigma})^*$, which transforms a list of closures into a list of terms, is defined as follows:

(i) $\underline{\sqcup} = \sqcup$,

(ii) $\underline{\langle M, \mathscr{E} \rangle :: \mathscr{S}} = M\{\overline{\mathscr{E}}\} :: \underline{\mathscr{S}}$.

The function $\overline{\cdot} : \mathbf{Env} \rightarrow \Sigma$, which allows environments to be transformed into substitutions, is defined as follows:

(i) $\overline{\sqcup} = id$,

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

(ii) $\overline{\langle M, \mathscr{E}' \rangle :: \mathscr{E}} = M\{\overline{\mathscr{E}'}\} \cdot \overline{\mathscr{E}}$,

(iii) $\overline{\langle \mathscr{S}, i \rangle :: \mathscr{E}} = \mathrm{S}[\![\langle \mathscr{S}, i \rangle]\!] \cdot \overline{\mathscr{E}}$.

Finally, the function $\mathrm{S}[\![\cdot]\!] : \mathbf{Stack} \times 2 \to \Lambda_{\mu\sigma}$, which transforms stacks into $\lambda\mu$-terms, is defined as follows:

(i) $\mathrm{S}[\![\langle \mathscr{S}, 0 \rangle]\!] = \lambda^{\circ}[1]\mathrm{App}(\mathbf{0}, \underline{\mathscr{L}\{\uparrow\}})$,

(ii) $\mathrm{S}[\![\langle \mathscr{S}, 1 \rangle]\!] = \lambda^{\circ}\mathrm{App}(\mathbf{0}, \underline{\mathscr{L}\{\uparrow\}})$.

We are now in position to prove the correctness and completeness of the μK-machine. The correctness property says that any move of the machine corresponds to a legal reduction of the $\lambda\mu$-calculus.

Proposition 7.3. (Correctness of the machine) Let $\mathscr{D}, \mathscr{D}' \in \mathbf{Dump}$ be such that $\mathscr{D} \to \mathscr{D}'$. Then $\mathrm{D}[\![\mathscr{D}]\!] \twoheadrightarrow_{\beta'\mu'\varepsilon'\rho'\sigma\beta'^{\circ}} \mathrm{D}[\![\mathscr{D}']\!]$. Moreover, Moves (iii) and (v) involve external β, μ, ε, or ρ-contractions; Moves (i), (ii), (iv), and (vi) are such that $\sigma(\mathrm{D}[\![\mathscr{D}]\!]) = \sigma(\mathrm{D}[\![\mathscr{D}']\!])$; Moves (vii), (viii), and (ix) are such that $|\sigma(\mathrm{D}[\![\mathscr{D}]\!])| > |\sigma(\mathrm{D}[\![\mathscr{D}']\!])|$, where $|\cdot|$ denotes the length of a term.

Proof. The proof, which is rather long but amounts mainly to algebraic manipulations of explicit substitutions, is given in Appendix B. □

The completeness property says that the machine allows any program to be evaluated. More technically, we add a single constant $\star$ and an atomic type ι to the $\lambda\mu$-calculus, and we extend its type system with the following axiom:

$$\Gamma \vdash \star : \iota.$$

This allows a *program* to be defined as a closed term of type ι. Then, the completeness property is stated as follows.

Proposition 7.4. (Completeness of the machine) Let Q be a program (that is, a closed $\lambda\mu$-term of type ι), and let $P \in \Lambda_{\mu}$ be such that $\vdash Q \Rightarrow P$. Then there exists an environment $\mathscr{E}$ such that

$$\langle P, \sqcup, \sqcup, 0 \rangle \xrightarrow{*}_1 \langle \star, \mathscr{E}, \sqcup, 0 \rangle.$$

Proof. First of all, note that $\mu\alpha.[\alpha]Q$, is a well-typed term of type ι, convertible to Q by θ-contraction. Consequently, by Proposition 5.1, $\mu[0]P$ is well-typed as well. Therefore, by Propositions 2.4, 5.1, 6.2 and 7.3, any state $\mathscr{D}$ of the machine involved in the evaluation of P is such that $\sigma\beta'^{\circ}(\mathrm{D}[\![\mathscr{D}]\!])$ is closed and well-typed. In particular, if the machine stops on a configuration of the form

$$\langle \star, \mathscr{E}, \mathscr{S}, i \rangle, \qquad (*)$$

we must have $\mathscr{S} = \sqcup$ and $i = 0$ because the other possibilities do not correspond to well-typed terms.

A priori, two things could prevent the strategy implemented by the machine from reaching a final state $(*)$: the machine could be stuck on another state, or could run for ever.

The machine cannot be stuck. The states on which the machine could possibly be stuck are the ones that do not match the left-hand side of any move:

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

(a) $\langle \mathbf{n}, \sqcup, \mathscr{S}, i\rangle$,

(b) $\langle \lambda M, \mathscr{E}, \sqcup, i\rangle$,

(c) $\langle [\mathbf{n}]\, M, \mathscr{E}, \mathscr{S}, i\rangle$, where $\mathscr{S} \neq \sqcup$,

(d) $\langle \mathbf{0}, \langle \mathscr{S}, i\rangle :: \mathscr{E}', \mathscr{S}', i'\rangle$, where $\mathscr{S}'$ is not of the form $\langle M, \mathscr{E}\rangle$,

(e) $\langle \mathbf{0}, \langle \mathscr{S}, 0\rangle :: \mathscr{E}', \langle M, \mathscr{E}\rangle, 0\rangle$.

In State (a), $\mathbf{n}$ corresponds to a free variable and this contradicts the fact that P is closed. State (b) corresponds to a functional term, which contradicts the fact that P is of atomic type. In State (c), the term $[\mathbf{n}]\, M$ must be of type $\bot$. Consequently, the stack $\mathscr{S}$ must be empty. In State (d), $\mathbf{0}$ stands for a μ-variable because its value in the environment is a stack. Therefore a state such as (d) can only be reached by a move of type (vi) followed by a (possibly empty) sequence of moves of type (ii). This implies that the stack $\mathscr{S}'$ in State (d) must be made of a single closure. Finally State (e) is such that

$$\begin{array}{lll} \mathrm{D}[\![\langle \mathbf{0}, \langle \mathscr{S}, 0\rangle :: \mathscr{E}', \langle M, \mathscr{E}\rangle, 0\rangle]\!] & = & \mu[0](\mathbf{0}\{\lambda^{\circ}[1]\textsc{App}(\mathbf{0}, \underline{S}\{\uparrow\}) \cdot id\})(M\{\overline{\mathscr{E}}\}) \\ & \to_{\sigma} & \mu[0](\lambda^{\circ}[1]\textsc{App}(\mathbf{0}, \underline{S}\{\uparrow\}))(M\{\overline{\mathscr{E}}\}) \\ & \to_{\beta' c irc} & \mu[0]([1]\textsc{App}(\mathbf{0}, \underline{S}\{\uparrow\}))\{M\{\overline{\mathscr{E}}\} \cdot id\} \\ & \twoheadrightarrow_{\sigma} & \mu[0]([0]\textsc{App}(M\{\overline{\mathscr{E}}\}, \underline{S})). \end{array}$$

Now, if state e was reachable, the term

$$\sigma\beta'^{\circ}(\mu[0]([0]\textsc{App}(M\{\overline{\mathscr{E}}\}, \underline{S}))) \qquad (**)$$

would be a reduct of P and, consequently, would be a closed well-typed term. But this is not possible because the only type that can be assigned to a term like (**) is $\bot$, and then the fact that (**) is closed would contradict the consistency of classical logic.

