Confluence Properties of Extensional and Non-Extensional λ-Calculi with Explicit Substitutions (Extended Abstract)

Delia Kesner

CNRS and LRI, Bât 490, Université Paris-Sud - 91405 Orsay Cedex, France.
e-mail:Delia.Kesner@lri.fr

Abstract. This paper studies confluence properties of extensional and non-extensional λ-calculi with explicit substitutions, where extensionality is interpreted by η-expansion. For that, we propose a general scheme for explicit substitutions which describes those abstract properties that are sufficient to guarantee confluence. Our general scheme makes it possible to treat at the same time many well-known calculi such as λ_σ, $\lambda_{\sigma_\Uparrow}$ and λ_υ, or some other new calculi that we propose in this paper. We also show for those calculi not fitting in the general scheme that can be translated to another one fitting the scheme, such as λ_s, how to reason about confluence properties of their extensional and non-extensional versions.

1 Introduction

The λ-calculus is a convenient framework to study functional programming, where the evaluation process is modeled by β-reduction. The main mechanism used to perform β-reduction is *substitution*, which is a meta-level operation in classical λ-calculus as in λ-calculus à la de Bruijn whereas it is incorporated into the language when using the formalism of λ-calculi with explicit substitutions.

When working with functions, an *extensional* equality is also necessary to reason about programs. Extensionality means that whenever two programs yield the same result for every possible argument, they are equal. The classical extensional axiom in the theory of λ-calculus is the η-equality, written as $\lambda x.Mx =_\eta M$, if x is not free in M.

This equality can be operationally interpreted from left to right, as the classical η-contraction rule, or in the other way round [36, 33, 23], as an $\widehat{\eta}$-expansion. The traditional contractive way breaks confluence in many λ-calculi [6, 12], while the $\widehat{\eta}$-expansion can be combined with many other higher-order reduction rules [11, 2, 15, 24, 12, 13], by preserving confluence and strong normalization.

The main goal of this paper is to study *confluence* properties of those extensional and non-extensional λ-calculi with explicit substitutions which are designed to implement the λ-calculus à la de Bruijn. To guarantee the correctness of the extensional calculi studied in this paper, we first show that the combination of β-reduction and $\widehat{\eta}$-expansion yields a confluent reduction relation when working with de Bruijn's notation. The rest of the paper is devoted to λ-calculi with explicit substitutions, for which we keep the classical definition of $Beta$-reduction associated to β-reduction, and define the $\widehat{Eta}$-expansion rule associated to the $\widehat{\eta}$-expansion one, this rule being more natural than all the contractive interpretations given in the literature [19, 20, 37, 5]. We study confluence properties of the reduction relation associated to the $Beta$-reduction alone as well as those associated to the combination of $Beta$-reduction and $\widehat{Eta}$-expansion.

Since we are interested in many different λ-calculi with explicit substitutions, which are either well-known in the literature (such as λ_σ [1], $\lambda_{\sigma_\Uparrow}$ [22], λ_υ [29]) or constitute new proposals given in this paper, we develop an "abstract" proof that allows to handle in a simple and general way all these calculi, by a simple instantiation of the proof on each case-study. This is done by extracting the necessary conditions needed for the proofs, then formulating a notion of "general scheme" for λ-calculi with explicit substitution, and finally verifying that every calculus in the paper (well-known or new) matches this general scheme. For those calculi not fitting in the general scheme, such as λ_s [27], that can be translated to another one fitting the scheme, we also give a proof technique allowing to derive the confluence property of the combination of $Beta$-reduction with $\widehat{Eta}$-expansion.

The paper is organized as follows. In Section 2 we recall the λ-calculus à la de Bruijn and we define the associated $\widehat{\eta}$-expansion rule. We prove that the combination of β-reduction and $\widehat{\eta}$-expansion yields a confluent reduction relation in the *typed* λ-calculus à la de Bruijn.

In Section 3 we introduce a general scheme for λ-calculi with explicit substitutions, which is defined to have a concrete structure of terms, but it does not specify the *full* behavior of *all* the substitution constructors. We define an *extensional* version for this general scheme using the notion of $\widehat{Eta}$-expansion associated to that of $\widehat{\eta}$-expansion in classical λ-calculus, which is shown to be correct with respect to the corresponding equational axiomatization. We develop abstract proofs of confluence for the extensional and non-extensional versions of the calculi fitting the general scheme.

In Section 4 we apply the method to some well-known and new λ-calculi with explicit substitutions by just verifying that these calculi fit in the general scheme defined in section 3. Finally, in section 5 we show that $Beta \cup \widehat{Eta}$ is also confluent in λ_s, and we outline a general method to do the same for calculi not fitting the general scheme that can be translated to our scheme. We conclude in section 6.

For details on the technical proofs we refer the reader to [25].

2 The λ-calculus à la de Bruijn

When using the β-reduction rule to model evaluation in classical λ-calculus, bound variables need to be renamed in order to ensure the correctness of the substitution operation. Thus substitution is not really defined on terms but on α-equivalence classes.

A classical way to avoid α-conversion problems is to use the de Bruijn's notation [9, 10] for λ-terms, where names of variables are replaced by natural numbers. Hence, the set of first-order representations of λ-terms, called here *pure terms* or simply *terms*, is defined by the following grammar:

$$Naturals\ n ::= 1 \mid n+1 \qquad Pure\ Terms\ a ::= \underline{n} \mid aa \mid \lambda a$$

A given occurrence u of a bound variable, say x, is replaced by the number of symbols λ whose occurrences are between the binder of this x and u. For example, $\lambda x.\lambda y.x(\lambda z.zx)y$ is written as $\lambda(\lambda(\underline{2}(\lambda(\underline{1}\ \underline{3}))\underline{1}))$. Free variables are represented w.r.t. a given context of variables in such a way that an occurrence of a variable $\underline{n}$ represents the $(n-m)$-th free variables of the context when $n > m$ and there are m symbols λ surrounding this occurrence n. For example, the term $\lambda x.(y\ z)$ is written as $\lambda(\underline{2}\ \underline{3})$ with respect to the context y, z.

To avoid confusion we will note classical λ-terms with capital letters M, N, ... and de Bruijn's terms with lower case letters a, b, ...

Definition 1 Contexts and applied terms. A *context* is a term containing a special symbol $\Box$ denoting a hole. We shall note a context as $C[\,]$ and the textual replacement of the hole by a term a as $C[a]$. Thus, $C[a]$ denotes a term. The subterm a is defined to be *applied* in any term having the form $C[(a\ b)]$.

Definition 2 β-reduction. β-reduction is defined to be the reflexive-transitive closure of the reduction step $(\lambda a)b \longrightarrow_\beta a\{1 \leftarrow b\}$.

The meta-level operation between brackets behaves as expected: for example, $(\lambda(\lambda((\underline{5}\ \underline{2})\ \underline{1})))\lambda(\underline{3}\ \underline{1})$ β-reduces to the term $\lambda((\underline{5}\ \underline{2})\ \underline{1})\{1 \leftarrow \lambda(\underline{3}\ \underline{1})\}$ which is equal to $\lambda((\underline{4}\ (\lambda(\underline{4}\ \underline{1})))\ \underline{1})$.

Type information can be added to this formalism to model the *typed* λ-calculus. It will be kept in an *environment*, defined as a list of types $A_1 \ldots A_n.nil$. The *typing rules* are the following:

$$\frac{}{A, \Gamma \vdash \underline{1} : A} \qquad \frac{\Gamma \vdash \underline{n} : B}{A, \Gamma \vdash \underline{n+1} : B} \qquad \frac{A, \Gamma \vdash b : B}{\Gamma \vdash \lambda A.b : A \to B} \qquad \frac{\Gamma \vdash a : A \to B \qquad \Gamma \vdash b : A}{\Gamma \vdash ab : B}$$

We say that a *is of type* A, written $a : A$, if there exists Γ such that $\Gamma \vdash a : A$ is derivable from the typing rules above.

2.1 Handling η-reduction

By far, the best known extensional equality in λ-calculus is the η-axiom, written as $\lambda x : A.Mx =_\eta M$, provided x is not free in M. This equality captures the fact that both functions, $\lambda x : A.Mx$ and M, return the same result when applied to the same argument N in any context $C[\ ]$, *i.e.*, $C[(\lambda x : A.Mx)N] =_\beta C[MN]$.

When one wants to turn the η-equality into a rule there are two different choices: either from left to right as a *contraction*, called η-contraction, or in the other way round as an *expansion*, called η-expansion or $\widehat{\eta}$-expansion, depending on whether some other restrictions are imposed to its application.

The η-axiom has traditionally been turned into a contraction. Such an interpretation is well behaved in the simply typed λ-calculus as it preserves confluence [35], but it is not in many other λ-calculi [6, 12].

Fortunately, expansions can be combined with many other higher-order reduction rules [11, 2, 15, 24, 12, 13]; all these combinations preserve confluence and strong normalization. For that, application of the η-expansion has to be restricted by some conditions in order to guarantee the strong normalization property, and thus the restricted reduction relation, called $\widehat{\eta}$-expansion, is no more a congruence on terms.

