A de Bruijn notation for higher-order rewriting (Extended Abstract)

Eduardo Bonelli[1,2], Delia Kesner[2], Alejandro Ríos[1]

[1] Departamento de Computación - Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Pabellón I, Ciudad Universitaria (1428), Buenos Aires, Argentina. {ebonelli,rios}@dc.uba.ar
[2] LRI (UMR 8623) - Bât 490, Université de Paris-Sud, 91405 Orsay Cedex, France. kesner@lri.fr

Abstract. We propose a formalism for higher-order rewriting in de Bruijn notation. This notation not only is used for terms (as usually done in the literature) but also for metaterms, which are the syntactical objects used to express general higher-order rewrite systems. We give formal translations from higher-order rewriting with names to higher-order rewriting with de Bruijn indices, and vice-versa. These translations can be viewed as an *interface* in programming languages based on higher-order rewrite systems, and they are also used to show some properties, namely, that both formalisms are operationally equivalent, and that confluence is preserved when translating one formalism into the other.

1 Introduction

Higher-order (term) rewriting concerns the transformation of terms in the presence of binding mechanisms for variables. Implementing higher-order rewriting requires, beforehand, taking care of a complex notion of substitution operation and of renaming of bound variables (α-conversion). As a paradigmatic example, the β-reduction axiom of λ-calculus [1], expressed $(\lambda x.M)N \longrightarrow_\beta M\{x \leftarrow N\}$, may be interpreted as: the result of executing function $\lambda x.M$ over argument N is obtained by substituting N for all (free) occurrences of x in M. Any implementation of higher-order rewriting must include instructions for computing this substitution. Although from the meta-level the execution of a substitution is atomic, the cost of computing it highly depends on the form of the terms, specially if unwanted variable capture conflicts must be avoided by renaming bound variables. De Bruijn indices take care of renaming because the representation of variables by indices completely eliminates unwanted capture of variables. However, de Bruijn formalisms have only been studied for particular systems (and only on the term level) and no general framework of higher-order rewriting with indices has been proposed. We address this problem here by focusing not only on de Bruijn terms (as usually done in the literature for λ-calculus [11]) but also on de Bruijn metaterms, which are the syntactical objects used to express any general higher-order rewrite system formulated in a de Bruijn context.

Many higher-order rewrite systems (HORS) exist and work in the area is currently very active: *CRS* [14], *ERS* [12], *CERS* [13], *HRS* [15], the systems in [22] and [20]. We choose in this work to use *ERS* because their syntax and semantics are simple and natural (they allow for example to write β-reduction in λ-calculus as usual while *CRS* do not) and the correspondence between *ERS* and *HRS* has already been established [21]. We shall begin with (a slightly simplified version of) the *ERS* formalism, that we shall call *SERS* (S for simplified) and introduce the de Bruijn index based higher-order rewrite system $SERS_{DB}$.

Our work is the first step in the construction of a formal interpretation of higher-order rewriting via a *first-order* theory. This kind of simulation would be possible with the aid of *explicit substitutions*. Indeed, this work follows, in some sense, the lines of [8] which interprets higher-order formalisms/problems into their respective first-order ones.

Our formalism is developed in order to be used as an *interface* of a programming language based on higher-order rewriting. Of course, the use of variable name based formalisms are necessary for humans to interact with computers in a user-friendly way. Clearly technical resources like de Bruijn indices and explicit substitutions should live behind the scene, in other words, should be implementation concerns. Moreover, it is required of whatever is behind the scene to be as faithful as possible as regards the formalism it is implementing. So a key issue shall be the detailed study of the relationship between SERS and $SERS_{DB}$. The translations we propose between them are extensions to higher-order of the translations studied in [11] and presented in [5].

As regards existing higher-order rewrite formalisms based on de Bruijn index notation and/or explicit substitutions to the best of the authors' knowledge there are but two: *Explicit CRS* [3] and *XRS* [16]. In [3] explicit substitutions a la $\lambda\mathtt{x}$ [19, 2] are added to the *CRS* formalism as a first step towards using higher-order rewriting with explicit substitutions for modeling the evaluation of functional programs in a faithful way. Since this is done in a variable name setting α-conversion must be dealt with as in *CRS*. Pagano's *XRS* constitutes the first HORS which fuses de Bruijn index notation and explicit substitutions. It is presented as a generalization of the $\lambda\sigma_{\Uparrow}$-calculus [6] but no connection has been established between *XRS* and well-known systems such as *CRS*, *ERS* and *HRS*. Indeed, it is not clear at all how some seemingly natural rules expressible, say, in the *ERS* formalism, may be written in an *XRS*. As an example, consider a rewrite system for logical expressions such that if $imply(e_1, e_2)$ reduces to the constant $true$ then e_1 logically implies e_2 in classical first-order predicate logic. A possible rewrite rule could be:

$$(imp) \quad imply(\exists x \forall y M, \forall y \exists x M) \longrightarrow true$$

A naïve attempt might consider the rewrite rule

$$(imp_{db}) \quad imply(\exists \forall M, \forall \exists M) \longrightarrow true$$

as a possible representation of this rule in the *XRS* formalism, but it does not have the desired effect since $\exists\forall M$ and $\forall\exists M$ correspond to $\exists x \forall y M$ and $\forall x \exists y M$ but $\forall x \exists y M$ and $\forall y \exists x M$ are not equivalent. Note that regardless of the fact that *XRS* incorporate explicit substitutions, this problem arises already at the level of de Bruijn notation. Another example of interest is:

$$(\eta) \quad \lambda x.(M\,x) \longrightarrow M \text{ if } x \text{ is not free in } M$$

which is usually expressed in a de Bruijn based system with explicit substitutions

$$(\eta_{db}) \quad \lambda(M1) \longrightarrow N \text{ if } M =_{\mathcal{C}} N[\uparrow]$$

where $M =_{\mathcal{C}} N$ means that M and N are equivalent modulo the theory of explicit substitutions $\mathcal{C}$. Neither the (imp) rule nor (η_{db}) is possible in the *XRS* formalism so that they do not have in principle the same expressive power as *ERS*. We shall propose de Bruijn based HORS that will allow such rules to be faithfully represented.

The main contribution of this paper is a general de Bruijn notation for higher-order syntax which bridges the gap between higher-order rewriting with names and with indices. This formalism suggests a first-order tool to implement HORS, which in contrast to [16] would represent *all* the HORS used in practice.

The rest of the paper is organized as follows. Section 2 introduces our work and study scenario, the *SERS* formalism. The de Bruijn based formalism $SERS_{DB}$ is defined in Section 3. Section 4 takes a close-up view of the relationship, via appropriate translations, between the formalisms *SERS* and $SERS_{DB}$. Also, preservation of confluence is considered. Finally, we conclude.

By lack of space we only present here an extended abstract, and therefore proofs, auxiliary lemmas and standard definitions are only hinted or just omitted, but the interested reader will find full details in [4].

2 Simplified Expression Reduction Systems

We introduce the variable name based higher-order rewrite formalism *SERS*.

2.1 Metaterms and terms

Definition 1 (Signature). *Consider the denumerable and disjoint infinite sets:*

- $\mathcal{V} = \{x_1, x_2, x_3, \ldots\}$ *a set of* variables*, arbitrary variables are denoted* $x, y, \ldots$
- $\mathcal{B}_{\mathcal{MV}} = \{\alpha_1, \alpha_2, \alpha_3, \ldots\}$ *a set of* pre-bound o-metavariables *(o for object), denoted* $\alpha, \beta, \ldots$
- $\mathcal{F}_{\mathcal{MV}} = \{\widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha_3}, \ldots\}$ *a set of* pre-free o-metavariables*, denoted* $\widehat{\alpha}, \widehat{\beta}, \ldots$
- $\mathcal{T}_{\mathcal{MV}} = \{X_1, X_2, X_3, \ldots\}$ *a set of* t-metavariables *(t for term), denoted* $X, Y, Z, \ldots$
- $\mathcal{F} = \{f_1, f_2, f_3, \ldots\}$ *a set of* function symbols *equipped with a fixed (possibly zero) arity, denoted* $f, g, h, \ldots$
- $\mathcal{B} = \{\lambda_1, \lambda_2, \lambda_3, \ldots\}$ *a set of* binder symbols *equipped with a fixed (non-zero) arity, denoted* $\lambda, \mu, \nu, \xi, \ldots$