The machine may not run for ever. We distinguish between three types of moves: (A) Moves (iii) and (v); (B) Moves (vii), (viii) and (ix); (C) Moves (i), (ii), (iv), (vi). By Propositions 6.3 and 7.3, an inifinite sequence of moves may not contain infinitely many moves of Type (A) because it would be possible to construct an infinite sequence of $\beta\mu\varepsilon\rho$-contractions starting from $\mu\alpha.[\alpha]\, Q$, which contradicts Proposition 2.5. Similarly, an infinite sequence of moves of Type (B) and (C) cannot contain infinitely many moves of Type (B) because these moves decrease the length of the σ-normal forms of the unloaded terms, while moves of Type (C) keep it unchanged. Finally moves of Type (C) decrease the structural complexity of the pair made of the two first components of the machine (*i.e.*, the term and the environment). Consequently, there cannot be an inifinite sequence of moves of Type (C) either. □

7.4. *The μK-abstract machine: simplified definition*

It is worth commenting on the proof of Proposition 7.4. Let us go back to the different states on which the machine could possibly be stuck. The nature of the arguments used to reject the different possibilities differs from one case to the next. States (a), (c) and (d) are rejected using a simple typing argument. On the other hand, the argument used to reject State (e) is deeper. It does not rely only on typing but also on the consistency of classical

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

logic. Indeed, if one destroys the logical concistency of the typing system, Proposition 7.4 no longer holds. For instance, if we add to the calculus a constant Ω of type $\iota \to \bot$, then the evaluation of the $\lambda\mu$-term $\mu\alpha.[\alpha]([\alpha](\Omega\star))$ will be stuck on a state similar to State (e). Finally, State (b) is rejected because Proposition 7.4 concerns programs, *i.e.*, terms of atomic type. Nevertheless, State (b) is a perfectly acceptable final state. It simply corresponds to the usual fact that a term of functional type is evaluated to a closure.

Imagine that we load the machine with a well-typed term M and that it eventually reaches a final state akin to State (b), *i.e.*,

$$\langle M, \sqcup, \sqcup, 0\rangle \xrightarrow{*} \langle \lambda N, \mathscr{E}, \sqcup, i\rangle.$$

If $i = 1$, then $\mathrm{D}[\![\langle \lambda N, \mathscr{E}, \sqcup, i\rangle]\!] = \mu((\lambda N)[\overline{\mathscr{E}}])$. This would imply that $(\lambda N)[\overline{\mathscr{E}}]$ is of type $\bot$, which is impossible. Therefore, when a final state akin to State (b) is reached, we have that $i = 0$. Hence, when the machine is loaded with a closed well-typed term, it eventually reaches a final state of the form $\langle M, \mathscr{E}, \sqcup, 0\rangle$, where M is either a λ-abstraction or the constant $\star$.

Now the value of the Boolean state variable, which is needed when unloading the machine, does not play any computational role. Indeed, in the course of a computation, the value of the Boolean state variable at some given moment (together with some Boolean value possibly saved in the environement) is only needed in order to compute its next value. In particular, Moves (vii), (viii) and (ix) in Definition 7.1 do not differ except in the handling of the Boolean state variable. Therefore, since we know that the final value of this variable is necessarily 0 when the machine is loaded with a well-typed closed term, we may simplify Definition 7.1 as follows.

Definition 7.5. (μK-machine – simplified definition) The set **Dump** of the states of the μK-machine is defined by the following equations:

$$\begin{array}{rcl} \textbf{Dump} & = & \Lambda_\mu \times \textbf{Env} \times \textbf{Stack} \\ \textbf{Env} & = & (\textbf{Stack} \cup \textbf{Closure})^* \\ \textbf{Stack} & = & \textbf{Closure}^* \\ \textbf{Closure} & = & \Lambda_\mu \times \textbf{Env}. \end{array}$$

Let $M, N \in \Lambda_\mu$; $\mathscr{E}, \mathscr{E}' \in \textbf{Env}$; $\mathscr{S} \in \textbf{Stack}$; $ccl \in \textbf{Stack} \cup \textbf{Closure}$; $cl \in \textbf{Closure}$. The moves of the μK-machine are specified by the following transition system:

(i) $\langle \mathbf{0}, \langle M, \mathscr{E}\rangle :: \mathscr{E}', \mathscr{S}\rangle \to \langle M, \mathscr{E}, \mathscr{S}\rangle$,
(ii) $\langle \mathbf{n{+}1}, ccl :: \mathscr{E}, \mathscr{S}\rangle \to \langle \mathbf{n}, \mathscr{E}, \mathscr{S}\rangle$,
(iii) $\langle \lambda M, \mathscr{E}, cl :: \mathscr{S}\rangle \to \langle M, cl :: \mathscr{E}, \mathscr{S}\rangle$,
(iv) $\langle (MN), \mathscr{E}, \mathscr{S}\rangle \to \langle M, \mathscr{E}, \langle N, \mathscr{E}\rangle :: \mathscr{S}\rangle$,
(v) $\langle \mu M, \mathscr{E}, \mathscr{S}\rangle \to \langle M, \mathscr{S} :: \mathscr{E}, \sqcup\rangle$,
(vi) $\langle [n]M, \mathscr{E}, \sqcup\rangle \to \langle \mathbf{n}, \mathscr{E}, \langle M, \mathscr{E}\rangle\rangle$,
(vii) $\langle \mathbf{0}, \mathscr{S} :: \mathscr{E}', \langle M, \mathscr{E}\rangle\rangle \to \langle M, \mathscr{E}, \mathscr{S}\rangle$.

The unloading function may be adapted to the above definition as follows:

$$\mathrm{D}[\![\langle M, \mathscr{E}, \mathscr{S}]\!] = \textsc{App}(M\{\overline{\mathscr{E}}\}, \underline{\mathscr{S}}).$$

However, Proposition 7.3 no longer holds with Definition 7.5 together with the above

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

unloading function. Nevertheless, as established by the above discussion, Propositions 7.3 and 7.4 may be replaced by the following one.

Proposition 7.6. (Completeness and correctness of the simplified machine) Let M be a closed well-typed $\lambda\mu$-term and let $N \in \Lambda_\mu$ be such that $\vdash M \Rightarrow N$. Then there exists an environment $\mathscr{E}$ and a term $O \in \Lambda_\mu$ such that

(1) $\langle N, \sqcup, \sqcup\rangle \xrightarrow{*} \langle O, \mathscr{E}, \sqcup\rangle$

(2) $N =_\theta \mu[\mathbf{0}]\, N \twoheadrightarrow_{\beta'\mu'\varepsilon'\rho'\sigma\beta'^\circ} \mu[\mathbf{0}]\,(O\{\overline{\mathscr{E}}\}) =_\theta O\{\overline{\mathscr{E}}\}$.

Moreover, O is either $\star$ or a λ-abstraction.