The η-equality in the calculus à la de Bruijn is intended to express the fact that for any context $C[\ ]$, the de Bruijn's representations of two terms $C[\lambda x.Mx]$ and $C[M]$ are equal. In terms of representation of variables, this means that all the free variables in M (that may be bound with respect to $C[\ ]$) traverse one more λ when M is replaced by $\lambda x.Mx$, suggesting in this way that a term a is not η-equivalent to $\lambda(a\ \underline{1})$ because of the adjustments to be made on the free variables of a. For example, the term $\lambda(\underline{2}\ \underline{1})$, placed in a context $\lambda[\ ]$ is η-equivalent to the term $\lambda\underline{1}$ and not to $\lambda\underline{2}$. This is not surprising since $\lambda(\lambda(\underline{2}\ \underline{1}))$ is the de Bruijn's representation of $\lambda y.\lambda x.yx$, which is η-equivalent to $\lambda y.y$, represented by $\lambda\underline{1}$. This condition can be expressed by first defining a^+ as the term obtained by lifting any free variable of a by 1, and then by defining η-equality as follows:

Definition 3 η-equality. Let a and b be pure terms. Then $\lambda(a\ \underline{1}) =_\eta b$, if $a = b^+$.

Now, the *η-contraction* is obtained by orienting the previous equality from left to right, while the *η-expansion* is obtained by orienting the equation into the other way round. Analogously to what is done in classical λ-calculus to guarantee strong normalization of extensional expansions [11], we can now define a conditional $\widehat{\eta}$-expansion as follows:

Definition 4 The conditional $\widehat{\eta}$-expansion. The conditional $\widehat{\eta}$-expansion is defined to be the reflexive-transitive closure of the following reduction step:

$$C[a] \longrightarrow_{\widehat{\eta}} C[\lambda A.(a^+\ \underline{1})] \qquad \text{if } \begin{cases} a \text{ is of type } A \to B \\ a \text{ is not a } \lambda\text{-abstraction} \\ a \text{ is not applied in } C[a] \end{cases}$$

Exactly as in the case of the simple typed λ-calculus, the rule enjoys the subject reduction property and is adequate, *i.e.*, any η-equality can be generated by the reflexive, symmetric and transitive closure of β-reduction plus the conditional $\widehat{\eta}$-expansion [11, 25]. From now on, the conditional $\widehat{\eta}$-expansion will be just called *$\widehat{\eta}$-expansion*.

2.2 Confluence of $\beta\widehat{\eta}$-reduction

We show now that the combination of β-reduction and $\widehat{\eta}$-expansion yields a confluent relation in the typed λ-calculus à la de Bruijn. For that, we will use the following abstract theorem proposed in [14] which allows to derive the confluence property for the combination of two different reduction relation R_1, R_2.

Proposition 5 Sufficient conditions for confluence. *Let R_1 and R_2 be two confluent reduction relations such that R_1 is strongly normalizing and, for all terms a, b and c such that $a \longrightarrow_{R_1} b$ and $a \longrightarrow_{R_2} c$, there is a term d such that $c \longrightarrow^{+}{}_{R_1} d$ and $b \longrightarrow^{*}{}_{R_2} d$ (local commutation property). Then $R_1 \cup R_2$ is confluent.*

Now, to prove that $\beta\widehat{\eta}$ is confluent, it is sufficient to take $R_1 = \beta$ and $R_2 = \widehat{\eta}$ in proposition 5, then to verify the hypothesis of the statement. As β-reduction is confluent and strongly normalizing in typed λ-calculus à la de Bruijn [3, 17] and $\widehat{\eta}$-expansion is confluent [11], we are just left to show the local commutation property, which can be proved by inspecting all the critical pairs between β and $\widehat{\eta}$. The proof proceeds exactly as in [11], but we refer the interested reader to [25] for a proof with de Bruijn's indices. Now, we are ready to conclude

Theorem 6. $\longrightarrow^{*}{}_{\beta\widehat{\eta}}$ *is confluent in the typed λ-calculus à la de Bruijn.*

3 A general scheme for explicit substitutions

When substitutions are specified by a meta-level operation, as in classical λ-calculus, an external mechanism takes place each time a β-redex needs to be reduced. The approach is very different in λ-calculi with explicit substitutions where the manipulation of substitutions is incorporated in an explicitly way to the calculus via a set of rewrite rules.

In this section we introduce a *general scheme for λ-calculi with explicit substitutions* which is defined to have a concrete structure of terms, but does not specify the *full* behavior of *all* the substitution constructors. The intended meaning of this general scheme is to extract from the various λ-calculi with explicit substitutions existing in the literature a uniform description of their specifications and a set of common features needed to reason about confluence. We also define the notion of $\widehat{Eta}$-expansion associated to that of $\widehat{\eta}$-expansion in classical λ-calculus, and show that the definition is correct with respect to the corresponding equational axiomatization. The "abstract" specification given in this section will be sufficient to show that both $Beta$ and $Beta \cup \widehat{Eta}$ are confluent reduction relations in many λ-calculi with explicit substitutions such as λ_σ [1], $\lambda_{\sigma_{\Uparrow}}$ [22], and λ_υ [29]. Moreover, the general scheme proposed here can also be used to show confluence properties of two *new* λ-calculi with explicit substitutions fitting this abstract specification: the λ_d and the λ_{dn} calculi, both described in section 4.

3.1 Syntax

Definition 7 Substitution signatures. Let us consider two distinguished symbols $\mathcal{T}$ (for terms) and $\mathcal{S}$ (for substitutions). A *substitution declaration* is a (possibly empty) word over the alphabet $\{\mathcal{T}, \mathcal{S}\}$. A *substitution signature* is defined to be a signature Σ (a set of symbols) such that every symbol in Σ is equipped with an arity n and a declaration of length n. We usually write $\xi : \langle n, \sigma_1 \ldots \sigma_n \rangle$ if ξ has arity n and declaration $\sigma_1 \ldots \sigma_n$.

Definition 8 Substitution language. The set of objects over a substitution signature Σ is defined to be the union of objects of sort $\mathcal{T}$ and $\mathcal{S}$. They are constructed in the following way :

- If n is a natural number, then $\underline{n}$, called a *variable*, is of sort $\mathcal{T}$.
- If a and b are of sort $\mathcal{T}$, then $(a\ b)$, called an *application*, is of sort $\mathcal{T}$.
- If a is of sort $\mathcal{T}$, then $\lambda(a)$, called an *abstraction*, is of sort $\mathcal{T}$.
- If a is of sort $\mathcal{T}$ and s is of sort $\mathcal{S}$, then $a[s]$, called a *closure*, is of sort $\mathcal{T}$.
- If every f_i is of sort σ_i, and $\xi : \langle n, \sigma_1 \ldots \sigma_n \rangle$, then $\xi(f_1, \ldots, f_n)$ is of sort $\mathcal{S}$.

For each substitution language W, the set of objects of sort $\mathcal{T}$ (resp. $\mathcal{S}$) is denoted by Λ^t_W (resp. Λ^s_W), and called *the set of terms (resp. substitutions) of W*. We write Λ_W for $\Lambda^t_W \cup \Lambda^s_W$.

For example, the signature Σ associated to the substitution language of the υ-calculus [29] is the set $\{\#, \Uparrow, \uparrow\}$, where $\# : \langle 1, \mathcal{T}\rangle$, $\Uparrow : \langle 1, \mathcal{S}\rangle$ and $\uparrow : \langle 0, \epsilon\rangle$. The signature Σ associated to the substitution language of the σ-calculus [1] is the set $\{., \circ, id, \uparrow\}$, where $. : \langle 2, \mathcal{TS}\rangle$, $\circ : \langle 2, \mathcal{SS}\rangle$, $id : \langle 0, \epsilon\rangle$ and $\uparrow : \langle 0, \epsilon\rangle$.

We now introduce the rewriting rules of the general scheme for λ-calculi with explicit substitutions. We start with the $Beta$ rule that is the main mechanism in these λ-calculi: it is used to initiate the computation and to explicitly introduce substitutions in the terms to be evaluated. The left-hand side of the $Beta$ rule corresponds to a β-redex $(\lambda a)b$ while the right-hand side must be an expression used to denote the term a where the index $\underline{1}$ is replaced by the term b, the index $\underline{2}$ by $\underline{1}$, the index $\underline{3}$ by $\underline{2}$, etc. This is done in different ways according to the substitution language of the corresponding calculus, but we can, in general, write this expression as $a[cons^W(b)]$, where $cons^W$ is a function from terms to substitutions that must be defined for each λ-calculus W. The $Beta$ rule of the general scheme can then be expressed as [1]:

$$(Beta) \qquad (\lambda a)b \longrightarrow a[cons^W(b)]$$

For example, the $cons^\upsilon$ function corresponding to the λ_υ-calculus is $cons^\upsilon(b) = \#(b)$, while the $cons^\sigma$ function corresponding to the λ_σ-calculus is $cons^\sigma(b) = b.id$.

The second rule of the general scheme, called the *App* rule, is used to distribute a substitution to both subterms of an application term:

$$(App) \qquad (a\ b)[s] \longrightarrow (a[s]\ b[s])$$

The third rule, called $Lambda$, applies when a substitution goes under a λ constructor, so that the left-hand side is a redex $(\lambda a)[s]$, while the right-hand side is written as $\lambda(a[lift^W(s)])$, where $lift^W$ is a function from substitutions to substitutions and must be defined for each λ-calculus W [2].

$$(Lambda) \qquad (\lambda a)[s] \longrightarrow \lambda(a[lift^W(s)])$$

As an example, $lift^\upsilon(s) = \Uparrow(s)$ and $lift^\sigma(s) = \underline{1}.(s \circ \uparrow)$.