The union of $\mathcal{B}_{\mathcal{MV}}$ *and* $\mathcal{F}_{\mathcal{MV}}$ *is the set of o-metavariables of the signature. When speaking of metavariables without further qualifiers we refer to o and t-metavariables. Since all these alphabets are ordered, given any symbol* s *we shall denote* $\mathcal{O}(s)$ *its position in the corresponding alphabet.*

Definition 2 (Labels). *A* label *is a finite sequence of symbols of an alphabet. We shall use* $k, l, l_i, \ldots$ *to denote arbitrary labels and* ϵ *for the empty label. If* s *is a symbol and* l *is a label then the notation* $s \in l$ *means that the symbol* s appears in *the label* l*, and also, we use* sl *to denote the new label whose head is* s *and whose tail is* l*. Other notations are* $|l|$ *for the* length *of* l *(number of symbols in* l*) and* $\mathtt{at}(l, n)$ *for the* n*-th element of* l *assuming* $n \leq |l|$*. Also, if* s *occurs (at least once) in* l *then* $\mathtt{pos}(s, l)$ *denotes the* position of the first occurrence *of* s *in* l*. If* θ *is a function defined on the alphabet of a label* $l \equiv s_1 \ldots s_n$*, then* $\theta(l)$ *denotes the label* $\theta(s_1) \ldots \theta(s_n)$*. In the sequel, we may use a label as a set (e.g.* $S \cap l$ *denotes the intersection of a set* S *with the set containing the elements of* l*) if no confusion arises. A* simple label *is a label without repeated symbols.*

Definition 3 (Pre-metaterms). *The set of SERS* pre-metaterms[1]*, denoted* $\mathcal{PMT}$*, is defined by:*

$$M \quad ::= \quad \alpha \mid \widehat{\alpha} \mid X \mid f(M, \ldots, M) \mid \xi\alpha.(M, \ldots, M) \mid M[\alpha \leftarrow M]$$

Arities are supposed to be respected and we shall use $M, N, M_i, \ldots$ *to denote pre-metaterms. The symbol* $.[. \leftarrow .]$ *in the pre-metaterm* $M[\alpha \leftarrow M]$ *is called* metasubstitution operator*. The o-metavariable* α *in a pre-metaterm of the form* $\xi\alpha.(M, \ldots, M)$ *or* $M[\alpha \leftarrow M]$ *is referred to as the* formal parameter*. The set of binder symbols together with the metasubstitution operator are called* binder operators*, thus the metasubstitution operator is a binder operator (since it has binding power) but* not *a binder symbol since it is not an element of* $\mathcal{B}$*.*

[1] The main difference between *SERS* and *ERS* is that in the latter binders and metasubstitutions are defined on *multiple* o-metavariables. Indeed, pre-metaterms like $\xi\alpha_1 \ldots \alpha_k.(M_1, \ldots, M_m)$ and $M[\alpha_1 \ldots \alpha_k \leftarrow M_1, \ldots, M_k]$ are possible in *ERS*, with the underlying hypothesis that $\alpha_1 \ldots \alpha_k$ are all distinct and with the underlying semantics that $M[\alpha_1 \ldots \alpha_k \leftarrow M_1, \ldots, M_k]$ denotes usual (parallel) substitution. It is well known that multiple substitution can be simulated by simple substitution. Furthermore, there is also a notion of scope indicator in *ERS*, used to express in which arguments the variables are bound. Scope indicators shall not be considered in *SERS* since they do not seem to contribute to the expressive power of *ERS*.

A pre-metaterm M has an associated *tree*, denoted $tree(M)$, defined as expected. In the case of the metasubstitution operator we have: if T_1, T_2 are the trees of M_1, M_2, then the tree of $M_1[\alpha \leftarrow M_2]$ has root "sub", and sons "$[]\alpha$" (with son T_1) and T_2.

A position is a label over the alphabet $\mathbb{N}$. Given a pre-metaterm N appearing in M, the set of *occurrences* of N in M is the set of positions of $tree(M)$ where N occurs (positions in trees are defined as usual). The *parameter path* of an occurrence p in a tree T is the list containing all the (pre-bound) o-metavariables occuring in the path from p to the root of T.

The following definition introduces the set of *metaterms*, which are pre-metaterms that are *well-formed* in the sense that all the formal parameters appearing in the same path of a pre-metaterm must be different and all the metavariables in $\mathcal{B}_{\mathcal{MV}}$ only occur bound.

Definition 4 (Metaterms). *A pre-metaterm M is a metaterm, denoted by $\mathcal{WF}(M)$, iff the predicate $\mathcal{WF}_\epsilon(M)$ holds, where $\mathcal{WF}_l(M)$ is defined as follows:*

- $\mathcal{WF}_l(\alpha)$ *iff* $\alpha \in l$
- $\mathcal{WF}_l(\widehat{\alpha})$ *and* $\mathcal{WF}_l(X)$ *are always true*
- $\mathcal{WF}_l(f(M_1, \ldots, M_n))$ *iff for all* $1 \leq i \leq n$ *we have* $\mathcal{WF}_l(M_i)$
- $\mathcal{WF}_l(\xi\alpha.(M_1, \ldots, M_n))$ *iff* $\alpha \notin l$ *and for all* $1 \leq i \leq n$ *we have* $\mathcal{WF}_{\alpha l}(M_i)$
- $\mathcal{WF}_l(M_1[\alpha \leftarrow M_2])$ *iff* $\alpha \notin l$ *and* $\mathcal{WF}_l(M_2)$ *and* $\mathcal{WF}_{\alpha l}(M_1)$.

For example, $f(\xi\alpha.(X), \lambda\alpha.(Y))$, $f(\widehat{\beta}, \lambda\alpha.(Y))$ and $g(\lambda\alpha.(\xi\beta.(h)))$ are meta-terms, while the pre-metaterms $f(\alpha, \xi\alpha.(X))$ and $f(\widehat{\beta}, \lambda\alpha.(\xi\alpha.(X)))$ are not.

In the sequel, pre-bound (free) o-metavariables occurring in metaterms shall simply be referred to as bound (free) o-metavariables. As we shall see, metaterms are used to specify rewrite rules.

Definition 5 (Free metavariables of pre-metaterms). *Let M be a pre-metaterm, then $FMVar(M)$ denotes the* set of free metavariables *of M, which is defined as follows:*

$$FMVar(X) \stackrel{\text{def}}{=} \{X\} \qquad FMVar(\alpha) \stackrel{\text{def}}{=} \{\alpha\} \qquad FMVar(\widehat{\alpha}) \stackrel{\text{def}}{=} \{\widehat{\alpha}\}$$
$$FMVar(f(M_1, \ldots, M_n)) \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} FMVar(M_i)$$
$$FMVar(\xi\alpha.(M_1, \ldots, M_n)) \stackrel{\text{def}}{=} (\bigcup_{i=1}^{n} FMVar(M_i)) \setminus \{\alpha\}$$
$$FMVar(M_1[\alpha \leftarrow M_2]) \stackrel{\text{def}}{=} (FMVar(M_1) \setminus \{\alpha\}) \cup FMVar(M_2)$$

All metavariables which are not free are bound. We use $BMVar(M)$ to denote the bound metavariables of a metaterm M. Note that only o-metavariables may occur bound in a metaterm. We denote the set of metavariables of a metaterm or a pre-metaterm M by $MVar(M)$. Note that if M is a metaterm, then $FMVar(M)$ does not contain pre-bound o-metavariables.

Definition 6 (Terms and contexts). *The set of SERS* terms*, denoted $\mathcal{T}$, and contexts are defined by:*

$$\begin{array}{lll} \textit{Terms} & t & ::= x \mid f(t, \ldots, t) \mid \xi x.(t, \ldots, t) \\ \textit{Contexts} & C & ::= \Box \mid f(t, \ldots, C, \ldots, t) \mid \xi x.(t, \ldots, C, \ldots, t) \end{array}$$

where $\Box$ denotes a "hole". We shall use $s, t, t_i, \ldots$ for terms and C, D for contexts. We remark that in contrast to other formalisms dealing with higher-order rewriting, here the set of terms is not contained in the set of pre-metaterms since the set of variables and the set of o-metavariables are disjoint. The set of free (resp. bound) variables of a term t, denoted $FV(t)$ (resp $BV(t)$) are defined as usual.