Appendix A. Proof of Proposition 2.5

In this appendix we prove the strong-normalisation of (our variant of) the $\lambda\mu$-calculus. A proof that the original system is strongly normalisable with respect to the β and μ-reductions relations (in the second order case) may be found in Parigot (1993).

Our proof is organised in three parts:

— We prove that any typable $\lambda\mu$-term is strongly $\beta\mu$-normalisable. To this end, we use the Tait–Girard reducibility method.

— We prove that any typable $\lambda\mu$-term is strongly $\rho\varepsilon\theta$-normalisable.

— We prove that the $\rho\varepsilon\theta$-contractions may be postponed with respect to the $\beta\mu$-contractions.

The strong normalisation proposition for the entire reduction system immediately follows from these three properties.

For the sake of simplicity, we consider a family $(\mathscr{X}_\tau)_{\tau\in\text{type}}$ of disjoint alphabets of λ-variables indexed by types, and a family $(\mathscr{A}_\sigma)_{\sigma\in\text{cotype}}$ of disjoint alphabets of μ-variables indexed by cotypes. We define

$$\Omega = \bigcup_{\tau\in\text{type}} \mathscr{X}_\tau \cup \bigcup_{\sigma\in\text{cotype}} \mathscr{A}_\sigma.$$

Ω may be consisdered as an infinite typing context where each λ-variable $x \in \mathscr{X}_\tau$ (respectively, each μ-variable $\alpha \in \mathscr{A}_\sigma$) is declared of type τ (respectively, of cotype σ). Then we say that a $\lambda\mu$-term M is typable (according to Ω) with type τ if there exists a finite context $\Gamma \subset \Omega$ such that $\Gamma \vdash M : \tau$. Clearly, any $\lambda\mu$-term typable according to the system of Subsection 2.2 is typable according to Ω (up to free-variable renaming). We write simply $M : \tau$ when a $\lambda\mu$-term M is typable with type τ according to Ω.

We also introduce the following notation. We write $\overline{N}$ for sequences of $\lambda\mu$-terms (including the empty one). Let $\overline{N} = N_0, \ldots, N_n$ and $\overline{x} = x_0, \ldots, x_n$. We write $M\,\overline{N}$ for the application $(M\,N_0)\ldots N_n$, and, when $\overline{N}$ is the empty sequence, $M\,\overline{N} \equiv M$. We write $M[x_i := N_i]_{i\in n}$ for the simultaneous substitution $M[\overline{x} := \overline{N}]$. Finally, if R is a notion of reduction, we write '$\to_R$', for the R-contraction relation (*i.e.*, the one-step reduction relation), and we write '$\xrightarrow{+}_R$' (respectively, '$\twoheadrightarrow_R$') for its transitive closure (respectively, transitive reflexive closure). These three relations may be axiomatised by means of formal systems, which allow inductive proofs to be performed (Barendregt 1984, Chapter 3, §1).

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Strong $\beta\mu$-normalisation

Let $\mathrm{SN}_{\beta\mu}$ denote the set of strongly $\beta\mu$-normalisable $\lambda\mu$-terms. The family of sets of reducible terms is defined, as usual, by induction on the types.

(i) $[\![\tau]\!] = \{M : \tau \mid M \in \mathrm{SN}_{\beta\mu}\}$, whenever τ is atomic.
(ii) $[\![\sigma \to \tau]\!] = \{M : \sigma \to \tau \mid \forall N \in [\![\sigma]\!], (M\,N) \in [\![\tau]\!]\}$.

Lemma A.1. Let τ be any type:

(a) $x\,\overline{N} \in [\![\tau]\!]$, for any λ-variable x and any sequence of strongly normalisable terms $\overline{N}$ such that $x\,\overline{N} : \tau$.
(b) $[\![\tau]\!] \subset \mathrm{SN}_{\beta\mu}$.

Proof. The proof is by induction on the structure of τ.

Basic case: τ is atomic.

(a) By hypothesis, every term in the sequence $\overline{N}$ is strongly normalisable. Hence $x\,\overline{N} \in \mathrm{SN}_{\beta\mu}$, which implies $x\,\overline{N} \in [\![\tau]\!]$.
(b) This is true by definition.

Induction case: $\tau \equiv \tau_1 \to \tau_2$.

(a) Let $N \in [\![\tau_1]\!]$. By induction hypothesis (b), N is strongly normalisable. Then, by induction hypothesis (a), $x\,\overline{N}\,N \in [\![\tau_2]\!]$. Hence, $x\,\overline{N} \in [\![\tau_1 \to \tau_2]\!]$.
(b) Let $M \in [\![\tau_1 \to \tau_2]\!]$, and let x be some variable of type τ_1. By induction hypothesis (a), $x \in [\![\tau_1]\!]$, which implies $M\,x \in [\![\tau_2]\!]$. Hence, by induction hypothesis (b), $M\,x$ is strongly normalisable, and so is M. □

Lemma A.2. Let $\lambda x.\,M : \sigma \to \tau$ be such that $M[x := N] \in [\![\tau]\!]$ whenever $N \in [\![\sigma]\!]$. Then $\lambda x.\,M \in [\![\sigma \to \tau]\!]$.

Proof. By definition, $\lambda x.\,M \in [\![\sigma \to \tau]\!]$ if and only if $(\lambda x.\,M)\,N\,\overline{O} \in \mathrm{SN}_{\beta\mu}$, for any $N \in [\![\sigma]\!]$, and any sequence $\overline{O}$ of reducible terms such that $(\lambda x.\,M)\,N\,\overline{O}$ is of atomic type. Let N and $\overline{O}$ be such a term and such a sequence of terms, and suppose $(\lambda x.\,M)\,N\,\overline{O} \notin \mathrm{SN}_{\beta\mu}$. By hypothesis, the $\lambda\mu$-terms M, N and $\overline{O}$ are reducible, and therefore, by Lemma A.1., Property (b), strongly normalisable. Consequently, any infinite reduction sequence starting from $(\lambda x.\,M)\,N\,\overline{O}$ cannot entirely consist of reduction within the subterms M, N and $\overline{O}$. Hence, such an infinite reduction sequence should obey the following scheme:

$$(\lambda x.\,M)\,N\,\overline{O} \twoheadrightarrow_{\beta\mu} (\lambda x.\,M')\,N'\,\overline{O}' \to_{\beta} M'[x := N']\,\overline{O}' \twoheadrightarrow_{\beta\mu} \cdots$$

But then, we could construct another infinite sequence starting from $M[x := N]\,\overline{O}$:

$$M[x := N]\,\overline{O} \twoheadrightarrow_{\beta\mu} M'[x := N']\,\overline{O}' \twoheadrightarrow_{\beta\mu} \cdots$$

contradicting the fact that $M[x := N]$ is reducible. □

Lemma A.3. Let $(\mu\alpha.\,M)\,\overline{N}$ and P be $\lambda\mu$-terms such that $(\mu\alpha.\,M)\,\overline{N} \twoheadrightarrow_{\beta\mu} P$. Then there exist a $\lambda\mu$-term M', and four sequences of $\lambda\mu$-terms $\overline{N}_1$, $\overline{N}_2$, $\overline{N}'_1$, $\overline{N}'_2$, such that:

(a) $\overline{N} \equiv \overline{N}_1, \overline{N}_2$;
(b) $P \equiv (\mu\beta.\,M')\,\overline{N}'_2$;

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

(c) $\overline{N}_1 \twoheadrightarrow_{\beta\mu} \overline{N}'_1$ and $\overline{N}_2 \twoheadrightarrow_{\beta\mu} \overline{N}'_2$;
(d) $M[\alpha := \lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{N}_1)] \twoheadrightarrow_{\beta\mu} M'$.