Definition 9 Substitution calculus. A rewrite rule $l \rightarrow r$ is *sort-preserving* if r is of sort $\mathcal{K}$ when l is of sort $\mathcal{K}$. A *Substitution Calculus*[3] (SC) consists of a signature containing the substitution constant $\uparrow : \langle 0, \epsilon\rangle$, which is called a *shift-constant* and will be used to denote extensional equalities, a lift-function, a cons-function, and a set of sort-preserving rewriting rules containing *App* and $Lambda$ (but not $Beta$).

Notation 10 We write simply W for the set of rewriting rules of the SC W, and λ_W for the set $W \cup Beta$. The notations $\longrightarrow_{\mathcal{R}}$ and $=_{\mathcal{R}}$ are respectively used to denote $\mathcal{R}$-reduction and $\mathcal{R}$-equality. For every function h, we define h_i (for $i \geq 0$) as $h_0(s) = s$ and $h_{n+1}(s) = h(h_n(s))$.

The rewriting rules $Beta$, *App* and $Lambda$ are necessary to describe some relations between terms and substitutions, but they are not sufficient to express all of them. In particular, we want to prove confluence of some reduction relations associated to all the λ-calculi satisfying our general scheme, so we have to impose some additional conditions on the set of rewriting rules of the SC. These conditions are mainly inspired from the various proofs of confluence existing in the literature for the relations $\longrightarrow_{\lambda_W}$.

Definition 11 Basic conditions. A substitution calculus W is said to be *basic* (written BSC) if it satisfies the following conditions:

[1] Since the right hand side of the $Beta$ rule depends on the substitution calculus W, we should denote this rule as $Beta_W$, but we prefer to omit the index W to make easier the notation.
[2] The same remark done for the $Beta$ rule applies here.
[3] Another notion of substitution calculus can be found in [38].

1. W is strongly normalizing[4].
2. W is confluent.
3. W-normal forms of terms are pure terms.
4. $W(a\ b) = (W(a)\ W(b))$ and $W(\lambda a) = \lambda W(a)$.
5. For any substitution s in W, $\underline{1}[lift^W(s)] =_W \underline{1}$ and for any $m \geq 1$, $\underline{m+1}[lift^W(s)] =_W \underline{m}[s][\uparrow]$.
6. For any term k in W, $\underline{1}[cons^W(k)] =_W k$ and for any $m \geq 1$, $\underline{m+1}[cons^W(k)] =_W \underline{m}$
7. For any variable $\underline{m}$ in W we have $\underline{m}[\uparrow] =_W \underline{m+1}$.

From conditions 1 and 2 in definition 11 we deduce the existence and uniqueness of W-normal forms, denoted by $W(\ )$ [5]. This, together with condition 3 guarantees that for every object a in Λ_W and every pure term b such that $W(a) = b$, then $a \longrightarrow^*_W b$. This property will be used in several lemmas of the paper.

The type system associated to each SC depends on the substitution function symbols of its signature. However, there is a set of *common* typing rules in all these systems, which is:

$$\frac{}{A, \Gamma \vdash \underline{1} : A} \qquad \frac{\Gamma \vdash \underline{n} : B}{A, \Gamma \vdash \underline{n+1} : B} \qquad \frac{\Gamma \vdash a : A \to B \qquad \Gamma \vdash b : A}{\Gamma \vdash ab : B}$$

$$\frac{}{A, \Gamma \vdash \uparrow\ \triangleright \Gamma} \qquad \frac{\Gamma \vdash a : A \qquad \Delta \vdash s \triangleright \Gamma}{\Delta \vdash a[s] : A} \qquad \frac{A, \Gamma \vdash b : B}{\Gamma \vdash \lambda A.b : A \to B}$$

3.2 Handling Eta-reduction

The standard equation corresponding to η-equality in classical λ-calculus is usually written in the formalism of λ-calculi with explicit substitutions as the axiom:

$$(Eta) \qquad \lambda(a[\uparrow]\ \underline{1}) = a$$

where $\uparrow$ is the substitution constant which was assumed to be in the signature of every SC.

The relation between $\widehat{\eta}$ and $\widehat{Eta}$ can be stated using the following property:

Lemma 12. *Let W be a BSC and let a be a pure term. Then $W(a[\uparrow])$ is the term a^+.*

When turning the Eta-axiom into a reduction rule some problems arise. If we orient this equation from left to right as the rewriting rule $\lambda(a[\uparrow]\ \underline{1}) \longrightarrow a$, an infinite set of critical pairs is generated. As a consequence, the rule is often expressed as the *conditional* rule [19, 20, 37]:

$$(Eta_1) \qquad \lambda(a\ \underline{1}) \longrightarrow b \qquad \text{if } a =_W b[\uparrow]$$

However, the condition $a =_W b[\uparrow]$ depends on the particular definition of W, and may be difficult (or expensive) to be verified each time that an Eta_1-reduction is performed.

There is another *unconditional* rewriting rule associated to the Eta-axiom, proposed by Briaud [5], expressed in our syntax as follows:

$$(Eta_2) \qquad \lambda(a\ \underline{1}) \longrightarrow a[cons^W(\perp)]$$

[4] The well-known substitution calculi appearing in the literature are all strongly normalizing, and moreover, standard proof techniques like polynomial orders are usually sufficient to show this property. However, when adding the $Beta$ rule, strong normalization is not evident, and in may cases it does not hold, even in a typed framework [32].

[5] Even if existence of unique normal forms can also be deduced from *weak normalization* and confluence, we prefer to require a substitution calculus to have the strong normalization property to guarantee that any strategy terminates when computing pure terms.

where $\perp$ is a *new* constant added to the set of terms. This rule can be expressed in classical λ-calculus as:

$$(Eta_3) \qquad \lambda x.(M\ x) \longrightarrow\ M\{x \leftarrow \perp\}$$

The η-contraction rule in classical λ-calculus coincides with the Eta_3-rule only if the variable x is not free in M. Similarly, the Eta_2-rule coincides with the conditional Eta_1-rule if the variable represented by $\underline{1}$ is not free in the term a. Otherwise, the Eta_1-rule is not allowed (because the condition is not verified), but Eta_2 is. The intended meaning of the constant $\perp$ is to denote something that is not a term, *i.e.*, a meaningless term. For this reason, every normal form involving $\perp$ has to be discharged, and this requires some kind of backtracking in the computation process, which is, as we know, an expensive operation.

The Eta_2-rule can also be interpreted as the following *conditional* rule:

$$(Eta_2') \qquad \lambda(a\ \underline{1}) \longrightarrow\ a[cons^W(\perp)], \qquad \text{if } \perp \notin W(a[cons^W(\perp)])$$

This leads to perform a verification before applying the rule, since one has to verify that the symbol $\perp$ does not appear in the W-normal of the term $a[cons^W(\perp)]$. Therefore, this solution is still expensive by means of computation steps and, exactly as for the Eta_1-rule, depends on the particular definition of the substitution calculus W.

Fortunately, there is still another reading of the Eta-axiom that avoids these problems, it corresponds to the *unrestricted* Eta-expansion:

$$(Eta) \qquad a \longrightarrow\ \lambda(a[\uparrow]\ \underline{1})$$

As explained in section 2.1, the expansive interpretation of the η-axiom has been very successful in these last years, mainly because it has been shown to be compatible with many other higher-order rules. This suggests that it would also be a good solution to define *extensional* versions of λ-calculi with explicit substitutions containing the *expansive* interpretations of the Eta-axiom.

As the unrestricted Eta-expansion is not strongly normalizing, one has to restrict its application, exactly as in the case of the $\widehat{\eta}$-expansion rule, by imposing some conditions to guarantee termination.

Definition 13 Applied-Terms and Lambda-Abstractions. A term a is a *Lambda-Abstraction* if either $a = \lambda b$, or $a = b[s]$ and b is a Lambda-Abstraction. Similarly, let $C[\ ]$ be any context and s any substitution. A subterm a is said to be an *Applied-Term* in $C[a]$ if it is applied in $C[a]$; while a is said to be an *Applied-Term* in $C[a[s]]$ if it is an Applied-Term in $C[a]$.

For example, $(\lambda\underline{1})$ and $(\lambda\underline{1})[\uparrow][\uparrow]$ are Lambda-Abstractions, and $\underline{1}$ is an Applied-Term in $(\underline{1}\ \underline{2})$ and $(\underline{1}[\uparrow]\ \underline{2})$.

We now present the conditional $\widehat{Eta}$-expansion rule, remark that the relation generated by this rule is no longer a congruence, as in the case of λ-calculus.

Definition 14. The conditional $\widehat{Eta}$-expansion rule is defined to be the reflexive-transitive closure of the following reduction step:

$$C[a] \longrightarrow_{\widehat{Eta}}\ C[\lambda A.(a[\uparrow]\ \underline{1})],\ \text{if} \begin{cases} a \text{ is of type } A \to B \\ a \text{ is not a Lambda-Abstraction} \\ a \text{ is not an Applied-Term in } C[a] \end{cases}$$

The *extensional* version of W is the calculus containing the rules W, $Beta$ and $\widehat{Eta}$. We note $\longrightarrow_{\lambda_W \cup \widehat{Eta}}$ the reduction relation associated to the *extensional* substitution calculus W.