With $C[t]$ we denote the term obtained by replacing the term t for the hole $\Box$ in the context C. Note that this operation may introduce variable capture. We define the label of a context *as a sequence of variables as follows:*

$$
\begin{array}{ll}
\texttt{label}(\Box) & \stackrel{\text{def}}{=} \epsilon \\
\texttt{label}(f(t_1,\ldots,C,\ldots,t_n)) & \stackrel{\text{def}}{=} \texttt{label}(C) \\
\texttt{label}(\xi x.(t_1,\ldots,C,\ldots,t_n)) & \stackrel{\text{def}}{=} \texttt{label}(C)x
\end{array}
$$

For example, the label of the context $C \equiv f(\lambda x.(z, \xi y.(h(y, \Box))))$ is the sequence yx. The label of a context is a notion analogous to that of a parameter path of an occurrence, but defined for terms instead of pre-metaterms and where the only occurrence considered is that of the hole.

Definition 7 ((Restricted) substitution of terms). *The* (restricted) substitution *of a term t for a variable x in a term s, denoted $s\{x \leftarrow t\}$, is defined:*

$$
\begin{array}{lll}
x\{x \leftarrow t\} & \stackrel{\text{def}}{=} t & \\
y\{x \leftarrow t\} & \stackrel{\text{def}}{=} y & \textit{if } x \not\equiv y \\
f(s_1,\ldots,s_n)\{x \leftarrow t\} & \stackrel{\text{def}}{=} f(s_1\{x \leftarrow t\},\ldots,s_n\{x \leftarrow t\}) & \\
\xi x.(s_1,\ldots,s_n)\{x \leftarrow t\} & \stackrel{\text{def}}{=} \xi x.(s_1,\ldots,s_n) & \\
\xi y.(s_1,\ldots,s_n)\{x \leftarrow t\} & \stackrel{\text{def}}{=} \xi y.(s_1\{x \leftarrow t\},\ldots,s_n\{x \leftarrow t\}) & \\
 & \quad \textit{if } x \not\equiv y, \textit{ and } (y \notin FV(t) \textit{ or } x \notin FV(s)) &
\end{array}
$$

Thus $.\{. \leftarrow .\}$ denotes the substitution operator on terms but it may *not* apply α-conversion (renaming of bound variables) in order to avoid unwanted variable captures. Therefore this notion of substitution is not defined for all terms (hence its name). When defining the notion of reduction relation on terms induced by rewrite rules we shall take α-conversion into consideration. We may define α-conversion on terms as the smallest reflexive, symmetric and transitive relation closed by contexts verifying the following equality:

$$(\alpha)\ \xi x.(s_1,\ldots,s_n) \equiv_\alpha \xi y.(s_1\{x \leftarrow y\},\ldots,s_n\{x \leftarrow y\}) \quad y \text{ not in } s_1,\ldots,s_n$$

Note that since y does not occur in $s_1,\ldots,s_n$ substitution is defined. We shall use $t \equiv_\alpha s$ to denote that the terms t and s are α-convertible. This conversion is sound in the sense that $t \equiv_\alpha s$ implies $FV(t) = FV(s)$.

The notion of α-conversion for terms has a symmetrical one for pre-metaterms which we call *v-equivalence* (v for variant). The intuitive meaning of two v-equivalent pre-metaterms is that they are able to receive the *same* set of potential "valuations" (c.f. Definition 10). Thus for example, as one would expect, $\lambda\alpha.(X) \neq_v \lambda\beta.(X)$ because when α and X are replaced by x and β is replaced by y, one obtains $\lambda x.(x)$ and $\lambda y.(x)$, which are not α-convertible. However, since pre-metaterms contain t-metavariables, the notion of v-equivalence is not straightforward as the notion of α-conversion in the case of terms.

Definition 8 (v-equivalence for pre-metaterms). *Given pre-metaterms M and N, we say that M is v-equivalent to N, iff $M =_v N$ where $=_v$ is the smallest reflexive, symmetric and transitive relation closed by metacontexts*[2] *verifying:*

$$
\begin{array}{lll}
(v1) & \xi\alpha.(P_1,\ldots,P_n) =_v \xi\beta.(P_1 \ll\alpha \leftarrow \beta\gg \ldots P_n \ll\alpha \leftarrow \beta\gg) \\
(v2) & P_1[\alpha \leftarrow P_0] & =_v P_1 \ll\alpha \leftarrow \beta\gg [\beta \leftarrow P_0]
\end{array}
$$

where β is a pre-bound o-metavariable which does not occur in $P_1,\ldots,P_n$ in $(v1)$ and does not occur in P_1 in $(v2)$, P_i does not contain t-metavariables for $1 \leq i \leq n$, and $P \ll\alpha \leftarrow Q \gg$ is the restricted substitution for pre-metaterms:

[2] Metacontexts are defined analogously to contexts. The notion of "label of a context" is extended to metacontexts as expected.

$$
\begin{array}{lll}
\alpha \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} Q & \\
\alpha' \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} \alpha' & \alpha \not\equiv \alpha' \\
\widehat{\alpha'} \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} \widehat{\alpha'} & \\
X \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} X & \\
f(M_1, \ldots, M_n) \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} f(M_1 \ll\alpha \leftarrow Q \gg, \ldots, M_n \ll\alpha \leftarrow Q \gg) & \\
(\xi\alpha.(M_1, \ldots, M_n)) \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} \xi\alpha.(M_1, \ldots, M_n) & \\
(\xi\alpha'.(M_1, \ldots, M_n)) \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} \xi\alpha'.(M_1 \ll\alpha \leftarrow Q \gg, \ldots, M_n \ll\alpha \leftarrow Q \gg) & \\
 & \quad \alpha \not\equiv \alpha',\ (\alpha' \notin FMVar(Q)\ \textit{or}\ \alpha \notin FMVar(P)) & \\
(M_1[\alpha \leftarrow M_2]) \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} M_1[\alpha \leftarrow M_2 \ll\alpha \leftarrow Q \gg] & \\
(M_1[\alpha' \leftarrow M_2]) \ll\alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} (M_1 \ll\alpha \leftarrow Q \gg)[\alpha' \leftarrow M_2 \ll\alpha \leftarrow Q \gg] & \\
 & \quad \alpha \not\equiv \alpha',\ (\alpha' \notin FMVar(Q)\ \textit{or}\ \alpha \notin FMVar(M_1)) &
\end{array}
$$

Example 1. $\lambda\alpha.(\alpha) =_v \lambda\beta.(\beta), \lambda\alpha.(f) =_v \lambda\beta.(f)$, but $\lambda\alpha.(X) \neq_v \lambda\beta.(X), \lambda\beta.(\lambda\alpha.(X)) \neq_v \lambda\alpha.(\lambda\beta.(X))$.

2.2 Reduction

Whereas the rewrite rules are specified by using metaterms, the reduction relation is defined on terms.

Definition 9 (*SERS* rewrite rule). *An SERS* rewrite rule *is a pair of meta-terms* (G, D) *(also written* $G \longrightarrow D$*) such that*

- *the first symbol in* G *is a function symbol or a binder symbol*
- $FMVar(D) \subseteq FMVar(G)$
- G *contains no occurrence of the metasubstitution operator*

Example 2. The λx-calculus [3, 19] is defined by the following *SERS* rewrite rules:

$$
\begin{array}{lll}
@(\lambda\alpha.(X), Z) & \longrightarrow_{Beta} & \Sigma(\sigma\alpha.(X), Z) \\
\Sigma(\sigma\alpha.(@(X, Y)), Z) & \longrightarrow_{App} & @(\Sigma(\sigma\alpha.(X), Z), \Sigma(\sigma\alpha.(Y), Z)) \\
\Sigma(\sigma\alpha.(\lambda\beta.(X)), Z) & \longrightarrow_{Lambda} & \lambda\beta.(\Sigma(\sigma\alpha.(X), Z)) \\
\Sigma(\sigma\alpha.(\alpha), Z) & \longrightarrow_{Var1} & Z \\
\Sigma(\sigma\alpha.(\widehat{\beta}), Z) & \longrightarrow_{Var2} & \widehat{\beta}
\end{array}
$$

Note that our formalism allows us to specify the Var2 rule as originally done in [19], while formalisms such as *CRS* force one to change this rule to a stronger one, called *gc*, written as $\Sigma(\sigma\alpha.(X), Z) \longrightarrow_{gc} X$, where the admissibility condition on valuations guarantees that if X/t is part of the valuation θ, then $\theta(\alpha)$ cannot be in $FV(t)$.