Proof. The proof is by a straightforward induction on the number of contraction steps in the reduction $(\mu\alpha.\,M)\,\overline{N} \twoheadrightarrow_{\beta\mu} P$. □

Lemma A.4. Let $\mu\alpha.\,M : \tau$ be a $\lambda\mu$-term such that $M[\alpha := \lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{N})] \in [\![\bot]\!]$, whenever $\overline{N}$ is a sequence of reducible terms such that $\lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{N}) : \tau^\bot$. Then $\mu\alpha.\,M \in [\![\tau]\!]$.

Proof. The proof is similar to that for Lemma A.2. Let $\overline{N}$ be a sequence of reducible terms, such that $(\mu\alpha.\,M)\,\overline{N}$ is of atomic type. Then, by Lemma A.3., any infinite reduction sequence starting from $(\mu\alpha.\,M)\,N\,\overline{O}$ could be turned into an infinite reduction sequence of the following form:

$$\begin{array}{rcl}(\mu\alpha.\,M)\,\overline{N} & \equiv & (\mu\alpha.\,M)\,\overline{N}_1\,\overline{N}_2 \\ & \twoheadrightarrow_{\beta\mu} & (\mu\alpha.\,M')\,\overline{N}'_1\,\overline{N}'_2 \\ & \twoheadrightarrow_{\mu} & (\mu\beta.\,M'[\alpha := \lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{N}'_1)])\,\overline{N}'_2 \\ & \twoheadrightarrow_{\beta\mu} & \cdots\end{array}$$

where the rest of the sequence would consist only of contractions taking place within $M'[\alpha := \lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{N}'_1)]$. But then, we could construct another infinite sequence:

$$M[\alpha := \lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{N}_1)] \twoheadrightarrow_{\beta\mu} M'[\alpha := \lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{N}'_1)] \twoheadrightarrow_{\beta\mu} \cdots$$

contradicting the reducibility of $M[\alpha := \lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{N}_1)]$. □

Lemma A.5. If $M : \tau$, then $M \in [\![\tau]\!]$.

Proof. Let $(x_i)_{i\in n}$ be the sequence of free λ-variables of M, and let $(\alpha_i)_{i\in m}$ be its sequence of free μ-variables. We prove that

$$M[x_i := N_i]_{i\in n}[\alpha_i := \lambda^\circ f.\,\lceil\beta_i\rceil\,(f\,\overline{O}_i)]_{i\in m} \in [\![\tau]\!]$$

for any sequence of reducible terms $(N_i)_{i\in n}$ that agrees on the types of the free λ-variables of M, and any sequence of sequences of reducible terms $(\overline{O}_i)_{i\in n}$ such that the sequence of names $(\lambda^\circ f.\,\lceil\beta_i\rceil\,(f\,\overline{O}_i))_{i\in m}$ agrees on the cotypes of the free μ-variables of M. Then the proposition will be established by taking $N_i \equiv x_i$ and $\lambda^\circ f.\,\lceil\beta_i\rceil\,(f\,\overline{O}_i) \equiv \lambda^\circ f.\,\lceil\alpha_i\rceil\,f$.

The proof is by induction on the strucuture of M. For the sake of conciseness, we write M^* for $M[x_i := N_i]_{i\in n}[\alpha_i := \lambda^\circ f.\,\lceil\beta_i\rceil\,(f\,\overline{O}_i)]_{i\in m}$.

Case 1: $M \equiv x_i$. We have $M^* \equiv N_i$, which is reducible by hypothesis.

Case 2: $M \equiv \lambda x.\,M_1$. We must have $\tau \equiv \tau_1 \to \tau_2$. By the induction hypothesis, we have $M_1^*[x := N] \in [\![\tau_2]\!]$, for any $N \in [\![\tau_1]\!]$. Hence, by Lemma A.2, $\lambda x.\,M_1^* \in [\![\tau_1 \to \tau_2]\!]$.

Case 3: $M \equiv M_1\,M_2$. By the induction hypothesis, M_1^* and M_2^* are both reducible, from which it follows that $M_1^*\,M_2^* \equiv (M_1\,M_2)^*$ is reducible.

Case 4: $M \equiv \mu\alpha.\,M_1$. By the induction hypothesis, we have $M_1^*[\alpha := \lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{O})] \in [\![\bot]\!]$ for any sequence of reducible terms $\overline{O}$ such that $\lambda^\circ f.\,\lceil\beta\rceil\,(f\,\overline{O}) : \tau^\bot$. Then we are done by Lemma A.4.

Case 5: $M \equiv \lceil\alpha_i\rceil\,M_1$. We have that $\tau \equiv \bot$. Consequently, $(\lceil\alpha_i\rceil\,M_1)^* \in [\![\bot]\!]$ if and only if

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

$([\alpha_i]\, M_1)^* \in \mathrm{SN}_{\beta\mu}$. By the induction hypothesis, M_1^* is reducible, and so is $M_1^*\, \overline{O}_i$. Then by Lemma A.1 Property (b), $M_1^*\, \overline{O}_i$ is strongly normalisable, and so is $[\alpha]\,(M_1^*\, \overline{O}_i) \equiv ([\alpha_i]\, M_1)^*$. □

Strong $\rho\varepsilon\theta$-normalisation

Lemma A.6. Any $\lambda\mu$-term is strongly $\rho\varepsilon\theta$-normalisable.

Proof. The proof is immediate because any ρ, ε or θ-contraction decreases the length of the contracted $\lambda\mu$-term. □

Postponement of the $\rho\varepsilon\theta$-contractions

Lemma A.7. Let R stand for ρ, ε or θ, and let R' stand only for ε or θ.

(a) If $M \to_R N$, then $M[x := O] \to_R N[x := O]$.
(b) If $N \to_R O$, then $M[x := N] \twoheadrightarrow_R M[x := O]$.
(c) If $M \to_{R'} N$, then $M[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)] \to_{R'} N[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)]$.
(d) If $M \to_\rho N$, then either $M[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)] \to_\rho N[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)]$, or $M[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)] \to_\mu P \to_\rho N[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)]$, for some term P.
(e) If $N \to_R O$, then $M[\alpha := \lambda^\circ f.\,[\beta]\,(f\,N)] \twoheadrightarrow_R N[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)]$.