Lemma 15 Subject reduction for $\widehat{Eta}$. *If $\Gamma \vdash a : A$ and $a \longrightarrow_{\widehat{Eta}} b$, then $\Gamma \vdash b : A$.*

Notice that if a subterm a is not defined to be an Applied-Term in $C[(a[s]\ b)]$, then the $\widehat{Eta}$-expansion rule will not be terminating as the following infinite reduction sequence shows:

$$\underline{a} \longrightarrow_{\widehat{Eta}} \lambda(\underline{a}[\uparrow]\ \underline{1}) \longrightarrow_{\widehat{Eta}} \lambda((\lambda(a[\uparrow]\ \underline{1}))[\uparrow]\ \underline{1}) \dots$$

Lemma 16 Strong normalization of $\widehat{Eta}$. *The reduction $\longrightarrow_{\widehat{Eta}}$ is strongly normalizing.*

Proof. Using a decreasing measure as defined in [34].

It is also worth noticing that even if a term a is not a Lambda-Abstraction according to definition 13, its W-normal form may be one: for example $\underline{1}[cons^W(\lambda\underline{1})]$ is not a Lambda-Abstraction but $W(\underline{1}[cons^W(\lambda\underline{1})]) = \lambda\underline{1}$ is. The same happens with the notion of Applied-Terms. This remark suggests that one could also use a different notion of restriction in definition 14, saying that a term a is a Lambda-Abstraction if $a =_W \lambda b$. This alternative notion of restricted expansion coincides exactly with that of simply typed λ-calculus, but introduces at the same time a condition that depends on the particular definition of W, exactly as in the case of the Eta_1-rule previously mentioned. Even if our approach introduces a conditional rule in definition 14, the restriction does not depend at all on the particular theory generated by the substitution calculus. On the other hand, there is no backtracking when applying this rule as in the case of the Eta_3-contraction rule.

The $\widehat{Eta}$-expansion generates the same equational theory as Eta_1, even if it seems more restrictive because of the syntactic conditions to be verified.

Theorem 17. *Let W be a BSC. Then any Eta-equality $a =_{Eta} b$ can be generated by the reflexive, symmetric and transitive closure of $\longrightarrow_{W \cup Beta \cup \widehat{Eta}}$.*

3.3 Confluence properties

We show in this section that extensional and non-extensional basic substitution calculi are confluent. For the non-extensional case, we use the *interpretation method* [18], that is the standard technique used to prove confluence of λ-calculi with explicit substitutions. When dealing with extensional rules, the interpretation method has to be slighty modified, resulting in the *generalized interpretation method*, to prove that $Beta$-reduction together with $\widehat{Eta}$-expansion preserve confluence of any SC. It is also worth noticing that standard techniques, like the Tait and Martin-Löf's parallel moves lemma, do not work when dealing with restricted expansions, this is the reason we prefer to use directly the interpretation method, even for the non-extensional versions of our λ-calculi. According to the philosophy of this paper, we extract the necessary conditions to be added to the basic conditions stated in definition 11 in order to verify the properties required by the interpretation methods. These conditions are very simple to understand and easy to prove as they just concern the interaction between variables and substitutions of the substitution language.

Definition 18 General scheme. We say that a BSC W obeys the *general scheme* iff for every variable $\underline{m} \in \Lambda_W^t$ and every function symbol $\xi \in \Sigma$ of arity q one of the two following cases holds:

- There is a variable $\underline{n}$, indices $1 \leq i_1, \dots, i_p \leq q$ $(p \geq 0)$ and substitutions $u_1, \dots, u_k$ $(k \geq 0)$ such that for all $s_1 \dots s_q$ we have $\underline{m}[\xi(s_1, \dots, s_q)] =_W \underline{n}[s_{i_1}] \dots [s_{i_p}][u_1] \dots [u_k]$
- There is an index i $(1 \leq i \leq q)$ such that for all $s_1 \dots s_q$ we have $\underline{m}[\xi(s_1, \dots, s_q)] =_W s_i$

We assume these equations to be well-typed, in the sense that whenever the first case holds, then $s_{i_1}, \dots, s_{i_p}$ are substitutions, and whenever the second case holds, then s_i is a term.

This condition says that the behavior of any substitution $s = \xi(s_1, \dots, s_q)$ with respect to a variable $\underline{m}$ is only determined by the constructor ξ and not at all by the other components $s_1 \dots s_q$. In both cases, we use the equality $=_W$ instead of the rewrite relation $\longrightarrow_W$ because sometimes the property is verified by a rewrite step $\longrightarrow_W$ and others by ${}_W\!\longleftarrow$: indeed, in the σ-calculus, we have $\underline{m}[s \circ t]\ {}_{Clos}\!\longleftarrow \underline{m}[s][t]$ and not $\underline{m}[s \circ t] \longrightarrow_{Clos} \underline{m}[s][t]$, while, for example in the λ_v-calculus, we have $\underline{m}[\uparrow] \longrightarrow_{VarShift} \underline{m+1}$. The general scheme also makes easier to reason by induction on the structure of an object.

We are now ready to apply the interpretation method which allows to reason about confluence in a quite modular way, by splitting an entire reduction relation R into two different and disjoint relations R_1 and R_2.

Lemma 19 Interpretation method [18]. *Let $R = R_1 \cup R_2$, where R_1 is a confluent and strongly normalizing reduction relation and R_2 an arbitrary reduction relation. If there exists a reduction relation R' on the set of R_1-normal forms such that*

$$R' \subseteq R^* \text{ and } (a \longrightarrow_{R_2} b \text{ implies } R_1(a) \longrightarrow_{R'} R_1(b))$$

then if R' is confluent, R is also confluent.

When showing the confluence property for any relation $\longrightarrow_{\lambda_W}$, one usually takes R' as $\longrightarrow^*_\beta$, R_1 as $\longrightarrow_W$ and R_2 as $\longrightarrow_{Beta}$. As every BSC W is confluent and strongly normalizing, and β-reduction is only defined on W-normal forms (that are pure terms), one is left to show that

$$\longrightarrow^*_\beta \subseteq \longrightarrow^*_{\lambda_W} \text{ and } (a \longrightarrow_{Beta} b \text{ implies } W(a) \longrightarrow^*_\beta W(b))$$

The first property is quite simple to prove whereas the second one is the essential part of the general abstract proof that distills the different techniques appearing in the literature. It is also worth noticing that the property concerns only terms because β-reduction is not defined on substitutions. The key case to prove that $a \longrightarrow_{Beta} b$ implies $W(a) \longrightarrow^*_\beta W(b)$ is when a and b are closures $c[s]$ and $c[s']$ such that $s \longrightarrow_{Beta} s'$. In all the proofs appearing in the litterature this is done by inspecting all the possible cases, whereas in our case we use a similar statement for substitutions (which results in the second point of the following lemma), and also the general scheme condition (which allows to reason by induction on the structure of objects).

Lemma 20. *Let W be a BSC having the general scheme and let $p \in \Lambda_W$. If $p \longrightarrow_{Beta} p'$, then*
1) if p is a term, then $W(p) \longrightarrow^*_\beta W(p')$
2) for every pure term d such that $d[p]$ is a term, and every $n \geq 0$, then
$W(d[lift^W_n(p)]) \longrightarrow^*_\beta W(d[lift^W_n(p')])$

Proof. The proof proceeds by induction on the structure of p, uses the fact that W has the general scheme, and a meta-substitution lemma stating that $a[cons^W(b)][s] =_W a[lift^W(s)][cons^W(b[s])]$, for any pure term a, any term $b \in \Lambda^t_W$ and any $n \geq 0$. We refer the reader to [25] for details.

Lemma 21. *Let W be a BSC. If $a \longrightarrow^*_\beta b$, then $a \longrightarrow^*_{\lambda_W} b$.*

Proof. The proof proceeds by induction on the number of reduction steps from a to b. We only show here the base case. Let $a \longrightarrow_\beta b$. Then $a = C[(\lambda c)d]$ and $b = C[c\{1 \leftarrow d\}]$. Since $W[c\{1 \leftarrow d\}]$ is equal to $W(c[cons^W(d)])$, then $a = C[(\lambda c)d] \longrightarrow_{Beta} C[c[cons^W(d)]] \longrightarrow^*_W C[c\{1 \leftarrow d\}]$.

Theorem 22 λ_W is confluent. *For every BSC W satisfying the general scheme, the relation λ_W is confluent.*

Proof. Using lemma 19, where R' is $\longrightarrow^*_\beta$, R_1 is $\longrightarrow_W$ and R_2 is $\longrightarrow_{Beta}$. The two properties required by the hypothesis of the lemma come from lemma 21 and lemma 20.

When $\widehat{Eta}$-expansions are also considered in the reduction system, the interpretation method is no longer useful because the second condition required by lemma 19 does not always hold, as the following example shows:

Example 1. Let us consider the extensional λ_υ-calculus and the following $\widehat{Eta}$-expansion step $a = \underline{1}[\#(\lambda\underline{1})] \longrightarrow_{\widehat{Eta}} (\lambda(\underline{1}[\uparrow]\ \underline{1}))[\#(\lambda\underline{1})] = b$. Then $W(a) = \lambda\underline{1}$ and $W(b) = \lambda((\lambda\underline{1})\ \underline{1})$, so neither $W(a)\longrightarrow^*_{\widehat{\eta}} W(b)$, nor $W(a)\longrightarrow^*_{\beta\widehat{\eta}} W(b)$. In fact, $W(a)$ is a $\beta\widehat{\eta}$-normal form so it cannot be reduced at all.

