From Higher-Order to First-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, Argentina.
Email:{ebonelli@dc.uba.ar,rios@dc.uba.ar}
[2] LRI (CNRS URA 410) - Bât 490, Université de Paris-Sud, 91405 Orsay Cedex, France. Email:kesner@lri.fr

Abstract. We show how higher-order rewriting may be encoded into first-order rewriting modulo an equational theory $\mathcal{E}$. We obtain a characterization of the class of higher-order rewriting systems which can be encoded by first-order rewriting modulo an empty theory (that is, $\mathcal{E} = \emptyset$). This class includes of course the λ-calculus. Our technique does not rely on a particular substitution calculus but on a set of abstract properties to be verified by the substitution calculus used in the translation.

1 Introduction

Higher-order substitution is a complex operation that consists in the replacement of variables by terms in the context of languages having variable bindings. These bound variables can be annotated by de Bruijn indices so that the renaming operation (α-conversion) which is necessary to carry out higher-order substitution can be avoided. However, substitution is still a complicated notion, which cannot be expressed by simple replacement (a.k.a. grafting) of variables as is done in first-order theories. To solve this problem, many researchers became interested in the formalization of higher-order substitution by *explicit substitutions*, so that higher-order systems/formalisms could be expressible in first-order systems/formalisms: the notion of variable binding is dropped because substitution becomes replacement. A well-known example of the combination of de Bruijn indices and explicit substitutions is the formulation of different first-order calculi for the λ-calculus $[1, 4, 17, 15, 6, 22]$, which is the paradigmatic example of a higher-order (term) rewriting system. Other examples are the translations of higher-order unification to first-order unification modulo [11], higher-order logic to first-order logic modulo [12], higher-order theorem proving to first-order theorem proving modulo [9], etc.

All these translations have a theoretical interest because the expressive power of higher and first-order formalisms is put in evidence, but another practical issue arises, that of the possibility of transferring results developed in the first-order framework to the higher-order one.

The goal of this paper is to give a translation of higher-order rewrite systems (*HORS*) to a first-order formalism. As a consequence, properties and techniques developed for first-order rewriting could be exported to the higher-order formalism. For example, techniques concerning confluence, termination, completion, evaluation strategies, etc. This is very interesting since, on one hand it is still not clear how to transfer techniques such as dependency pairs [2], semantic labelling [26] or completion [3] to the higher-order framework, on the other hand, termination techniques such as RPO for higher-order systems [14] turn out to be much more complicated than their respective first-order versions [7, 16].

The main difficulty encountered in our translation can be intuitively explained by the fact that in higher-order rewriting a metavariable occurring on the right-hand side in a higher-order rewrite rule may occur in a different *binding context* on the left-hand side: for example, in the usual presentation of the extensional rule for functional types in λ-calculus (η):$\lambda\alpha.app(X,\alpha) \longrightarrow X$, the occurrence of X on the right-hand side of the rule, which does not appear in any binding context, is related to the occurrence of X on the left-hand side, which appears inside a binding context. The immediate consequence of this fact is that the first-order translation of the rule (η) cannot be defined as the naive translation taking both sides of the rule independently. This would give the first-order rule $\lambda(app(X,\underline{1})) \longrightarrow X$, which does not reflect the intended semantics and hence the translation would be incorrect.

As mentioned before, the need for (α)-conversion immediately disappears when de Bruijn notation is considered. Following the example recently introduced, one can express the (η)-rule in a higher-order de Bruijn setting, such as for example the $SERS_{DB}$ (Simplified Expression Reduction Systems) formalism [5], by the rule (η_{dB}) :$\lambda(app(X_\alpha,\underline{1})) \longrightarrow X_\epsilon$. The notation used to translate the metavariable X into the de Bruijn formalism enforces the fact that the occurrence of X on the right-hand side of the rule (η) does not appear in a binding context, so it is translated as X_ϵ where ϵ represents an empty binding path, while the occurrence of X on the left-hand side of the rule appears inside a binding context, so it is translated as X_α, where α represents a binding path of length 1 surrounding the metavariable X.