Proof. The proof is by induction on the structure of M. Note that Property (c) does not hold for the notion of ρ-reduction, and must be replaced by Property (d). This is due to the case where $M \equiv \mu\gamma.\,[\alpha]\,(\mu\delta.\,M')$ and $N \equiv \mu\gamma.\,M'[\delta := \lambda^\circ f.\,[\alpha]\,f]$. Then, we have

$$\begin{aligned}
&\mu\gamma.\,[\alpha]\,(\mu\delta.\,M')[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)]\\
&\quad\equiv\quad \mu\gamma.\,[\beta]\,((\mu\delta.\,M'[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)])\,O)\\
&\quad\to_\mu\quad \mu\gamma.\,[\beta]\,(\mu\delta.\,M'[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)][\delta := \lambda^\circ f.\,[\delta]\,(f\,O)])\\
&\quad\to_\rho\quad \mu\gamma.\,M'[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)][\delta := \lambda^\circ f.\,[\delta]\,(f\,O)][\delta := \lambda^\circ f.\,[\beta]\,f]\\
&\quad\equiv\quad \mu\gamma.\,M'[\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)][\delta := \lambda^\circ f.\,[\beta]\,(f\,O)]\\
&\quad\equiv\quad \mu\gamma.\,M'[\delta := \lambda^\circ f.\,[\alpha]\,f][\alpha := \lambda^\circ f.\,[\beta]\,(f\,O)].
\end{aligned}$$

□

Lemma A.8. Let R stand for β or μ, and let M and N be $\lambda\mu$-terms such that

$$M[\alpha := \lambda^\circ f.\,[\beta]\,f] \to_R N.$$

Then there exist a $\lambda\mu$-term O such that

(a) $N \equiv O[\alpha := \lambda^\circ f.\,[\beta]\,f]$,
(b) $M \to_R O$.

Proof. The proof is by induction on the derivation of $M[\alpha := \lambda^\circ f.\,[\alpha]\,f] \to_R N$. □

Lemma A.9. Let R stand for β or μ, let $M : \tau$ and $N : \tau$, and let α be a μ-variable of cotype $\perp^\perp$ such that

$$M[\alpha := \lambda^\circ f.\,f] \to_R N.$$

Then there exist a $\lambda\mu$-term O such that

(a) $N \equiv O[\alpha := \lambda^\circ f.\,f]$,
(b) $M \to_R O$.

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Proof. The proof is by induction on the derivation of $M[\alpha := \lambda^\circ f.f] \to_R N$. Contrary to the case of Lemma A.8, we need the extra hypothesis that the $\lambda\mu$-terms are well-typed. This eliminates the following *pathological* case:

$$\begin{array}{rcl}((\lceil\alpha\rceil\,\lambda x.P)\,Q)[\alpha := \lambda^\circ f.f] & \equiv & (\lambda x.P[\alpha := \lambda^\circ f.f])\,Q[\alpha := (\lambda^\circ f.f)] \\ & \to_\beta & P[x := Q][\alpha := \lambda^\circ f.f].\end{array}$$

Indeed the term $\lceil\alpha\rceil\,\lambda x.P$ must be of type $\bot$, and therefore the application $(\lceil\alpha\rceil\,\lambda x.P)\,Q$ is not typable. The same phenomenon allows μ-redexes to be created by *untyped* ε-contractions. □

Lemma A.10. (ρ/β-postponement) Let M, N, O be such that $M \to_\rho N \to_\beta O$. Then there exists a $\lambda\mu$-term P such that $M \to_\beta P \twoheadrightarrow_\rho O$.

Proof. The proof is by induction on the derivation of $M \to_\rho N$ using Lemma A.7 Properties (a) and (b), with $R = \rho$, and Lemma A.8, with $R = \beta$. □

Lemma A.11. (ρ/μ-postponement) Let M, N, O be such that $M \to_\rho N \to_\mu O$. Then there exists a $\lambda\mu$-term P such that $M \overset{+}{\to}_\mu P \twoheadrightarrow_\rho O$.

Proof. The proof is by induction on the derivation of $M \to_\rho N$ using Lemma A.7 Properties (d) and (e), with $R = \rho$, and Lemma A.8, with $R = \mu$. □

Lemma A.12. (ε/β-postponement) Let $M, N, O : \tau$ be such that $M \to_\varepsilon N \to_\beta O$. Then there exists a $\lambda\mu$-term $P : \tau$ such that $M \to_\beta N \twoheadrightarrow_\varepsilon O$.

Proof. The proof is by induction on the derivation of $M \to_\varepsilon N$ using Lemma A.7 Properties (a) and (b), with $R = \varepsilon$, and Lemma A.9, with $R = \beta$. □

Lemma A.13. (ε/μ-postponement) Let $M, N, O : \tau$ be such that $M \to_\varepsilon N \to_\mu O$. Then there exist a $\lambda\mu$-term $P : \tau$ such that $M \to_\mu P \twoheadrightarrow_\varepsilon O$.

Proof. The proof is by induction on the derivation of $M \to_\varepsilon N$ using Lemma A.7 Properties (c) and (e), with $R = \varepsilon$, and Lemma A.9, with $R = \mu$. □

Lemma A.14. (θ/β-postponement) Let M, N, O be such that $M \to_\theta N \to_\beta O$. Then there exists a $\lambda\mu$-term P such that $M \overset{+}{\to}_{\beta\mu} P \twoheadrightarrow_\theta O$.

Proof. The proof is by induction on the derivation of $M \to_\theta N$ using Lemma A.7 Properties (a) and (b), with $R = \theta$. Note the unusual form of this postponement statement. Indeed, postponing θ with respect to β may introduce μ-contractions. This corresponds to the following case:

$$(\mu\alpha.\lceil\alpha\rceil\,(\lambda x.Q))\,R \to_\theta (\lambda x.Q)\,R \to_\beta Q[x := R],$$

which is transformed into:

$$(\mu\alpha.\lceil\alpha\rceil\,(\lambda x.Q))\,R \to_\mu \mu\alpha.\lceil\alpha\rceil\,((\lambda x.Q)\,R) \to_\beta \mu\alpha.\lceil\alpha\rceil\,Q[x := R] \to_\theta Q[x := R]$$ □

Lemma A.15. (θ/μ-postponement) Let M, N, O be such that $M \to_\theta N \to_\mu O$. Then there exists a $\lambda\mu$-term P such that $M \overset{+}{\to}_\mu P \twoheadrightarrow_\theta O$.

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Proof. The proof is by induction on the derivation of $M \to_\theta N$ using Lemma A.7 Properties (c) and (e), with $R = \theta$. □

Appendix B. Proof of Proposition 7.3

We first state and prove two technical lemmas.

Lemma B.1. Let $M \in \Lambda_{\mu\sigma}$, and let $\mathscr{L} \in \Lambda^*_{\mu\sigma}$ be non-empty. Then

$$\textsc{App}(\mu M, \mathscr{L}) \twoheadrightarrow_{\sigma\beta'\circ\mu'} \mu(M\{\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0}, \mathscr{L}\{\uparrow\circ\uparrow\}) \cdot \uparrow\})$$

Proof. We proceed by induction on $\mathscr{L}$:

Base case: $\mathscr{L} = (N :: \sqcup)$.

$$\begin{array}{rcl}
\textsc{App}(\mu M, N :: \sqcup) & = & (\mu M)N \\
& \to_{\mu'} & \mu(M\{\lambda^\circ[\mathbf{1}]\mathbf{0}(N\{\uparrow\circ\uparrow\}) \cdot \uparrow\}) \\
& = & \mu(M\{\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0}, (N :: \sqcup)\{\uparrow\circ\uparrow\}) \cdot \uparrow\}).
\end{array}$$

Induction step: $\mathscr{L} = (N :: \mathscr{L}')$.