As the previous example suggests, the second condition required by the interpretation method can be weakened to recover confluence as follows:

$$a \longrightarrow_{R_2} b \text{ implies } R_1(a) =_{R'} R_1(b)$$

As a consequence, lemma 19 becomes a particular case of the following one:

Lemma 23 Generalized Interpretation Method. *Let $R = R_1 \cup R_2$, where R_1 is a confluent and strongly normalizing reduction relation and R_2 an arbitrary reduction relation. If there exists a reduction relation R' on the set of R_1-normal forms such that*

$$R' \subseteq R^* \text{ and } (a \longrightarrow_{R_2} b \text{ implies } R_1(a) =_{R'} R_1(b))$$

then if R' is confluent, R is also confluent.

When showing confluence of any relation $\longrightarrow_{\lambda_W \cup \widehat{Eta}}$, one usually takes R' as $\longrightarrow^*_{\beta\widehat{\eta}}$, R_1 as $\longrightarrow_W$ and R_2 as $\longrightarrow_{Beta \cup \widehat{Eta}}$. As every BSC W is confluent and strongly normalizing, and $\beta \cup \widehat{\eta}$-reduction is only defined on W-normal forms (that are pure terms), one is left to show the two properties required by the lemma. In particular, we will show that

$$\longrightarrow^*_{\beta\widehat{\eta}} \subseteq \longrightarrow^*_{\lambda_W \cup \widehat{Eta}} \text{ and } (a \longrightarrow_{Beta \cup \widehat{Eta}} b \text{ implies } W(a)\longrightarrow^*_{\widehat{\eta}}\ {}_{\beta}{}^*\!\longleftarrow W(b))$$

This is in fact stronger than what is required by lemma 23, but we think that this formulation is more useful to understand the interaction between $\widehat{Eta}$-expansion, terms and explicit substitutions. Exactly as in the case of the confluence proof of $\longrightarrow_{\lambda_W}$, the first property is quite simple to prove whereas the second one is the essential part of the general abstract proof, it can be summarized by the following lemma.

Lemma 24. *Let W be a BSC having the general scheme and let $p \in \Lambda_W$. If $p \longrightarrow_{\widehat{Eta}} p'$, then*
1) if p is a term, then $W(p)\longrightarrow^*_{\widehat{\eta}}\ {}_{\beta}{}^*\!\longleftarrow W(p')$
2) for every pure term d such that $d[p]$ is a term, and every $n \geq 0$, then
$W(d[lift^W_n(p)])\longrightarrow^*_{\widehat{\eta}}\ {}_{\beta}{}^*\!\longleftarrow W(d[lift^W_n(p')])$

Now, by lemma 20 together with lemma 24 we obtain

Corollary 25. *Let W be BSC having the general scheme and let a be a term of W such that $a \longrightarrow_{Beta \cup \widehat{Eta}} b$. Then $W(a)\longrightarrow^*_{\beta\widehat{\eta}}\ {}_{\beta\widehat{\eta}}{}^*\!\longleftarrow W(b)$.*

Lemma 26. *Let W be a BSC and let a be a pure term. Then $a \longrightarrow_{\widehat{\eta}} b$ implies $a\longrightarrow^*_{W \cup \widehat{Eta}} b$.*

Proof. Let $C[a] \longrightarrow_{\widehat{\eta}} C[\lambda(a^+\ \underline{1})]$, where $a \longrightarrow_{\widehat{\eta}} \lambda(a^+\ \underline{1})$. Then a is not a λ-abstraction and is not applied to another term. By definition $a \longrightarrow_{\widehat{Eta}} \lambda(a[\uparrow]\ \underline{1})$, and by lemma 12 $W(a[\uparrow])$ is exactly a^+, so that $\lambda(a[\uparrow]\ \underline{1})\longrightarrow^*_W \lambda(W(a[\uparrow])\ \underline{1}) = \lambda(a^+\ \underline{1})$ which implies $a\longrightarrow^*_{W \cup \widehat{Eta}} \lambda(a^+\ \underline{1})$ and concludes the proof.

Now, by lemma 26 together with lemma 21 we obtain

Corollary 27. *Let W be a BSC and let a be a pure term. Then $a\longrightarrow^*_{\beta\widehat{\eta}} b$ implies $a\longrightarrow^*_{\lambda_W \cup \widehat{Eta}} b$.*

Theorem 28 $\lambda_W \cup \widehat{Eta}$ is confluent. *For every BSC W having the general scheme, the relation $\lambda_W \cup \widehat{Eta}$ is confluent.*

Proof. By the generalized interpretation method (lemma 23), where R' is $\longrightarrow^*_{\beta\widehat{\eta}}$, R_1 is $\longrightarrow_W$ and R_2 is $\longrightarrow_{Beta \cup \widehat{Eta}}$. Note that $\longrightarrow_W$ is confluent and strongly normalizing by conditions 1 and 2, the two other properties required by the hypothesis of the lemma come from Corollary 27 and Corollary 25.

4 Some λ-calculi fitting the general scheme

In this section we apply our method to some well-known λ-calculi with explicit substitutions such as λ_υ, λ_σ and $\lambda_{\sigma_\Uparrow}$ and also to some new calculi that we proposed here, namely, λ_d and λ_{dn}. For that, we have just to associate functions $lift^W$ and $cons^W$ to each calculus W, and then verify not only the existence of the $Lambda$, App and $Beta$ rule, but also conditions required by definitions 11 (basic conditions) and 18 (general scheme).

The λ_σ, $\lambda_{\sigma_\Uparrow}$ and λ_υ calculi

In the λ_σ-calculus [1], terms and substitutions are defined by the grammar

$$(Terms)\ \ a ::= \underline{1} \mid aa \mid \lambda a \mid a[s] \qquad (Substitutions)\ \ s ::= id \mid a.s \mid \ \uparrow\ \mid s \circ s$$

As a consequence, any variable $\underline{n+1}$ (for $n \geq 1$) is represented by $\underline{1}[\uparrow^n] = \underline{1}[\underbrace{\uparrow \circ(\uparrow \circ(\ldots \circ \uparrow))}_{n\ times}]$, which becomes in this way a pure term.

The lift-function is defined by $lift^\sigma(s) = \underline{1}.(s\circ \uparrow)$ and the cons-function by $cons^\sigma(b) = b.id$. The reduction rules are the following:

$$\begin{array}{lll}
(Beta) & (\lambda a)b & \longrightarrow\ a[b.id]\\
(App) & (a\ b)[s] & \longrightarrow\ (a[s]\ b[s])\\
(Lambda) & (\lambda a)[s] & \longrightarrow\ \lambda(a[\underline{1}.(s\circ \uparrow)])\\
(Clos) & (a[s])[t] & \longrightarrow\ a[s \circ t]\\
(VarId) & \underline{1}[id] & \longrightarrow\ \underline{1}\\
(VarCons) & \underline{1}[a.s] & \longrightarrow\ a\\
(IdL) & id \circ s & \longrightarrow\ s\\
(ShiftId) & \uparrow \circ id & \longrightarrow\ \uparrow\\
(ShiftCons) & \uparrow \circ(a.s) & \longrightarrow\ s\\
(Ass) & (s_1 \circ s_2) \circ s_3 & \longrightarrow\ s_1 \circ (s_2 \circ s_3)\\
(Map) & (a.s) \circ t & \longrightarrow\ a[t].(s \circ t)
\end{array}$$

The implementation of the lift-function is much simpler in the $\lambda_{\sigma_\Uparrow}$-calculus [22, 7]: it only uses a unary substitution constructor instead of the two constants and two operators used in the case of λ_σ. Terms and substitutions of the $\lambda_{\sigma_\Uparrow}$-calculus can be defined by the following grammar:

$$(Terms)\ \ a ::= \underline{n} \mid aa \mid \lambda a \mid a[s] \qquad (Substitutions)\ \ s ::= id \mid a.s \mid \ \Uparrow(s) \mid \ \uparrow\ \mid s \circ s$$