Example 3. The $\lambda\Delta$-calculus [18] is defined by the following *SERS* rewrite rules:

$$
\begin{array}{lll}
@(\lambda\alpha.(X), Z) & \longrightarrow_{Beta} & X[\alpha \leftarrow Z] \\
@(\Delta\alpha.(X), Z) & \longrightarrow_{\Delta 1} & \Delta\beta.(X[\alpha \leftarrow \lambda\gamma.(@(\beta, @(\gamma, Z)))]) \\
\Delta\alpha.(@(\alpha, X)) & \longrightarrow_{\Delta 2} & X \\
\Delta\alpha.(@(\alpha, (\Delta\beta.(@(\alpha, X))))) & \longrightarrow_{\Delta 3} & X
\end{array}
$$

Definition 10 (Valuation). *A* variable assignment *is a (partial) function θ_v from o-metavariables to variables with finite domain, such that for every pair of o-metavariables $\alpha, \widehat{\beta}$ we have $\theta_v\alpha \not\equiv \theta_v\widehat{\beta}$ (pre-bound and pre-free o-metavariables are assigned different values).*

A valuation *θ is a pair of (partial) functions (θ_v, θ_t) where θ_v is a variable assignment and θ_t maps t-metavariables to terms. We write $Dom(\theta)$ for $Dom(\theta_v) \cup Dom(\theta_t)$[3]. A valuation θ may be extended in a unique way to the set of pre-metaterms M such that $MVar(M) \subseteq Dom(\theta)$ as follows:*

$$\begin{array}{lll}
\overline{\theta}\alpha \stackrel{\text{def}}{=} \theta_v\alpha & \overline{\theta}f(M_1,\ldots,M_n) & \stackrel{\text{def}}{=} f(\overline{\theta}M_1,\ldots,\overline{\theta}M_n) \\
\overline{\theta}\widehat{\alpha} \stackrel{\text{def}}{=} \theta_v\widehat{\alpha} & \overline{\theta}(\xi\alpha.(M_1,\ldots,M_n)) & \stackrel{\text{def}}{=} \xi\theta_v\alpha.(\overline{\theta}M_1,\ldots,\overline{\theta}M_n) \\
\overline{\theta}X \stackrel{\text{def}}{=} \theta_t X & \overline{\theta}(M_1[\alpha \leftarrow M_2]) & \stackrel{\text{def}}{=} \overline{\theta}(M_1)\{\theta_v\alpha \leftarrow \overline{\theta}M_2\}
\end{array}$$

We shall not distinguish between θ and $\overline{\theta}$ if no ambiguities arise. Also, we sometimes write $\theta(M)$ thereby implicitly assuming that $MVar(M) \subseteq Dom(\theta)$.

Returning to the intuition behind *v-equivalence* the idea is that it can be translated into α-conversion in the sense that $M =_v N$ implies $\theta M \equiv_\alpha \theta N$ for any valuation θ such that θM and θN are defined. Indeed, coming back to Example 1 and taking $\theta = \{\alpha/x, \beta/y, X/x\}$, we have $\theta\lambda\alpha.(\alpha) \equiv \lambda x.(x) \equiv_\alpha \lambda y.(y) \equiv \theta\lambda\beta.(\beta)$, $\theta\lambda\alpha.(f) \equiv \lambda x.(f) \equiv_\alpha \lambda y.(f) \equiv \theta\lambda\beta.(f)$, $\theta\lambda\alpha.(X) \equiv \lambda x.(x) \not\equiv_\alpha \lambda y.(x) \equiv \theta\lambda\beta.(X)$, $\theta\lambda\beta.(\lambda\alpha.(X)) \equiv \lambda y.(\lambda x.(x)) \not\equiv_\alpha \lambda x.(\lambda y.(x)) \equiv \theta\lambda\alpha.(\lambda\beta.(X))$.

Definition 11 (Safe valuations). *Let $M \in \mathcal{PMT}$ and θ a valuation with $MVar(M) \subseteq Dom(\theta)$. We say that θ* is safe for *M if $\overline{\theta}M$ is defined. Likewise, if (G, D) is a rewrite rule, we say that θ is* safe for *(G, D) if $\overline{\theta}D$ is defined.*

Note that if the notion of substitution we are dealing with were not restricted then α-conversion could be required in order to apply a valuation to a pre-metaterm. Also, for any valuation θ and pre-metaterm M with $MVar(M) \subseteq Dom(\theta)$ that contains no occurrences of the metasubstitution operator θ is safe for M. Thus, we only ask θ to be safe for D (not G) in the previous definition.

The following condition is the classical notion of *admissibility* used in higher-order rewriting [21] to avoid inconsistencies in rewrite steps.

Definition 12 (Path condition for t-metavariables). *Let X be a t-meta-variable. Consider all the occurrences $p_1, \ldots, p_n$ of X in (G, D), and their respective parameter paths $l_1, \ldots, l_n$ in the trees corresponding to G and D. A valuation θ verifies* the path condition *for X in (G, D) if for every $x \in FV(\theta X)$, either ($\forall 1 \leq i \leq n$ we have $x \in \theta l_i$) or ($\forall 1 \leq i \leq n$ we have $x \notin \theta l_i$).*

This definition may be read as: one occurrence of $x \in FV(\theta X)$ with X in (G, D) is in the scope of some binding occurrence of x iff every occurrence of X in (G, D) is in the scope of a bound o-metavariable α with $\theta\alpha \equiv x$. For example, consider the *SERS* rule $\lambda\alpha.(\xi\beta.(X)) \longrightarrow \xi\beta.(X)$ and the valuations $\theta_1 = \{\alpha/x, \beta/y, X/z\}$ and $\theta_2 = \{\alpha/x, \beta/y, X/x\}$. Then θ_1 verifies the path condition for X, but θ_2 does not since when instantiating the rewrite rule with θ_2 the variable x shall occur both bound (on the LHS) and free (on the RHS).

Definition 13 (Admissible valuations). *A valuation θ is said to be* admissible for a rewrite rule *(G, D) iff*

- *θ is safe for (G, D)*
- *if α and β occur in (G, D) with $\alpha \not\equiv \beta$ then $\theta_v\alpha \not\equiv \theta_v\beta$*
- *θ verifies the path condition for every t-metavariable in (G, D)*

[3] As usual, $Dom(\psi)$ denotes the domain of the partial function ψ.

Note that an admissible valuation is safe by definition, but a safe valuation may not be admissible: consider the rule $\lambda\alpha.app(X,\alpha) \longrightarrow X$, the valuation $\theta = \{\alpha/x, X/x\}$ is trivially safe but is not admissible since the path condition is not verified: $x \in \theta(\alpha)$ but $x \notin \theta(\epsilon)$ (x occurs bound on the LHS and free on the RHS).

Now, there are two possible and equivalent ways to define reduction in a higher-order framework. One can either define reduction via a notion of substitution which makes explicit use of α-conversion, as it is usually done [10], or, as it is done here, reduction is explicitly defined as reduction modulo α-conversion and using a notion of restricted substitution which does not make use of α-conversion. We choose this second (and more involved) approach since we prefer to have a notion of reduction on terms in both formalisms (with names and de Bruijn indices), which is similar enough to make technical proofs work easily.

Definition 14 (Reduction on terms). *Let $\mathcal{R}$ be a set of SERS rewrite rules and s, t terms. We say that s* $\mathcal{R}$-reduces to *t, written $s \longrightarrow_{\mathcal{R}} t$, iff there exists a rewrite rule $(G, D) \in \mathcal{R}$, an admissible valuation θ for (G, D) and a context C such that $s \equiv_\alpha C[\theta G]$ and $t \equiv_\alpha C[\theta D]$.*

3 Simplified Expression Reduction Systems with Indices

We introduce de Bruijn indices based higher-order rewrite formalism $SERS_{DB}$.

3.1 De Bruijn metaterms and terms

A classical way to avoid α-conversion is to use de Bruijn index notation [7], where names of variables are replaced by natural numbers. When talking about a set N of de Bruijn indices we may refer to $\texttt{Names}(N)$ as the set of names of N given by the order on the set of variables $\mathcal{V}$ introduced in Section 2. Indeed, if $N = \{n_1, \ldots, n_m\}$, then $\texttt{Names}(N) = \{x_{n_1}, \ldots, x_{n_m}\}$.

In the sequel, in order to distinguish a concept defined for the *SERS* formalism from its corresponding version (if it exists) in the $SERS_{DB}$ formalism we may prefix it using the qualifying term "de Bruijn", eg. "de Bruijn metaterms".