For the sake of conciseness, we define the following abbreviations:

$$A = \lambda^\circ[\mathbf{1}]\mathbf{0}(N\{\uparrow\circ\uparrow\}), \quad B = \lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0}, \mathscr{L}'\{\uparrow\circ\uparrow\}), \quad C = \mathbf{0}(N\{\uparrow\circ\uparrow\}).$$

Then we have:

$$\begin{array}{rcll}
\textsc{App}(\mu M, N :: \mathscr{L}') & = & \textsc{App}((\mu M)N, \mathscr{L}') & \\
& \to_{\mu'} & \textsc{App}(\mu(M\{A \cdot \uparrow\}), \mathscr{L}') & \\
& \twoheadrightarrow_{\sigma\beta'\circ\mu'} & \mu(M\{A \cdot \uparrow\}\{B \cdot \uparrow\}) & \text{By the induction hypothesis.} \\
& \twoheadrightarrow_{\sigma} & \mu(M\{(A \cdot \uparrow) \circ (B \cdot \uparrow)\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{A\{B \cdot \uparrow\} \cdot (\uparrow \circ (B \cdot \uparrow))\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{A\{B \cdot \uparrow\} \cdot \uparrow\}) & \\
& = & \mu(M\{(\lambda^\circ[\mathbf{1}]\mathbf{0}(N\{\uparrow\circ\uparrow\}))\{B \cdot \uparrow\} \cdot \uparrow\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{\lambda^\circ(\mathbf{1}\{\Uparrow(B\cdot\uparrow)\}(\mathbf{0}\{\Uparrow(B\cdot\uparrow)\}(N\{\uparrow\circ\uparrow\}\{\Uparrow(B\cdot\uparrow)\}))) \cdot \uparrow\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{\lambda^\circ(\mathbf{0}\{(B\cdot\uparrow) \circ \uparrow\}(\mathbf{0}(N\{\uparrow\circ\uparrow \circ \Uparrow(B\cdot\uparrow)\}))) \cdot \uparrow\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{\lambda^\circ(B\{\uparrow\}(\mathbf{0}(N\{\uparrow \circ (B\cdot\uparrow)\circ\uparrow\}))) \cdot \uparrow\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{\lambda^\circ(B\{\uparrow\}(\mathbf{0}(N\{\uparrow\circ\uparrow\}))) \cdot \uparrow\}) & \\
& = & \mu(M\{\lambda^\circ((\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0}, \mathscr{L}'\{\uparrow\circ\uparrow\}))\{\uparrow\}C)) \cdot \uparrow\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{\lambda^\circ((\lambda^\circ[\mathbf{2}]\textsc{App}(\mathbf{0}, \mathscr{L}'\{\uparrow\circ\uparrow\}\{\Uparrow(\uparrow)\}))C)) \cdot \uparrow\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{\lambda^\circ((\lambda^\circ[\mathbf{2}]\textsc{App}(\mathbf{0}, \mathscr{L}'\{\uparrow\circ\uparrow\circ\uparrow\}))C)) \cdot \uparrow\}) & \\
& \to_{\beta'\circ} & \mu(M\{\lambda^\circ([\mathbf{2}]\textsc{App}(\mathbf{0}, \mathscr{L}'\{\uparrow\circ\uparrow\circ\uparrow\})\{C \cdot id\})) \cdot \uparrow\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{\lambda^\circ([\mathbf{1}]\textsc{App}(C, \mathscr{L}'\{\uparrow\circ\uparrow\circ\uparrow \circ (C \cdot id)\}))) \cdot \uparrow\}) & \\
& \twoheadrightarrow_{\sigma} & \mu(M\{\lambda^\circ([\mathbf{1}]\textsc{App}(C, \mathscr{L}'\{\uparrow\circ\uparrow\}))) \cdot \uparrow\}) & \\
& = & \mu(M\{\lambda^\circ([\mathbf{1}]\textsc{App}(\mathbf{0}(N\{\uparrow\circ\uparrow\}, \mathscr{L}'\{\uparrow\circ\uparrow\}))) \cdot \uparrow\}) & \\
& = & \mu(M\{\lambda^\circ([\mathbf{1}]\textsc{App}(\mathbf{0}, (N :: \mathscr{L}')\{\uparrow\circ\uparrow\}))) \cdot \uparrow\}). &
\end{array}$$

□

Lemma B.2. Let $\mathscr{L} \in \Lambda_{\mu\sigma}$, and let s be a substitution. Then

$$(\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0}, \mathscr{L}\{\uparrow \circ \uparrow\}) \cdot (s \circ \uparrow)) \circ (\lambda^\circ[\mathbf{1}]\mathbf{0} \cdot id) \twoheadrightarrow_{\sigma\beta'\circ} (\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0}, \mathscr{L}\{\uparrow\}) \cdot s).$$

Proof. For the sake of conciseness, we define the following abbreviations:

$$A = \lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0}, \mathscr{L}\{\uparrow\circ\uparrow\}), \quad B = \lambda^\circ[\mathbf{1}]\mathbf{0}.$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Then we have:

$$\begin{array}{rcl}
(A\cdot(s\circ\uparrow))\circ(B\cdot id) & \twoheadrightarrow_\sigma & A\{B\cdot id\}\cdot(s\circ\uparrow\circ(B\cdot id))\\
& \twoheadrightarrow_\sigma & A\{B\cdot id\}\cdot s\\
& = & (\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0},\mathscr{L}\{\uparrow\circ\uparrow\}))\{B\cdot id\}\cdot s\\
& \twoheadrightarrow_\sigma & (\lambda^\circ\mathbf{1}\{\Uparrow(B\cdot id)\}\textsc{App}(\mathbf{0}\{\Uparrow(B\cdot id)\},\mathscr{L}\{\uparrow\circ\uparrow\}\{\Uparrow(B\cdot id)\}))\cdot s\\
& \twoheadrightarrow_\sigma & (\lambda^\circ B\{\uparrow\}\textsc{App}(\mathbf{0},\mathscr{L}\{\uparrow\circ\uparrow\circ\Uparrow(B\cdot id)\}))\cdot s\\
& \twoheadrightarrow_\sigma & (\lambda^\circ B\{\uparrow\}\textsc{App}(\mathbf{0},\mathscr{L}\{\uparrow\circ(B\cdot id)\circ\uparrow\}))\cdot s\\
& \twoheadrightarrow_\sigma & (\lambda^\circ B\{\uparrow\}\textsc{App}(\mathbf{0},\mathscr{L}\{\uparrow\}))\cdot s\\
& = & (\lambda^\circ(\lambda^\circ[\mathbf{1}]\mathbf{0})\{\uparrow\}\textsc{App}(\mathbf{0},\mathscr{L}\{\uparrow\}))\cdot s\\
& \twoheadrightarrow_\sigma & (\lambda^\circ(\lambda^\circ[\mathbf{2}]\mathbf{0})\textsc{App}(\mathbf{0},\mathscr{L}\{\uparrow\}))\cdot s\\
& \to_{\beta'^\circ} & (\lambda^\circ([\mathbf{2}]\mathbf{0})\{\textsc{App}(\mathbf{0},\mathscr{L}\{\uparrow\})\cdot id\})\cdot s\\
& \twoheadrightarrow_\sigma & (\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0},\mathscr{L}\{\uparrow\}))\cdot s.
\end{array}$$

□

We are now in a position to prove Proposition 7.3. The first four moves of the machine correspond exactly to those of the Krivine machine. Therfore, we only focus on Moves (v), (vi) and (vii) – Moves (viii) and (ix) being similar to Move (vii).