The lift-function is defined by $lift^{\sigma_\Uparrow}(s) = \Uparrow(s)$ and the cons-function by $cons^{\sigma_\Uparrow}(b) = b.id$. Reduction rules of the calculus are the following:

$$\begin{array}{lll@{\qquad}lll}
(Beta) & (\lambda a)b & \longrightarrow\ a[b.id] & (Ass) & (s_1 \circ s_2) \circ s_3 & \longrightarrow\ s_1 \circ (s_2 \circ s_3)\\
(App) & (a\ b)[s] & \longrightarrow\ (a[s]\ b[s]) & (Map) & (a.s) \circ t & \longrightarrow\ a[t].(s \circ t)\\
(Lambda) & (\lambda a)[s] & \longrightarrow\ \lambda(a[\Uparrow(s)]) & (Shift) & \uparrow \circ(a.s) & \longrightarrow\ s\\
(Clos) & (a[s])[t] & \longrightarrow\ a[s \circ t] & (ShiftLift1) & \uparrow \circ \Uparrow(s) & \longrightarrow\ s\circ \uparrow\\
(VarShift1) & \underline{n}[\uparrow] & \longrightarrow\ \underline{n+1} & (ShiftLift2) & \uparrow \circ(\Uparrow(s) \circ t) & \longrightarrow\ s \circ (\uparrow \circ t)\\
(VarShift2) & \underline{n}[\uparrow \circ s] & \longrightarrow\ \underline{n+1}[s] & (Lift1) & \Uparrow(s)\circ \Uparrow(t) & \longrightarrow\ \Uparrow(s \circ t)\\
(FVar) & \underline{1}[a.s] & \longrightarrow\ a & (Lift2) & \Uparrow(s) \circ (\Uparrow(t) \circ u) & \longrightarrow\ \Uparrow(s \circ t) \circ u\\
(FVarLift1) & \underline{1}[\Uparrow(s)] & \longrightarrow\ \underline{1} & (LiftEnv) & \Uparrow(s) \circ (a.t) & \longrightarrow\ a.(s \circ t)\\
(FVarLift2) & \underline{1}[\Uparrow(s) \circ t] & \longrightarrow\ \underline{1}[t] & (IdL) & id \circ s & \longrightarrow\ s\\
(RVar) & \underline{n+1}[a.s] & \longrightarrow\ \underline{n}[s] & (IdR) & s \circ id & \longrightarrow\ s\\
(RVarLift1) & \underline{n+1}[\Uparrow(s)] & \longrightarrow\ \underline{n}[s\circ \uparrow] & (LiftId) & \Uparrow(id) & \longrightarrow\ id\\
(RVarLift2) & \underline{n+1}[\Uparrow(s) \circ t] & \longrightarrow\ \underline{n}[s \circ (\uparrow \circ t)] & (Id) & a[id] & \longrightarrow\ a
\end{array}$$

In a different spirit, Pierre Lescanne [29] introduces the λ_υ-calculus, which is, in some sense, the *minimal* calculus of explicit substitutions that satisfies the basic conditions in definition 11. As a consequence, it can be derived from our general scheme in a very direct way: we have just to associate a unary substitution constructor to the cons-function $cons^\upsilon$, and another one to the the lift-function

$lift^{\upsilon}$, then any of the three last basic conditions becomes a rewrite rule; whereas conditions on substitutions in definition 18 can be immediately deduced.

Terms and substitutions of the λ_{υ}-calculus can be defined by the following grammar:

$$(Terms)\ \ a ::= \underline{n} \mid aa \mid \lambda a \mid a[s] \qquad (Substitutions)\ \ s ::= \#(a) \mid\ \Uparrow(s) \mid\ \uparrow$$

The lift-function is defined by $lift^{\upsilon}(s) = \Uparrow(s)$ and the cons-function by $cons^{\upsilon}(b) = \#(b)$. We use here $\#$ instead of the $/$ symbol introduced in [37] and used in [29]. The reduction rules are:

$$\begin{array}{llll llll}
(Beta) & (\lambda a)b & \longrightarrow & a[\#(b)] & (App) & (a\,b)[s] & \longrightarrow & (a[s]\,b[s]) \\
(Lambda) & (\lambda a)[s] & \longrightarrow & \lambda(a[\Uparrow(s)]) & (FVar) & \underline{1}[\#(a)] & \longrightarrow & a \\
(RVar) & \underline{n+1}[\#(a)] & \longrightarrow & \underline{n} & (FVarLift) & \underline{1}[\Uparrow(s)] & \longrightarrow & \underline{1} \\
(RVarLift) & \underline{n+1}[\Uparrow(s)] & \longrightarrow & \underline{n}[s][\uparrow] & (VarShift) & \underline{n}[\uparrow] & \longrightarrow & \underline{n+1}
\end{array}$$

The λ_d and λ_{dn} calculi

Mainly inspired from the λ_{τ}-calculus in [37], which was in turn inspired by [16], we propose here the λ_d-calculus by putting together the nice properties of the three previous substitution calculi: we just use unary substitution operators for the lift-function and the cons-function, keeping composition of substitutions which is one the main features needed to describe different implementations of functional programming. Terms and substitutions are defined by

$$(Terms)\ \ a ::= \underline{1} \mid aa \mid \lambda a \mid a[s] \qquad (Substitutions)\ \ s ::= id \mid \#(a) \mid\ \Uparrow(s) \mid\ \uparrow\ \mid s \circ s$$

We represent, exactly as in the case of the λ_{σ}-calculus, any variable $\underline{n+1}$ (for $n \geq 1$) by $\underline{1}[\uparrow^n]$.

The functions $lift^d$ and $cons^d$ are defined by $lift^d(s) = \Uparrow(s)$ and $cons^d(b) = \#(b)$. The reduction rules are the following ones:

$$\begin{array}{llll llll}
(Beta) & (\lambda a)b & \longrightarrow & a[\#(b)] & (App) & (a\,b)[s] & \longrightarrow & (a[s]\,b[s]) \\
(Lambda) & (\lambda a)[s] & \longrightarrow & \lambda(a[\Uparrow(s)]) & (Clos) & (a[s])[t] & \longrightarrow & a[s \circ t] \\
(FVar1) & \underline{1}[\#(a)] & \longrightarrow & a & (FVar2) & \underline{1}[\#(a) \circ s] & \longrightarrow & a[s] \\
(FVarLift1) & \underline{1}[\Uparrow(s)] & \longrightarrow & \underline{1} & (FVarLift2) & \underline{1}[\Uparrow(s) \circ t] & \longrightarrow & \underline{1}[t] \\
(Shift1) & \uparrow \circ \#(a) & \longrightarrow & id & (Shift2) & \uparrow \circ (\#(a) \circ s) & \longrightarrow & s \\
(ShiftLift1) & \uparrow \circ \Uparrow(s) & \longrightarrow & s \circ \uparrow & (ShiftLift2) & \uparrow \circ (\Uparrow(s) \circ t) & \longrightarrow & s \circ (\uparrow \circ t) \\
(Lift1) & \Uparrow(s) \circ \Uparrow(t) & \longrightarrow & \Uparrow(s \circ t) & (Lift2) & \Uparrow(s) \circ (\Uparrow(t) \circ u) & \longrightarrow & \Uparrow(s \circ t) \circ u \\
(IdL) & id \circ s & \longrightarrow & s & (IdR) & s \circ id & \longrightarrow & s \\
(LiftId) & \Uparrow(id) & \longrightarrow & id & (Id) & a[id] & \longrightarrow & a \\
(Ass) & (s_1 \circ s_2) \circ s_3 & \longrightarrow & s_1 \circ (s_2 \circ s_3) & & & &
\end{array}$$

Note that the $MapEnv$ rule of the λ_{τ}-calculus in [37] $\#(a) \circ s \longrightarrow\ \Uparrow(s) \circ \#(a[s])$ is replaced here by $FVar2$ and $Shift2$. This may be in principle superfluous, but is essential to make our calculus fit in the general scheme. This difference makes it possible to prove confluence and strong normalization of d, while confluence remains still as a conjecture in the case of the τ-calculus and strong normalization does not hold [28]. However, the rule $MapEnv$ is essential if one wants to recover confluence on open terms [6], and is at the same time, as explained by [32] and [4], the cause of non termination in many λ-calculi with explicit substitutions. It is then natural to conjecture that λ_d preserves β-strong normalization.

Theorem 29. *The d-calculus is locally confluent and strongly normalizing.*

Proof. The proof of local confluence can be done automatically, for example by using CIME [31]. For strong normalization, we know that the $\sigma_{\Uparrow}$-calculus is strongly normalizing [19], so it is sufficient to interpret the d-calculus into the $\sigma_{\Uparrow}$-calculus in the following way:

Let $\mathcal{T}$ be a function from objects in Λ_d into objects in $\Lambda_{\sigma_{\Uparrow}}$ defined to be the identity on all the objects, but on substitutions of the form $\#(a)$, where $\mathcal{T}(\#(a))$ is equal to $\mathcal{T}(a).id$.

We can show by case-analysis that $a \longrightarrow_d b$ implies $\mathcal{T}(a) \longrightarrow^+_{\sigma_{\Uparrow}} \mathcal{T}(b)$, so that strong normalization of $\sigma_{\Uparrow}$ implies that of d.

[6] Indeed, the open term $((\lambda X)a)[s]$ reduces to both $X[\#(a) \circ s]$ and $X[\Uparrow(s) \circ \#(a[s])]$, which have no common reduct in λ_d but can be joined in the λ_{τ}-calculus proposed in [37].

The λ_{dn}-calculus is just λ_d plus natural numbers to represent variables. It is a natural extension of λ_d which also satisfies the abstract properties required by the general scheme.