Definition 15 (de Bruijn signature). *Consider the denumerable and disjoint infinite sets:*

- $\{\alpha_1, \alpha_2, \alpha_3, \ldots\}$ *a set of symbols called* binder indicators*, denoted $\alpha, \beta, \ldots$,*
- $\mathcal{I}_{\mathcal{MV}} = \{\widehat{\alpha_1}, \widehat{\alpha_2}, \ldots\}$ *a set of* i-metavariables *(i for index), denoted $\widehat{\alpha}, \widehat{\beta}, \ldots$,*
- $\mathcal{T}_{\mathcal{MV}} = \{X_l^1, X_l^2, X_l^3, \ldots\}$ *a set of* t-metavariables *(t for term), where l ranges over the set of labels built over binder indicators, denoted $X_l, Y_l, Z_l, \ldots$,*
- $\mathcal{F} = \{f_1, f_2, f_3, \ldots\}$ *a set of* function symbols *equipped with a fixed (possibly zero) arity, denoted $f, g, h, \ldots$,*
- $\mathcal{B} = \{\lambda_1, \lambda_2, \lambda_3, \ldots\}$ *a set of binder symbols equipped with a fixed (non-zero) arity, denoted $\lambda, \mu, \nu, \xi, \ldots$.*

We remark that the set of binder indicators is exactly the set of pre-bound o-metavariables introduced in Definition 1. The reason for using the same alphabet in both formalisms shall become clear in Section 4, but intuitively, we need a mechanism to annotate binding paths in the de Bruijn setting to distinguish metaterms like $\xi\beta.(\xi\alpha.(X))$ and $\xi\alpha.(\xi\beta.(X))$ appearing in the same rule when translated into an $SERS_{DB}$ system.

Definition 16 (de Bruijn pre-metaterms). *The set of* de Bruijn pre-meta-terms*, denoted $\mathcal{PMT}_{db}$, is defined by the following two-sorted grammar:*

$$\begin{array}{ll} \textit{metaindices} & I ::= 1 \mid \mathtt{S}(I) \mid \widehat{\alpha} \\ \textit{pre-metaterms} & A ::= I \mid X_l \mid f(A,\ldots,A) \mid \xi(A,\ldots,A) \mid A[\![A]\!] \end{array}$$

The symbol $.[\![.]\!]$ *in a pre-metaterm* $A[\![A]\!]$ *is called* de Bruijn metasubstitution operator. *The binder symbols together with the de Bruijn metasubstitution operator are called* binder operators, *and the same remark of Definition 3 applies.*

We shall use $A, B, A_i, \ldots$ to denote de Bruijn pre-metaterms and the convention that $\mathtt{S}^0(1) = 1$, $\mathtt{S}^0(\widehat{\alpha}) = \widehat{\alpha}$ and $\mathtt{S}^{j+1}(n) = \mathtt{S}(\mathtt{S}^j(n))$. As usually done for indices, we shall abbreviate $\mathtt{S}^{j-1}(1)$ as j.

Even if the formal mechanism used to translate pre-metaterms with names into pre-metaterms with de Bruijn indices will be given in Section 4, let us introduce intuitively some ideas in order to justify the syntax used for i-metavariables. In the formalism *SERS* there is a clear distinction between free and bound o-metavariables. This fact must also be reflected in the formalism $SERS_{DB}$, where bound o-metavariables are represented with indices and free o-metavariables are represented with i-metavariables (this distinction between free and bound variables is also used in some formalizations of λ-calculus [17]). However, free variables in $SERS_{DB}$ appear always in a binding context, so that a de Bruijn valuation of such kind of variables has to reflect the adjustment needed to represent the same variables but in a different context. This can be done by surrounding the i-metavariable by as many operators $\mathtt{S}$ as necessary. As an example consider the pre-metaterm $\xi\alpha.(\widehat{\beta})$. If we translate it to $\xi(\widehat{\beta})$, then a de Bruijn valuation like $\kappa = \{\widehat{\beta}/1\}$ binds the variable whereas this is completely impossible in the name formalism thanks to the conditions imposed on a name valuation (c.f. condition on variable assignments in Definition 10). Our solution is then to translate the pre-metaterm $\xi\alpha.(\widehat{\beta})$ by $\xi(\mathtt{S}(\widehat{\beta}))$ in such a way that there is no capture of variables since $\kappa(\xi(\mathtt{S}(\widehat{\beta})))$ is exactly $\xi(2)$. The solution adopted here is in some sense what is called *pre-cooking* in [9].

We use $MVar(A)$ (resp. $MVar_i(A)$ and $MVar_t(A)$) to denote the set of all metavariables (resp. i- and t-metavariables) of the de Bruijn pre-metaterm A.

As in the *SERS* formalism, we also need here a notion of well-formed pre-metaterm. The first motivation is to guarantee that labels of t-metavariables are correct w.r.t the context in which they appear, the second one is to ensure that indices like $\mathtt{S}^i(1)$ (resp. $\mathtt{S}^i(\widehat{\alpha})$) correspond to bound (resp. free) variables. Indeed, the pre-metaterms $\xi(X_{\alpha\beta})$, $\xi(\xi(4))$ and $\xi(\widehat{\alpha})$ shall not make sense for us, and hence shall not be considered well-formed.

Definition 17 (de Bruijn metaterms). *A pre-metaterm* $A \in \mathcal{PMT}_{db}$ *is said to be a* metaterm *iff the predicate* $\mathcal{WF}(A)$ *holds, where* $\mathcal{WF}(A)$ *iff* $\mathcal{WF}_\epsilon(A)$, *and* $\mathcal{WF}_l(A)$ *is defined as follows:*

- $\mathcal{WF}_l(\mathtt{S}^j(1))$ *iff* $j + 1 \leq |l|$
- $\mathcal{WF}_l(\mathtt{S}^j(\widehat{\alpha}))$ *iff* $j = |l|$
- $\mathcal{WF}_l(X_k)$ *iff* $l = k$ *and* l *is a simple label*
- $\mathcal{WF}_l(f(A_1,\ldots,A_n))$ *iff for all* $1 \leq i \leq n$ *we have* $\mathcal{WF}_l(A_i)$
- $\mathcal{WF}_l(\xi(A_1,\ldots,A_n))$ *iff there exists* $\alpha \notin l$ *such that for all* $1 \leq i \leq n$ *we have* $\mathcal{WF}_{\alpha l}(A_i)$
- $\mathcal{WF}_l(A_1[\![A_2]\!])$ *iff* $\mathcal{WF}_l(A_2)$ *and there exists* $\alpha \notin l$ *such that* $\mathcal{WF}_{\alpha l}(A_1)$

Therefore indices of the form $\mathtt{S}^j(1)$ may only occur in metaterms if they represent bound variables and well-formed metaindices of the form $\mathtt{S}^j(\widehat{\alpha})$ always represent a free variable. Note that when considering $\mathcal{WF}_l(M)$ and $\mathcal{WF}_l(A)$ it is Definitions 4 and 17 which are referenced, respectively.

Example 4. Pre-metaterms $\xi(X_\alpha, \lambda(Y_{\beta\alpha}, \mathtt{S}(1)))$, $f(\widehat{\beta}, \lambda(Y_\alpha, \mathtt{S}(\widehat{\alpha})))$, $g(\lambda(\xi(h)))$ are metaterms, while $f(\mathtt{S}(\widehat{\alpha}), \xi(X_\beta))$, $\lambda(\xi(X_{\alpha\alpha}))$, $f(\widehat{\beta}, \lambda(\xi(\mathtt{S}(\widehat{\beta}))))$ are not.

Definition 18 (de Bruijn terms and de Bruijn contexts). *The set of* de Bruijn terms*, denoted $\mathcal{T}_{db}$, and the set of de Bruijn contexts are defined by:*

$$\begin{array}{ll} \textit{de Bruijn indices} & n ::= 1 \mid \mathtt{S}(n) \\ \textit{de Bruijn terms} & a ::= n \mid f(a, \ldots, a) \mid \xi(a, \ldots, a) \\ \textit{de Bruijn contexts} & E ::= \Box \mid f(a, \ldots, E, \ldots, a) \mid \xi(a, \ldots, E, \ldots, a) \end{array}$$

We use $a, b, a_i, b_i, \ldots$ for de Bruijn terms and $E, F, \ldots$ for de Bruijn contexts. We may refer to the *binder path number* of a context, which is the number of binders between the $\Box$ and the root.