Move (v): $\mathscr{D}=\langle\mu M,\mathscr{E},\mathscr{S},i\rangle$ *and* $\mathscr{D}'=\langle M,\langle\mathscr{S},i\rangle::\mathscr{E},\sqcup,1\rangle$. We take $i=0$, the case where $i=1$ being similar. Then, we have

$$\begin{array}{rcll}
\mathrm{D}[\![\mathscr{D}]\!] & = & \mathrm{D}[\![\langle\mu M,\mathscr{E},\mathscr{S},0\rangle]\!] & \\
& = & \mu[\mathbf{0}]\textsc{App}((\mu M)\{\overline{\mathscr{E}}\},\underline{\mathscr{L}}) & \\
& \twoheadrightarrow_\sigma & \mu[\mathbf{0}]\textsc{App}(\mu(M\{\Uparrow(\overline{\mathscr{E}})\}),\underline{\mathscr{L}}) & \\
& \twoheadrightarrow_{\sigma\beta'^\circ\mu'} & \mu[\mathbf{0}]\mu(M\{\Uparrow(\overline{\mathscr{E}})\}\{\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0},\underline{\mathscr{L}}\{\uparrow\circ\uparrow\})\cdot\uparrow\}) & \\
& & & \text{by Lemma B.1.}\\
& \twoheadrightarrow_\sigma & \mu[\mathbf{0}]\mu(M\{\Uparrow(\overline{\mathscr{E}})\circ(\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0},\underline{\mathscr{L}}\{\uparrow\circ\uparrow\})\cdot\uparrow)\}) & \\
& \twoheadrightarrow_\sigma & \mu[\mathbf{0}]\mu(M\{\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0},\underline{\mathscr{L}}\{\uparrow\circ\uparrow\})\cdot(\overline{\mathscr{E}}\circ\uparrow)\}) & \\
& \twoheadrightarrow_{\rho'} & \mu(M\{\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0},\underline{\mathscr{L}}\{\uparrow\circ\uparrow\})\cdot(\overline{\mathscr{E}}\circ\uparrow)\}\{\lambda^\circ[\mathbf{1}]\mathbf{0}\cdot id\}) & \\
& \twoheadrightarrow_\sigma & \mu(M\{(\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0},\underline{\mathscr{L}}\{\uparrow\circ\uparrow\})\cdot(\overline{\mathscr{E}}\circ\uparrow))\circ(\lambda^\circ[\mathbf{1}]\mathbf{0}\cdot id)\}) & \\
& \twoheadrightarrow_{\sigma\beta'^\circ} & \mu(M\{\lambda^\circ[\mathbf{1}]\textsc{App}(\mathbf{0},\underline{\mathscr{L}}\{\uparrow\})\cdot\overline{\mathscr{E}}\}) & \text{by Lemma B.2.}\\
& = & \mu(M\{\mathrm{S}[\![\langle\mathscr{S},0\rangle]\!]\cdot\overline{\mathscr{E}}\}) & \\
& = & \mu(M\{\overline{\langle\mathscr{S},0\rangle::\mathscr{E}}\}) & \\
& = & \mathrm{D}[\![\langle M,\langle\mathscr{S},0\rangle::\mathscr{E},\sqcup,1\rangle]\!] & \\
& = & \mathrm{D}[\![\mathscr{D}']\!]. &
\end{array}$$

Move (vi): $\mathscr{D}=\langle[\mathbf{n}]M,\mathscr{E},\sqcup,i\rangle$ *and* $\mathscr{D}'=\langle\mathbf{n},\mathscr{E},\langle M,\mathscr{E}\rangle,i\rangle$. We take $i=0$, the other case being identical.

$$\begin{array}{rcl}
\mathrm{D}[\![\mathscr{D}]\!] & = & \mathrm{D}[\![\langle[\mathbf{n}]M,\mathscr{E},\sqcup,0\rangle]\!]\\
& = & \mu[\mathbf{0}]([\mathbf{n}]\mathbf{M})\{\overline{\mathscr{E}}\}\\
& \to_\sigma & \mu[\mathbf{0}](\mathbf{n}\{\overline{\mathscr{E}}\}M\{\overline{\mathscr{E}}\})\\
& = & \mu[\mathbf{0}](\mathbf{n}\{\overline{\mathscr{E}}\}\overline{\langle M,\mathscr{E}\rangle})\\
& = & \mathrm{D}[\![\langle\mathbf{n},\mathscr{E},\langle M,\mathscr{E}\rangle,0\rangle]\!]\\
& = & \mathrm{D}[\![\mathscr{D}']\!].
\end{array}$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Move (vii): $\mathscr{D} = \langle \mathbf{0}, \langle \mathscr{S}, 0\rangle :: \mathscr{E}', \langle M, \mathscr{E}\rangle, 1\rangle$ *and* $\mathscr{D}' = \langle M, \mathscr{E}, \mathscr{S}, 0\rangle$.

$$\begin{array}{rcl}
\mathrm{D}[\![\mathscr{D}]\!] & = & \mathrm{D}[\![\langle \mathbf{0}, \langle \mathscr{S}, 0\rangle :: \mathscr{E}', \langle M, \mathscr{E}\rangle, 1\rangle]\!] \\
& = & \mu\mathrm{App}(\mathbf{0}\{\overline{\langle \mathscr{S}, 0\rangle :: \mathscr{E}'}\}, \underline{\langle M, \mathscr{E}\rangle}) \\
& = & \mu(\mathbf{0}\{\mathrm{S}[\![\langle \mathscr{S}, 0\rangle]\!] \cdot \overline{\mathscr{E}'}\}(M\{\overline{\mathscr{E}}\})) \\
& \to_{\sigma} & \mu(\mathrm{S}[\![\langle \mathscr{S}, 0\rangle]\!](M\{\overline{\mathscr{E}}\})) \\
& = & \mu((\lambda^{\circ}[\mathbf{1}]\mathrm{App}(\mathbf{0}, \underline{\mathscr{L}\{\uparrow\}}))(M\{\overline{\mathscr{E}}\})) \\
& \to_{\beta'^{\circ}} & \mu(([\mathbf{1}]\mathrm{App}(\mathbf{0}, \underline{\mathscr{L}\{\uparrow\}}))\{M\{\overline{\mathscr{E}}\} \cdot id\}) \\
& \twoheadrightarrow_{\sigma} & \mu[\mathbf{0}]\mathrm{App}(M\{\overline{\mathscr{E}}\}, \underline{\mathscr{L}\{\uparrow\}}\{M\{\overline{\mathscr{E}}\} \cdot id\}) \\
& \to_{\sigma} & \mu[\mathbf{0}]\mathrm{App}(M\{\overline{\mathscr{E}}\}, \underline{\mathscr{L}\{\uparrow \circ (M\{\overline{\mathscr{E}}\} \cdot id)\}}) \\
& \twoheadrightarrow_{\sigma} & \mu[\mathbf{0}]\mathrm{App}(M\{\overline{\mathscr{E}}\}, \mathscr{L}) \\
& = & \mathrm{D}[\![\langle M, \mathscr{E}, \mathscr{S}, 0\rangle]\!].
\end{array}$$