The sets of terms and substitutions are defined as follows:

$$(Terms)\ \ a ::= \underline{n} \mid aa \mid \lambda a \mid a[s] \qquad (Substitutions)\ \ s ::= id \mid \#(a) \mid\ \Uparrow(s) \mid\ \uparrow\ \mid s \circ s$$

The function $lift^{dn}$ (resp. $cons^{dn}$) is defined exactly as $lift^{d}$ (resp. $cons^{d}$). We add the following set of rules to those of the λ_d-calculus.

$$\begin{array}{llll}
(VarShift1) & \underline{n}[\uparrow] & \longrightarrow & \underline{n+1} \\
(RVar) & \underline{n+1}[\#(a)] & \longrightarrow & \underline{n} \\
(RVarLift2) & \underline{n+1}[\Uparrow(s) \circ t] & \longrightarrow & \underline{n}[s \circ (\uparrow \circ t)]
\end{array}
\qquad
\begin{array}{llll}
(VarShift2) & \underline{n}[\uparrow \circ s] & \longrightarrow & \underline{n+1}[s] \\
(RVarLift1) & \underline{n+1}[\Uparrow(s)] & \longrightarrow & \underline{n}[s \circ \uparrow] \\
(RMap) & \underline{n+1}[\#(a) \circ s] & \longrightarrow & \underline{n}[s]
\end{array}$$

Theorem 30. *The dn-calculus is locally confluent and strongly normalizing.*

Proof. The proof of local confluence can be done automatically, for example by using CIME [31]. For strong normalization, we proceed exactly as in theorem 29.

Theorem 31 Special cases of the general scheme. *The non-extensional and extensional versions of the λ_σ, $\lambda_{\sigma_\Uparrow}$, λ_υ, λ_d and λ_{dn} calculi are all confluent.*

Proof. The proof comes from theorem 22 and theorem 28. Since $Lambda$, App and $Beta$ are well defined according to the definition of each cons-function and lift-function, we are left to show that these calculi are all basic (definition 11) and satisfy the general scheme (definition 18).

- Basic Conditions:
  - The proofs of strong normalization can be found in [21, 8] for σ, in [22] for $\sigma_\Uparrow$ and in [29] for υ. Strong normalization for d (resp. dn) follows from theorem 29 (resp. theorem 30).
  - Confluence comes from local confluence and strong normalization by Newman's lemma.
  - σ-normal forms of terms are pure terms by [1]. The same happens for $\sigma_\Uparrow$ (see [22]) and υ (see [29]). For the d and dn calculi, an induction on the structure of a suffices (see [25] for details).
  - Conditions 4, 5, 6 and 7 in definition 11 are straightforward in all the cases.
- General Scheme: we have to verify that for every variable $\underline{n}$ and every constructor ξ one of the cases described in definition 18 holds. By a simple inspection, we can observe in all these calculi that the behavior of id, $\uparrow$, $\circ$ and $\Uparrow$ corresponds to the first case of the definition, while . and $\#$ corresponds to the second case for the variable $\underline{1}$ and to the first case for any other variable $\underline{m+1}$.

5 Deriving confluence properties of other calculi

Our method can also be applied to λ-calculi that do not fit our definition of SC but that can be translated to some other substitution calculi. In particular, we show that the non-extensional and extensional versions of the λ_s-calculus [27] are confluent on closed terms.

Terms and substitutions of the λ_s-calculus are given by the following grammar:

$$(Terms)\ \ a ::= \underline{n} \mid aa \mid \lambda a \mid a\sigma^i b \mid \varphi^i_k a$$

The reduction rules are:

$$\begin{array}{llll}
(\sigma\textit{-generation}) & (\lambda a)b & \longrightarrow & a\sigma^1 b \\
(\sigma\textit{-}\lambda\textit{-generation}) & (\lambda a)\sigma^j b & \longrightarrow & \lambda(a\sigma^{j+1} b) \\
(\sigma\textit{-destruction}) & \underline{n}\sigma^j b & \longrightarrow & \begin{cases} \underline{n-1} & \text{if } n > j \\ \varphi^j_0 b & \text{if } n = j \\ \underline{n} & \text{if } n < j \end{cases} \\
(\sigma\textit{-app-transition}) & (a\ b)\sigma^j c & \longrightarrow & (a\sigma^j c)(b\sigma^j c) \\
(\Phi\textit{-}\lambda\textit{-transition}) & \varphi^i_k(\lambda a) & \longrightarrow & \lambda(\varphi^i_{k+1} a) \\
(\Phi\textit{-app-transition}) & \varphi^i_k(a\ b) & \longrightarrow & (\varphi^i_k a)(\varphi^i_k b) \\
(\Phi\textit{-destruction}) & \varphi^i_k \underline{n} & \longrightarrow & \begin{cases} \underline{n+i-1} & \text{if } n > k \\ \underline{n} & \text{if } n \le k \end{cases}
\end{array}$$

Even if this calculus does not fit in the general scheme defined in this paper, it turns out that it can be translated into a substitution calculus by preserving the desired confluence properties. In particular, λ_s turns out to be confluent as shown in [27], and, on the other hand, we can add an $\widehat{Eta}$-expansion rule to the λ_s-calculus preserving confluence of the whole reduction system. In order to define the $\widehat{Eta}$-rule in the λ_s-calculus, we first need the notions of Applied-Terms and Lambda-Abstractions introduced in definition 13.

Definition 32 Applied-Terms and Lambda-Abstractions. A term a is a *Lambda-Abstraction* if either it is λb; or it is equal to $b\sigma^i c$ or $\varphi^i_k b$ and b is a Lambda-Abstraction. Similarly, a subterm a is said to be an *Applied-Term* in $C[a]$ if it is applied in $C[a]$; while a is said to be an *Applied-Term* in $C[a\sigma^i c]$ (resp. in $C[\varphi^i_k a]$) if a is an Applied-Term in $C[a]$.

We are now ready to define $\widehat{Eta}$-expansions in the λ_s-calculus:

Definition 33 $\widehat{Eta}$-expansion in λ_s. The condition $\widehat{Eta}$-expansion in λ_s is defined to be the reflexive-transitive closure of the following reduction step:

$$C[a] \longrightarrow_{\widehat{Eta}} C[\lambda((\varphi^2_0 a)\ \underline{1})],\ \text{if} \begin{cases} a : A \to B \\ a \text{ is not a Lambda-Abstraction} \\ a \text{ is not an Applied-Term in } C[a] \end{cases}$$

As expected, β-reduction and $\widehat{\eta}$-expansion can be simulated in the $\lambda_s \cup \widehat{Eta}$-calculus:

Theorem 34. *Let a be a pure term. Then $a \longrightarrow_{\beta\widehat{\eta}} b$ implies $a \longrightarrow^*_{\lambda_s \cup \widehat{Eta}} b$.*

Proof. If $a \longrightarrow_\beta b$, then the property holds by lemma 11 in [27]. Let $C[a] \longrightarrow_{\widehat{\eta}} C[\lambda(a^+\ \underline{1})]$, where $a \longrightarrow_{\widehat{\eta}} \lambda(a^+\ \underline{1})$. Then a is not a λ-abstraction and is not applied to another term, so by definition $a \longrightarrow_{\widehat{Eta}} \lambda((\varphi^2_0 a)\ \underline{1})$. By lemma 10 in [27] we have $\varphi^2_0 a \longrightarrow^*{}_s U^2_0(s(a)))$, but $s(a) = a$ since a is a pure term, and $U^2_0(a) = a^+$, which concludes the proof.

The λ_s-calculus has already been shown to be confluent in [27], so the aim here is to show that $\lambda_s \cup \widehat{Eta}$ is also confluent. We will use the following properties:

- s is strongly normalizing.
- s-normal forms of terms are pure terms.
- If a and b are pure terms, then $a \longrightarrow_{\beta\widehat{\eta}} b$ implies $a \longrightarrow^*_{\lambda_s \cup \widehat{Eta}} b$.
- There is a translation $\mathcal{T}$ from Λ^t_s to terms of some SC W such that $\mathcal{T}$ is the identity on pure terms and $a \longrightarrow_{\lambda_s \cup \widehat{Eta}} b$ implies $\mathcal{T}(a) \longrightarrow^*_{\lambda_W \cup \widehat{Eta}} \mathcal{T}(b)$.

We claim that these properties are sufficient to show that $\lambda_s \cup \widehat{Eta}$ is confluent. Indeed,

Proof. Let a be a term such that $b\ {}_{\lambda_s \cup \widehat{Eta}}{}^*\!\longleftarrow a \longrightarrow^*_{\lambda_s \cup \widehat{Eta}} c$. Since s is strongly normalizing, we can reduce b and c to their s-normal forms so that $s(b) = b'\ {}_{\lambda_s \cup \widehat{Eta}}{}^*\!\longleftarrow a \longrightarrow^*_{\lambda_s \cup \widehat{Eta}} c' = s(c)$. By the fourth hypothesis $\mathcal{T}(b')\ {}_{\lambda_W \cup \widehat{Eta}}{}^*\!\longleftarrow \mathcal{T}(a) \longrightarrow^*_{\lambda_W \cup \widehat{Eta}} \mathcal{T}(c')$, and by corollary 25 $W(\mathcal{T}(b')) \longrightarrow^*_{\beta\widehat{\eta}} k_1\ {}_{\beta\widehat{\eta}}{}^*\!\longleftarrow W(\mathcal{T}(a))$ and $W(\mathcal{T}(a)) \longrightarrow^*_{\beta\widehat{\eta}} k_2\ {}_{\beta\widehat{\eta}}{}^*\!\longleftarrow W(\mathcal{T}(c'))$, so that $k_1 \longrightarrow^*_{\beta\widehat{\eta}} k\ {}_{\beta\widehat{\eta}}{}^*\!\longleftarrow k_2$ by confluence of $\longrightarrow^*_{\beta\widehat{\eta}}$ (theorem 6). Since b' and c' are pure terms by the second hypothesis, then $\mathcal{T}(b') = b'$ and $\mathcal{T}(c') = c'$ hold by the fourth hypothesis and $W(b') = b'$ and $W(c') = c'$ hold by definition of substitution calculi. We then have $b' \longrightarrow^*_{\beta\widehat{\eta}} k\ {}_{\beta\widehat{\eta}}{}^*\!\longleftarrow c'$, and by the third hypothesis we obtain $b' \longrightarrow^*_{\lambda_s \cup \widehat{Eta}} k\ {}_{\lambda_s \cup \widehat{Eta}}{}^*\!\longleftarrow c'$ which concludes the proof.