We use $FV(a)$ to denote the set of free variables (indices) in a; the result of substituting a term b for the index $n \geq 1$ in a term a is denoted $a\{\!\{n \leftarrow b\}\!\}$; the updating functions are denoted $\mathcal{U}_i^n(.)$ for $i \geq 0$ and $n \geq 1$. All these concepts are defined as usual.

Definition 19 (Free de Bruijn metavariables). *Let A be a de Bruijn pre-metaterm. The set of* free metavariables *of A, $FMVar(A)$, is defined as:*

$$\begin{array}{lll} FMVar(1) \stackrel{\text{def}}{=} \emptyset & & FMVar(f(A_1, \ldots, A_n)) \stackrel{\text{def}}{=} \bigcup_{i=1}^n FMVar(A_i) \\ FMVar(\mathtt{S}(I)) \stackrel{\text{def}}{=} FMVar(I) & & FMVar(\xi(A_1, \ldots, A_n)) \stackrel{\text{def}}{=} \bigcup_{i=1}^n FMVar(A_i) \\ FMVar(\widehat{\alpha}) \stackrel{\text{def}}{=} \{\widehat{\alpha}\} & & FMVar(A_1[\![A_2]\!]) \stackrel{\text{def}}{=} FMVar(A_1) \cup FMVar(A_2) \\ FMVar(X_l) \stackrel{\text{def}}{=} \{X_l\} & & \end{array}$$

Note that this definition also applies to de Bruijn metaterms. The set of names of free metavariables of A is the set of free metavariables of A where each X_l is replaced simply by X. This notion will be used in Definition 20.

3.2 Reduction

We define rewrite rules, valuations, their validity, and reduction in $SERS_{DB}$.

Definition 20 (de Bruijn rewrite rule). *A* de Bruijn rewrite rule *is a pair of de Bruijn metaterms (L, R) (also written $L \longrightarrow R$) such that*

- *the first symbol in L is a function symbol or a binder symbol*
- *the set of names of $FMVar(R)$ is included in the set of names of $FMVar(L)$*
- *the metasubstitution operator does not occur in L*

Definition 21 (de Bruijn valuation). *A* de Bruijn valuation *κ is a pair of (partial) functions (κ_i, κ_t) where κ_i is a function from i-metavariables to integers, and κ_t is a function from t-metavariables to de Bruijn terms. We denote by $Dom(\kappa)$ the set $Dom(\kappa_i) \cup Dom(\kappa_t)$. A valuation κ determines in a unique way a function $\overline{\kappa}$ from the set of pre-metaterms A with $FMVar(A) \subseteq Dom(\kappa)$ to the set of terms as follows:*

$$\begin{array}{lll} \overline{\kappa}1 \stackrel{\text{def}}{=} 1 & & \overline{\kappa}f(A_1, \ldots, A_n) \stackrel{\text{def}}{=} f(\overline{\kappa}A_1, \ldots, \overline{\kappa}A_n) \\ \overline{\kappa}\mathtt{S}(I) \stackrel{\text{def}}{=} \mathtt{S}(\overline{\kappa}I) & & \overline{\kappa}\xi(A_1, \ldots, A_n) \stackrel{\text{def}}{=} \xi(\overline{\kappa}A_1, \ldots, \overline{\kappa}A_n) \\ \overline{\kappa}\widehat{\alpha} \stackrel{\text{def}}{=} \kappa_i\widehat{\alpha} & & \overline{\kappa}(A_1[\![A_2]\!]) \stackrel{\text{def}}{=} \overline{\kappa}(A_1)\{\!\{1 \leftarrow \overline{\kappa}A_2\}\!\} \\ \overline{\kappa}X_l \stackrel{\text{def}}{=} \kappa_t X_l & & \end{array}$$

Note that in the above definition the substitution operator $.\{\!\{. \leftarrow .\}\!\}$ refers to the usual substitution defined on terms with de Bruijn indices.

We now introduce the notion of *value function* which is used to give semantics to metavariables with labels in the $SERS_{DB}$ formalism. The goal pursued by the labels of metavariables is that of incorporating "context" information as a defining part of a metavariable. As a consequence, we must verify that the terms substituted for every occurrence of a fixed metavariable coincide "modulo" their corresponding context. Dealing with such notion of "coherence" of substitutions in a de Bruijn formalism is also present in other formalisms but in a more restricted form. Thus for example, as mentioned before, a pre-cooking function is used in [9] in order to avoid variable capture in the higher-order unification procedure. In *XRS* [16] the notions of binding arity and pseudo-binding arity are introduced in order to take into account the parameter path of the different occurrences of t-metavariables appearing in a rewrite rule. Our notion of "coherence" is implemented with *valid valuations* (cf. Definition 23) and it turns out to be more general than the solutions proposed in [9] and [16].

Definition 22 (Value function). *Let $a \in \mathcal{T}_{db}$ and l be a label of binder indicators. Then we define the value function $Value(l, a)$ as $Value^0(l, a)$ where*

$$Value^i(l, n) \stackrel{\text{def}}{=} \begin{cases} n & \text{if } n \leq i \\ \mathtt{at}(l, n-i) & \text{if } 0 < n - i \leq |l| \\ x_{n-i-|l|} & \text{if } n - i > |l| \end{cases}$$

$$Value^i(l, f(a_1, \ldots, a_n)) \stackrel{\text{def}}{=} f(Value^i(l, a_1), \ldots Value^i(l, a_n))$$

$$Value^i(l, \xi(a_1, \ldots, a_n)) \stackrel{\text{def}}{=} \xi(Value^{i+1}(l, a_1), \ldots, Value^{i+1}(l, a_n))$$

It is worth noting that $Value^i(l, n)$ may give three different kinds of results. This is just a technical trick to make easier later proofs. Indeed, we have for example $Value(\alpha\beta, \xi(f(3, 1))) \equiv \xi(f(\beta, 1)) \equiv Value(\beta\alpha, \xi(f(2, 1)))$ and $Value(\epsilon, f(\xi(1), \lambda(2))) \equiv f(\xi(1), \lambda(x_1)) \not\equiv f(\xi(1), \lambda(\alpha)) \equiv Value(\alpha, f(\xi(1), \lambda(2)))$. Thus the function $Value(l, a)$ interprets the de Bruijn term a in an l-context: bound indices are left untouched, free indices referring to the l-context are replaced by the corresponding binder indicator and the remaining free indices are replaced by their corresponding variable names.

In order to introduce the notion of valid de Bruijn valuations let us consider the following rule:

$$\xi\alpha.(\xi\beta.(X)) \longrightarrow_r \xi\beta.(\xi\alpha.(X))$$

Even if translation of rewrite rules into de Bruijn rewrite rules has not been defined yet (Section 4), one may guess that a reasonable translation would be the following rule (called r_{DB}):

$$\xi(\xi(X_{\beta\alpha})) \longrightarrow_{r_{DB}} \xi(\xi(X_{\alpha\beta}))$$

which indicates that β (resp. α) is the first bound occurrence in the LHS (resp. RHS) while α (resp. β) is the second bound occurrence in the LHS (resp. RHS). Now, if X is instantiated by x, α by x and β by y in the *SERS* system, then we have a r-reduction step $\xi x.(\xi y.(x)) \longrightarrow \xi y.(\xi x.(x))$. However, to reflect this fact in the corresponding $SERS_{DB}$ system we need to instantiate $X_{\beta\alpha}$ by 2 and $X_{\alpha\beta}$ by 1, thus obtaining a r_{DB}-reduction step $\xi(\xi(2)) \longrightarrow \xi(\xi(1))$. This clearly shows that de Bruijn t-metavariables having the same name but different label cannot be instantiated arbitrarily as they have to reflect the renaming of variables which is indicated by their labels. This is exactly the role of the property of validity:

Definition 23 (Valid de Bruijn valuation). *A de Bruijn valuation κ is said to be* valid *if for every pair of t-metavariables X_l and $X_{l'}$ in $Dom(\kappa)$ we have $Value(l, \kappa X_l) \equiv Value(l', \kappa X_{l'})$. Likewise, we say that a de Bruijn valuation κ is* valid for a rewrite rule (L, R) *if for every pair of t-metavariables X_l and $X_{l'}$ in (L, R) we have $Value(l, \kappa X_l) \equiv Value(l', \kappa X_{l'})$.*

It is interesting to note that there is no concept analogous to safeness (cf. Definition 11) as used for named *SERS* due to the use of de Bruijn indices. Also, the last condition in the definition of an admissible valuation (cf. Definition 13) is subsumed by the above Definition 23 in the setting of $SERS_{DB}$.