Appendix C. The rewriting system $\sigma_{\Uparrow}$

$$\begin{array}{ll}
\text{(App)} & (MN)\{s\} \to_{\sigma} (M\{s\}N\{s\}) \\
\text{(Abs)} & (\lambda M)\{s\} \to_{\sigma} (M\{s\}N\{s\}) \\
\text{(Abs}^{\circ}\text{)} & (\lambda^{\circ} M)\{s\} \to_{\sigma} \lambda^{\circ}(M\{\Uparrow(s)\}) \\
\text{(Name)} & ([N]\, M)\{s\} \to_{\sigma} N\{s\}(M\{s\}) \\
\text{(Muabs)} & (\mu M)\{s\} \to_{\sigma} \mu(M\{\Uparrow(s)\}) \\
\text{(Clos)} & M\{s\}\{t\} \to_{\sigma} M\{s \circ t\} \\
\text{(VarShift1)} & \mathbf{n}\{\uparrow\} \to_{\sigma} \mathbf{n{+}1} \\
\text{(VarShift2)} & \mathbf{n}\{\uparrow \circ s\} \to_{\sigma} \mathbf{n{+}1}\{s\} \\
\text{(FVarCons)} & \mathbf{0}\{M \cdot s\} \to_{\sigma} M \\
\text{(FVarLift1)} & \mathbf{0}\{\Uparrow(s)\} \to_{\sigma} \mathbf{0} \\
\text{(FVarLift2)} & \mathbf{0}\{\Uparrow(s) \circ t\} \to_{\sigma} \mathbf{0}\{t\} \\
\text{(RVarCons)} & \mathbf{n{+}1}\{M \cdot s\} \to_{\sigma} \mathbf{n}\{s\} \\
\text{(RVarLift1)} & \mathbf{n{+}1}\{\Uparrow(s)\} \to_{\sigma} \mathbf{n}\{s \circ \uparrow\} \\
\text{(RVarLift2)} & \mathbf{n{+}1}\{\Uparrow(s) \circ t\} \to_{\sigma} \mathbf{n}\{s \circ (\uparrow \circ t)\} \\
\text{(AssEnv)} & (s \circ t) \circ u \to_{\sigma} s \circ (t \circ u) \\
\text{(MapEnv)} & (M \cdot s) \circ t \to_{\sigma} M\{t\} \cdot (s \circ t) \\
\text{(ShiftCons)} & \uparrow \circ (M \cdot s) \to_{\sigma} s \\
\text{(ShiftLift1)} & \uparrow \circ \Uparrow(s) \to_{\sigma} s \circ \uparrow \\
\text{(ShiftLift2)} & \uparrow \circ (\Uparrow(s) \circ t) \to_{\sigma} s \circ (\uparrow \circ t) \\
\text{(Lift1)} & \Uparrow(s) \circ \Uparrow(t) \to_{\sigma} \Uparrow(s \circ t) \\
\text{(Lift2)} & \Uparrow(s) \circ (\Uparrow(t) \circ u) \to_{\sigma} \Uparrow(s \circ t) \circ u \\
\text{(LiftEnv)} & \Uparrow(s) \circ (M \cdot t) \to_{\sigma} M \cdot (s \circ t) \\
\text{(IdL)} & id \circ s \to_{\sigma} s \\
\text{(IdR)} & s \circ id \to_{\sigma} s \\
\text{(LiftId)} & \Uparrow(id) \to_{\sigma} id \\
\text{(Id)} & M\{id\} \to_{\sigma} M
\end{array}$$

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

References

Audebaud, P. (1994) Explicit substitutions for the lambda-mu calculus. Research Report 94-26, Ecole Normale Supérieure de Lyon.

Barendregt, H. P. (1984) *The lambda calculus, its syntax and semantics*, North-Holland (revised edition).

Curien, P.-L., Hardin, T. and Lévy, J.-J. (1996) Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM* **43** (2) 362–397.

de Bruijn, N. G. (1972) Lambda calculus notations with nameless dummies, a tool for automatic formula manipulation, with an application to the Church–Rosser theorem. *Indagationes Mathematicae* **34** 381–392.

de Groote, P. (1994a) A CPS-translation of the $\lambda\mu$-calculus. In: Tison, S. (ed.) Proceedings of the 19th International Colloquium on Trees in Algebra and Programming (CAAP'94). *Springer-Verlag Lecture Notes in Computer Science* **787** 85–99.

de Groote, P. (1994b) On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In: Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning–LPAR'94. *Springer-Verlag Lecture Notes in Computer Science* **822** 31–43.

Felleisen, M., Friedman, D. P., Kohlbecker, E. and Duba, B. (1987) A syntactic theory of sequential control. *Theoretical Computer Science* **52** 205–237.

Felleisen, M. and Hieb, R. (1992) The revised report on the syntactic theory of sequential control and state. *Theoretical Computer Science* **102** 235–271.

Gentzen, G. (1955) *Recherches sur la déduction logique (Untersuchungen über das logische schliessen)*, Presses Universitaires de France. (Traduction et commentaire par R. Feys et J. Ladrière.)

Griffin, T. G. (1990) A formulae-as-types notion of control. In: *Conference record of the seventeenth annual ACM symposium on Principles of Programming Languages* 47–58.

Klop, J. W., Van Oostrom, V. and Van Raamsdonk, F. (1993) Combinatory reduction systems: introduction and survey. *Theoretical Computer Science* **121** (1-2) 279–308.

Krivine, J.-L. (1994) Classical logic, storage operators and second order λ-calculus. *Annals of Pure and Applied Logic* **68** 53–78.

Landin, P. J. (1964) The mechanical evaluation of expressions. *Computer Journal* **6** 308–320.

Mellies, P.-A. (1995) Typed λ-calculi with explicit substitutions may not terminate. In: Dezani, M. and Plotkin, G. (eds.) Second International Conference on Typed Lambda Calculi and Applications, TLCA'95. *Springer-Verlag Lecture Notes in Computer Science* **902** 328–334.

Parigot, M. (1992) $\lambda\mu$-Calculus: an algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.) Proceedings of the International Conference on Logic Programming and Automated Reasoning. *Springer-Verlag Lecture Notes in in Artificial Intelligence* **624** 190–201.

Parigot, M. (1993) Strong normalization for second order classical natural deduction. In: *Proceedings of the eighth annual IEEE symposium on logic in computer science* 39–46.

Prawitz, D. (1965) *Natural Deduction, A Proof-Theoretical Study*, Almqvist & Wiksell, Stockholm.

Py, W. (1988) *Confluence en $\lambda\mu$-calcul*, Thèse de doctorat, Université de Savoie.

Rehof, N. J. and Sørensen, M. H. (1994) The λ_Δ-calculus. In: Hagiya, M. and Mitchell, J. C. (eds.)

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press

Proceedings of the International Symposium on Theoretical Aspects of Computer Software – TACS'94. *Springer-Verlag Lecture Notes in Computer Science* **789** 516–542.

Streicher, T. and Reus, B. (to appear) Classical logic, continuations and abstract machines. To appear in the *Journal of Functional Programming*.

https://doi.org/10.1017/S0960129598002667 Published online by Cambridge University Press