Now, since s is strongly normalizing by Corollary 1 in [27], s-normal forms of terms are pure terms by lemma 12 in [26] and whenever a and b are pure terms, then $a \longrightarrow_{\beta\widehat{\eta}} b$ implies $a \longrightarrow^*_{\lambda_s \cup \widehat{Eta}} b$ by theorem 34, then we are left to show that there is a translation $\mathcal{T}$ satisfying the required properties.

As showed in [27], the λ_s-calculus has a nice translation $\mathcal{T}$ into the λ_υ-calculus that allows also to derive the confluence of $\lambda_s \cup \widehat{Eta}$ via the following property:

Proposition 35. *If* $a \longrightarrow_{\lambda_s \cup \widehat{Eta}} b$*, then* $\mathcal{T}(a) \longrightarrow^*_{\lambda_\upsilon \cup \widehat{Eta}} \mathcal{T}(b)$*.*

Proof. If $a \longrightarrow_{\lambda_s \cup \widehat{Eta}} b$ is not a root reduction, then we can proceed by induction on the structure of a. For that, it is important to notice that only root expansions are translated into root expansions, so that the inductive proof concerning this case is straightforward. If $a \longrightarrow_{\lambda_s \cup \widehat{Eta}} b$ is a root reduction, we proceed by case-analysis.

We can now finally obtain the following:

Theorem 36. $\longrightarrow^*_{\lambda_s \cup \widehat{Eta}}$ *is confluent.*

6 Conclusion

We give a general and homogeneous treatment of λ-calculi with explicit substitutions. Such treatment is essential in the field of implementation of functional programming because of the growing number of such calculi proposed in the literature. The major contributions of this paper can be summarized by the following points:

- We propose a general scheme for λ-calculi with explicit substitutions which allows to reason in an abstract way and to deal with many different calculi. In particular, we are able to deal with λ_σ, $\lambda_{\sigma_{\Uparrow}}$ and λ_υ, between others, using the same formalism and tools.
- We study confluence properties of extensional and non-extensional λ-calculi with explicit substitutions, where extensionality is interpreted by η-expansion. This reading of extensionality has never been studied before in the framework of explicit substitutions, and it turns out to be much more natural and less complex than previous interpretations via η-contraction.
- We derive two *new* calculi with explicit substitutions having nice properties. In particular, they have a composition operator for substitutions and simple constructors to implement all the other kind of substitutions. We conjecture that those calculi preserve β-strong normalization.

For those few calculi not fitting in the general scheme (such as λ_s [27]) that can be translated into a substitution calculus, we also showed how to reason about confluence of the combination of $Beta$-reduction with $\widehat{Eta}$-expansion. This is in some sense a generalization of the technique proposed via the general scheme.

We think that an abstract approach in the spirit of the one proposed in this paper would also be useful to study confluence properties on *open terms* and preservation of β-strong normalization, the challenge being the definition of an appropriate general scheme associated to those properties.

Acknowledgments

I would like to thank Pierre Lescanne and Gilles Dowek for interesting discussions, Roberto Di Cosmo for careful proofreading and useful comments, and Alejandro Ríos and Paul-André Mellies for explaining me some key points in the theory of calculi with explicit subtitutions.

References

[1] M. Abadi, L. Cardelli, P-L. Curien, and J-J. Lévy. Explicit substitutions. *JFP*, 4(1):375–416, 1991.

[2] Y. Akama. On mints' reductions for ccc-calculus. In *TLCA*, LNCS 664, 1993.

[3] H. Barendregt. *The Lambda Calculus; Its syntax and Semantics (revised edition)*. North Holland, 1984.

[4] Z-E-A. Benaissa and D. Briaud and P. Lescanne and J. Rouyer-Degli. λ_υ, a calculus of explicit substitutions which preserves strong normalisation. Available from `http://www.loria.fr/ lescanne/publications.html`. 1995.

[5] D. Briaud. An explicit *eta* rewrite rule. In *TLCA*, LNCS 902, 1995.

[6] P-L. Curien and R. Di Cosmo. A confluent reduction system for the λ-calculus with surjective pairing and terminal object. In *ICALP*, LNCS 510, 1991.

[7] P-L. Curien, T. Hardin, and J-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. Technical Report 1617, INRIA Rocquencourt, 1992.

[8] P-L. Curien, T. Hardin, and A. Ríos. Strong normalisation of substitutions. In *MFCS'92*, LNCS 629, 1992.

[9] N. de Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indag. Mat.*, 5(35):381–392, 1972.

[10] N. de Bruijn. Lambda-calculus notation with namefree formulas involving symbols that represent reference transforming mappings. *Indag. Mat.*, (40):384–356, 1978.

[11] Roberto Di Cosmo and Delia Kesner. A confluent reduction for the extensional typed $\lambda-$calculus with pairs, sums, recursion and terminal object. In *ICALP*, LNCS 700, 1993.

[12] Roberto Di Cosmo and Delia Kesner. Combining first order algebraic rewriting systems, recursion and extensional typed lambda calculi. In *ICALP*, LNCS 820, 1994.

[13] R. Di Cosmo and D. Kesner. Rewriting with extensional polymorphic λ-calculus. In *CSL*, 1995.

[14] R. Di Cosmo and A. Piperno. Expanding extensional polymorphism. In *TLCA*, LNCS 902, 1995.

[15] D. Dougherty. Some lambda calculi with categorical sums and products. In *RTA*, LNCS 690, 1993.

[16] Thomas Ehrhard. *Une sémantique catégorique des types dépendants. Application au calcul des constructions.* Thèse de doctorat, Université de Paris VII, 1988.

[17] J-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1990.

[18] T. Hardin. *Résultats de confluence pour les règles fortes de la logique combinatoire catégorique et liens avec les lambda-calculs*. Thèse de doctorat, Université de Paris VII, 1987.

[19] T. Hardin. η-reduction for explicit substitutions. In *ALP'92*, LNCS 632, 1992.

[20] T. Hardin. Eta-conversion for the languages of explicit substitutions. *AAECC*, 5:317–341, 1994.

[21] T. Hardin and A. Laville. Proof of termination of the rewriting system subst on c.c.l. *TCS*, 1986.

[22] T. Hardin and J-J. Lévy. A confluent calculus of substitutions. In *France-Japan Art. Int. and Comp. Sci. Symp.*, 1989.

[23] G. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. Thèse de doctorat d'état, Université Paris VII, 1976.

[24] C. Barry Jay and N. Ghani. The virtues of eta-expansion. *JFP*, 5(2):135–154, 1995.

[25] D. Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions, 1995. Available as `ftp://ftp.lri.fr/LRI/articles/kesner/explicit.ps.gz`.

[26] F. Kamareddine and A. Ríos. The confluence of the λs_e-calculus via a generalized interpretation method, 1995. Draft.

[27] F. Kamareddine and A. Ríos. A λ-calculus à la de bruijn with explicit substitutions. In *PLILP*, LNCS 982, 1995.

[28] P. Lescanne. Personal Communication.

[29] P. Lescanne. From λ_σ to λ_ν, a journey through calculi of explicit substitutions. In *POPL*, pages 60–69. ACM, 1994.

[30] P. Lescanne and J. Rouyer-Degli. The calculus of explicit substitutions $\lambda\nu$. Technical Report, INRIA, Lorraine, 1994.

[31] C. Marché. *Réécriture modulo une théorie présentée par un système convergent et décidabilité des problèmes du mot dans certains classes de théories équationnelles*. PhD thesis, Université Paris-Sud, Orsay, 1993.

[32] P-A. Mellies. Typed λ-calculi with explicit substitutions may not terminate. In *TLCA*, LNCS 902, 1995.

[33] G. Mints. Closed categories and the theory of proofs. *Zap. Nauch. Semin. Leningradskogo*, 68:83–114, 1977.

[34] G. Mints. Teorija categorii i teoria dokazatelstv.I. *Aktualnye problemy logiki i metodologii nauky*, pages 252–278, 1979.

[35] G. Pottinger. The Church Rosser Theorem for the Typed lambda-calculus with Surjective Pairing. *Notre Dame Jour. of Formal Logic*, 22(3):264–268, 1981.

[36] D. Prawitz. Ideas and results in proof theory. *Proceedings of the 2nd Scandinavian Logic Symposium*, pages 235–307, 1971.

[37] A. Ríos. *Contribution à l'étude des λ-calculus avec substitutions explicites*. Thèse de doctorat, Université de Paris VII, 1993.

[38] V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: the higher-order case. In *LICS*, 1994