Example 5. Returning to the example just after Definition 22 we have that $\kappa = \{X_{\beta\alpha}/2, X_{\alpha\beta}/1\}$ is valid since $Value(\beta\alpha, 2) \equiv \alpha \equiv Value(\alpha\beta, 1)$.

Another interesting example is the well-known η-contraction rule $\lambda x.@(X, x) \longrightarrow X$ if $x \notin FV(X)$. It can be expressed in the *SERS* formalism as the rule (η_n) $\lambda\alpha.@(X, \alpha) \longrightarrow X$, and in the $SERS_{DB}$ formalism as the rule (η_{db}) $\lambda(@(X_\alpha, 1)) \longrightarrow X_\epsilon$.

Remark that this kind of rule cannot be expressed in the *XRS* formalism [16] since it does not verify the binding arity condition. Our formalism allows us to write rules like η_{db} because valid valuations will test for coherence of values. Indeed, an admissible valuation for η_n is a valuation θ such that θX does not contain a free occurrence of $\theta(\alpha)$. This is exactly the condition used in any usual formalization of the η-rule.

Definition 24 (Reduction on de Bruijn terms). *Let $\mathcal{R}$ be a set of de Bruijn rules and a, b de Bruijn terms. We say that a $\mathcal{R}$-*reduces to *b, written $a \longrightarrow_{\mathcal{R}} b$, iff there is a de Bruijn rule $(L, R) \in \mathcal{R}$ and a de Bruijn valuation κ valid for (L, R) such that $a \equiv E[\kappa L]$ and $b \equiv E[\kappa R]$, where E is a de Bruijn context.*

Thus, the term $\lambda(app(\lambda(app(1, 3)), 1))$ rewrites by the η_{db} rule to $\lambda(app(1, 2))$, using the (valid) valuation $\kappa = \{X_\alpha/\lambda(app(1, 3), X_\epsilon/\lambda(app(1, 2))\}$.

4 Relating *SERS* and $SERS_{DB}$

In this section we show how reduction in the *SERS* formalism may be simulated in the $SERS_{DB}$ formalism and vice-versa.

Definition 25 (From terms (and contexts) to de Bruijn terms (and contexts)). *The translation of a term t, denoted $T(t)$, is defined as $T_\epsilon(t)$ where*

$$\begin{array}{ll} T_k(x) & \stackrel{\text{def}}{=} \begin{cases} \mathtt{pos}(x, k) & \text{if } x \in k \\ \mathcal{O}(x) + |k| & \text{if } x \notin k \end{cases} \\ T_k(f(t_1, \ldots, t_n)) & \stackrel{\text{def}}{=} f(T_k(t_1), \ldots, T_k(t_n)) \\ T_k(\xi x.(t_1, \ldots, t_n)) & \stackrel{\text{def}}{=} \xi(T_{xk}(t_1), \ldots, T_{xk}(t_n)) \end{array}$$

The translation of a context, denoted $T(C)$, adds the clause $T_k(\Box) \stackrel{\text{def}}{=} \Box$.

Definition 26 (From pre-metaterms to de Bruijn pre-metaterms). *The translation of a pre-metaterm M, denoted $T(M)$, is defined as $T_\epsilon(M)$ where:*

$$\begin{array}{llll} T_k(\alpha) & \stackrel{\text{def}}{=} \mathtt{pos}(\alpha, k), \text{ if } \alpha \in k & T_k(f(M_1, \ldots, M_n)) & \stackrel{\text{def}}{=} f(T_k(M_1), \ldots, T_k(M_n)) \\ T_k(\widehat{\alpha}) & \stackrel{\text{def}}{=} \mathtt{S}^{|k|}(\widehat{\alpha}) & T_k(\xi\alpha.(M_1, \ldots, M_n)) & \stackrel{\text{def}}{=} \xi(T_{\alpha k}(M_1), \ldots, T_{\alpha k}(M_n)) \\ T_k(X) & \stackrel{\text{def}}{=} X_k & T_k(M_1[\alpha \leftarrow M_2]) & \stackrel{\text{def}}{=} T_{\alpha k}(M_1)[\![T_k(M_2)]\!] \end{array}$$

Note that if M is a metaterm, then $T(M)$ will be a de Bruijn metaterm and only have t-metavariables with simple labels. Note also that, for some pre-metaterms, such as $\xi\alpha.(\beta)$, the translation $T(.)$ is not defined.

Lemma 1 (T preserves well-formedness). *If M is a metaterm, then $T(M)$ is a de Bruijn metaterm.*

Definition 27 (From *SERS* rewrite rules to $SERS_{DB}$ rewrite rules). *Let (G, D) be a rewrite rule in the SERS formalism. Then $T(G, D)$ denotes the translation of the rewrite rule, defined as $(T(G), T(D))$.*

As an immediate consequence of Lemma 1 and Definition 27, if (G, D) is an *SERS* rewrite rule, then $T(G, D)$ is an $SERS_{DB}$ rewrite rule.

Example 6. Following Example 2, the specification of $\lambda\mathtt{x}$ in the $SERS_{DB}$ formalism is given below.

$$\begin{array}{lll}
@(\lambda(X_\alpha), Z_\epsilon) & \longrightarrow & \Sigma(\sigma(X_\alpha), Z_\epsilon) \\
\Sigma(\sigma(@(X_\alpha, Y_\alpha)), Z_\epsilon) & \longrightarrow & @(\Sigma(\sigma(X_\alpha), Z_\epsilon), \Sigma(\sigma(Y_\alpha), Z_\epsilon)) \\
\Sigma(\sigma(\lambda(X_{\beta\alpha})), Z_\epsilon) & \longrightarrow & \lambda(\Sigma(\sigma(X_{\alpha\beta}), Z_\beta)) \\
\Sigma(\sigma(1), Z_\epsilon) & \longrightarrow & Z_\epsilon \\
\Sigma(\sigma(\mathtt{S}(\widehat{\beta})), Z_\epsilon) & \longrightarrow & \widehat{\beta}
\end{array}$$

The rule $\Sigma(\sigma(\lambda(X_{\beta\alpha})), Z_\epsilon) \longrightarrow \lambda(\Sigma(\sigma(X_{\alpha\beta}), Z_\beta))$ is interesting since it illustrates the use of binder commutation from $X_{\beta\alpha}$ to $X_{\alpha\beta}$ and shows how some index adjustment shall be necessary when going from Z_ϵ to Z_β.

Example 7. The translation of the $\lambda\Delta$-calculus (Example 3) yields the following rewrite rules in the $SERS_{DB}$ formalism

$$\begin{array}{lll}
@(\lambda(X_\alpha), Z_\epsilon) & \longrightarrow & X_\alpha[\![Z_\epsilon]\!] \\
@(\Delta(X_\alpha), Z_\epsilon) & \longrightarrow & \Delta(X_{\alpha\beta}[\![\lambda(@(\mathtt{S}(1), @(1, Z_{\gamma\beta})))]\!]) \\
\Delta(@(1, X_\alpha)) & \longrightarrow & X_\epsilon \\
\Delta(@(1, (\Delta(@(\mathtt{S}(1), X_{\beta\alpha}))))) & \longrightarrow & X_\epsilon
\end{array}$$

We remark that the translation of Δ_1, Δ_2 and Δ_3 would not be possible in XRS [16].

Proposition 1 (Simulating *SERS* reduction via $SERS_{DB}$ reduction). *Suppose $s \longrightarrow t$ in the SERS formalism using the rewrite rule (G, D). Then we have $T(s) \longrightarrow T(t)$ in the $SERS_{DB}$ formalism using the rule $T(G, D)$.*

We now consider how reduction in $SERS_{DB}$ may be simulated in *SERS*.

Definition 28 (From de Bruijn terms (contexts) to terms (contexts)). *We define the translation of $a \in \mathcal{T}_{db}$, denoted $U(a)$, as $U_\epsilon^{\mathtt{Names}(FV(a))}(a)$ where, for every finite set of variables S, and label of variables k, $U_k^S(a)$ is defined by:*

$$\begin{array}{lll}
U_k^S(n) & \stackrel{\text{def}}{=} & \begin{cases} \mathtt{at}(k, n) & \text{if } n \leq |k| \\ x_{n-|k|} & \text{if } n > |k| \text{ and } x_{n-|k|} \in S \end{cases} \\
U_k^S(f(a_1, \ldots, a_n)) & \stackrel{\text{def}}{=} & f(U_k^S(a_1), \ldots, U_k^S(a_n)) \\
U_k^S(\xi(a_1, \ldots, a_n)) & \stackrel{\text{def}}{=} & \xi x.(U_{xk}^S(a_1), \ldots, U_{xk}^S(a_n)) \quad \text{for any } x \notin k \cup S
\end{array}$$

The translation of a de Bruijn context E, denoted $U(E)$, is defined as above but adding the clause $U_k^S(\Box) \stackrel{\text{def}}{=} \Box$. We remark that we can always choose $x \notin k \cup S$ since both k and S are finite.

Note that $U(.)$ is not a function in the sense that the choice of bound variables is non-deterministic. However, if t and t' belong both to $U(a)$, then $t \equiv_\alpha t'$. Thus, $U(.)$ can be seen as a function from de Bruijn terms to α-equivalence classes.

Definition 29 (From de Bruijn pre-metaterms to pre-metaterms). *The translation of a de Bruijn pre-metaterm A, denoted $U(A)$, is defined as $U_\epsilon(A)$, where $U_l(A)$ is defined as follows:*

$$
\begin{array}{lll}
U_l(\mathtt{S}^i(1)) & \stackrel{\text{def}}{=} \mathtt{at}(l, i+1) & \text{if } i+1 \leq |l| \\
U_l(\mathtt{S}^{|l|}(\widehat{\alpha})) & \stackrel{\text{def}}{=} \widehat{\alpha} & \\
U_l(X_l) & \stackrel{\text{def}}{=} X & \\
U_l(f(A_1, \ldots, A_n)) & \stackrel{\text{def}}{=} f(U_l(A_1), \ldots, U_l(A_n)) & \\
U_l(\xi(A_1, \ldots, A_n)) & \stackrel{\text{def}}{=} \xi\alpha.(U_{\alpha l}(A_1), \ldots, U_{\alpha l}(A_n)) & \\
 & \quad \text{if } 1 \leq i \leq n \quad \mathcal{WF}_{\alpha l}(A_i) \text{ for some } \alpha \notin l & \\
U_l(A_1[\![A_2]\!]) & \stackrel{\text{def}}{=} U_{\alpha l}(A_1)[\alpha \leftarrow U_l(A_2)] & \\
 & \quad \text{if } \mathcal{WF}_{\alpha l}(A_1) \text{ for some } \alpha \notin l &
\end{array}
$$

As in Definition 28 we remark that the translation of a de Bruijn pre-metaterm is not a function since it depends on the choice of the names for o-metavariables. Indeed, two different pre-metaterms obtained by this translation will be v-equivalent. Also, for some de Bruijn pre-metaterms such as $\xi(2)$, the translation may not be defined. However, it is defined on de Bruijn metaterms.

Definition 30 (From $SERS_{DB}$ rewrite rules to $SERS$ rewrite rules). *Let (L, R) be a de Bruijn rewrite rule then its translation, denoted $U(L, R)$, is the pair of metaterms $(U_\epsilon(L), U_\epsilon(R))$.*

Note that if $\mathcal{WF}_l(A)$ holds then $U_l(A)$ is also a named metaterm, that is, $\mathcal{WF}_l(U_l(A))$ also holds. Therefore, by Definition 9 the translation of a de Bruijn rule is a rule in *SERS*. As mentioned above, if a de Bruijn pre-metaterm A is not a de Bruijn metaterm then $U_l(A)$ may not be defined.

Example 8. Consider the rule $@(\Delta(X_\alpha), Z_\epsilon) \longrightarrow \Delta(X_{\alpha\beta}[\![\lambda(@(\mathtt{S}(1), @(1, Z_{\gamma\beta})))]\!])$ from Example 7. The translation in Definition 29 yields the rule $@(\Delta\alpha.(X), Z) \longrightarrow \Delta\beta.(X[\alpha \leftarrow \lambda\gamma.(@(\beta, @(\gamma, Z)))])$ and the translation in Definition 29 on the rule $\Sigma(\sigma(\mathtt{S}(\widehat{\beta})), Z_\epsilon) \longrightarrow \widehat{\beta}$ yields $\Sigma(\sigma\gamma.(\widehat{\beta}), Z) \longrightarrow \widehat{\beta}$ for some bound metavariable γ.

Proposition 2 (Simulating $SERS_{DB}$ reduction via $SERS$ reduction). *Suppose $a \longrightarrow b$ in the $SERS_{DB}$ formalism using rewrite rule (L, R). Then we have $U(a) \longrightarrow U(b)$ in the SERS formalism using rule $U(L, R)$.*

As regards the relationship between the translations over pre-metaterms and terms introduced above we may obtain two results stating, respectively, that given a metaterm M then $U(T(M))$ is v-equivalent to M and that given a de Bruijn metaterm A then $T(U(A))$ is identical to A. These results are used to show that confluence is preserved when translating in both directions.

Theorem 1 (Preservation of confluence).

1. *If $\mathcal{R}$ is a confluent SERS then $T(\mathcal{R})$ is a confluent $SERS_{DB}$.*
2. *If $\mathcal{R}$ is a confluent $SERS_{DB}$ then $U(\mathcal{R})$ is a confluent SERS.*

5 Conclusions

We have proposed a formalism for higher-order rewriting with de Bruijn notation and we have shown that rewriting with names and rewriting with indices are semantically equivalent. We have given formal translations from one formalism into the other which can be viewed as an *interface* in programming languages based on higher-order rewrite systems. This work fills the gap between classical presentations of higher-order rewriting with names existing in the literature and first-order

presentations of higher-order rewriting such as [16]. Moreover, it explicitly suggests that XRS are not sufficient to express *an arbitrary* higher-order rewrite system.

Further ongoing work uses the formalism presented here to propose a tool for implementing higher-order rewrite systems via first-order ones. This tool would incorporate not only de Bruijn notation but also explicit substitutions in a very general form.

References

1. H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1984. Revised edition.
2. R. Bloo. *Preservation of Termination for Explicit Substitution*. PhD thesis, Eindhoven University, 1997.
3. R. Bloo and K. Rose. Combinatory reduction systems with explicit substitution that preserve strong normalisation. In RTA, LNCS 1103, pages 169-183. 1996.
4. E. Bonelli, D. Kesner, and A. Ríos. A de Bruijn notation for higher-order rewriting, 2000. Available as `ftp://ftp.lri.fr/LRI/articles/kesner/dBhor.ps.gz`.
5. P.-L. Curien. *Categorical combinators, sequential algorithms and functional programming*. Progress in Theoretical Computer Science. Birkhaüser, 1993. Second edition.
6. P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, march 1996.
7. 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.
8. G. Dowek. *La part du calcul*. Univesité de Paris VII, 1999. Thèse d'Habilitation à diriger des recherches.
9. G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. In LICS, 1995.
10. R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus*. London Mathematical Society. 1980.
11. F. Kamareddine and A. Ríos. Bridging de bruijn indices and variable names in explicit substitutions calculi. *Logic Journal of the Interest Group of Pure and Applied Logic (IGPL)*, 6(6):843–874, 1998.
12. Z. Khasidashvili. Expression reduction systems. In *Proceedings of I. Vekua Institute of Applied Mathematics*, volume 36, Tbilisi, 1990.
13. Z. Khasidashvili and V. van Oostrom. Context-sensitive Conditional Expression Reduction Systems. In ENTCS, Vol.2, Proceedings of the Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation (SEGRAGRA'95), Volterra, 1995.
14. J. W. Klop. *Combinatory Reduction Systems*, vol. 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, 1980. PhD Thesis.
15. T. Nipkow. Higher order critical pairs. In LICS, pages 342–349, 1991.
16. B. Pagano. *Des Calculs de Susbtitution Explicite et leur application à la compilation des langages fonctionnels*. PhD thesis, Université Paris VI, 1998.
17. R. Pollack. Closure under alpha-conversion. In TYPES, LNCS 806. 1993.
18. J. Rehof and M. H. Sørensen. The λ_Δ calculus. In TACS, LNCS 789, pages 516–542. 1994.
19. K. Rose. Explicit cyclic substitutions. In CTRS, LNCS 656, pages 36–50, 1992.
20. V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: the higher-order case. In LFCS, LNCS 813, pages 379–392, 1994.
21. F. van Raamsdonk. *Confluence and Normalization for higher-Order Rewriting*. PhD thesis, Amsterdam University, The Netherlands, 1996.
22. D. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1993.