Now, the term $\lambda(app(\underline{3},\underline{1}))$ reduces to $\underline{2}$ via the (η_{dB}) rule. In an explicit substitution setting, we have the alternative formulation:

$$(\eta_{fo}) : \lambda(app(X[\uparrow],\underline{1})) \longrightarrow X$$

However, in order for the metaterm $X[\uparrow]$ to match the subterm $\underline{3}$ first-order matching no longer suffices: we need $\mathcal{E}$-matching, that is, matching modulo an equational theory $\mathcal{E}$. For an appropriate substitution calculus $\mathcal{E}$ we would need to solve the equation $\underline{3} \stackrel{?}{=}_{\mathcal{E}} X[\uparrow]$. Equivalently, we could make use of the theory of conditional rewriting:

$$\lambda(app(Y,\underline{1})) \longrightarrow X \text{ where } Y =_{\mathcal{E}} X[\uparrow]$$

Another less evident example is given by a commutation rule as for example

$$(C) : Implies(\exists\alpha.\forall\beta.X, \forall\beta.\exists\alpha.X) \longrightarrow true$$

which expresses that the formula appearing as the first argument of the *Implies* function symbol implies the one in the second argument. The naive translation to first-order $Implies(\exists(\forall(X)), \forall(\exists(X))) \longrightarrow true$ is evidently not correct, so that we take its translation in the de Bruijn higher-order formalism $SERS_{DB}$ and then translate it to first-order via the conversion presented in this work obtaining:

$$(C_{fo}) : Implies(\exists(\forall(X)), \forall(\exists(X[\underline{2} \cdot \underline{1} \cdot id]))) \longrightarrow true$$

Now, the rule (C_{fo}) has exactly the same semantics as the original higher-order rule (C). This difficulty does not seem to appear in other problems dealing with translations from higher-order to first-order recently mentioned.

In this work we shall see how higher-order rewriting may be systematically reduced to first-order rewriting modulo an equational theory $\mathcal{E}$. To do this, we choose to work with Expression Reduction Systems [18], and in particular, with de Bruijn based $SERS_{DB}$ as defined in [5] which facilitates the translation of higher-order systems to first-order ones. However, we claim that the same translation could be applied to other higher-order rewriting formalisms existing in the literature. We obtain a characterization of the class of $SERS_{DB}$ (including λ-calculus) for which a translation to a full ($\mathcal{E} = \emptyset$) first-order rewrite system exists. Thus the result mentioned above on the λ-calculus becomes a particular case of our work.

To the best of our knowledge there is just one formalism, called *XRS* [24], which studies higher-order rewrite formalisms based on de Bruijn index notation and explicit substitutions. The formalism *XRS*, which is a first-order formalism, is presented as a generalization of the first-order $\sigma_{\Uparrow}$-calculus [13] to higher-order rewriting and not as a first-order formulation of higher-order rewriting. As a consequence, many well-known higher-order rewriting systems cannot be expressed in such a formalism [5]. Not only do we provide a first-order presentation of higher-order rewriting, but we do not attach to the translation any particular substitution calculus. Instead, we have chosen to work with an abstract formulation of substitution calculi, as done for example in [17] to deal with confluence proofs of λ-calculi with explicit substitutions. As a consequence, the method we propose can be put to work in the presence of different calculi of explicit substitution such as σ [1], $\sigma_{\Uparrow}$ [13], υ [4], f [17], d [17], s [15], χ [20].

The paper is organized as follows. Section 2 recalls the formalism of higher-order rewriting with de Bruijn indices defined in [5], and Section 3 defines a first-order syntax which will be used as target calculus in the conversion procedure given in Section 4. Properties of the conversion procedure are studied in Section 4.1: the conversion is a translation from higher-order rewriting to first-order rewriting modulo, the translation is conservative and finally we give the syntactical criterion to be used in order to decide if a given higher-order system can be translated into a full first-order one (a first-order system modulo an empty theory). We conclude in Section 5.

Due to lack of space proofs are omitted and only the main results are stated. For further details the reader is referred to the full version accessible by ftp at `ftp://ftp.lri.fr/LRI/articles/kesner/ho-to-fo.ps.gz`.

2 The Higher-Order Framework

We briefly recall here the de Bruijn indices based higher-order rewrite formalism called Simplified Expression Reduction Systems with de Bruijn Indices ($SERS_{DB}$) which was introduced in [5]. In full precision we shall work with $SERS_{DB}$ without i(ndex)-metavariables (c.f. full version for details).

Definition 1 (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 α is a symbol and l is a label then $\alpha \in l$ means that the symbol α* appears in *the label l. Other notations are $|l|$ for the* length *of l and $\mathtt{at}(l, n)$ for the n-th element of l assuming $n \leq |l|$. Also, if α occurs (at least once) in l then $\mathtt{pos}(\alpha, l)$ denotes the* position of the first occurrence *of α in l. A* simple label *is a label without repeated symbols.*

Definition 2 (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 $\alpha, \beta, \ldots$,*
- $\{X_l^1, X_l^2, X_l^3, \ldots\}$ *a set of* t-metavariables *(t for term), where l ranges over the set of labels built over binder indicators, denoted $X_l, Y_l, Z_l, \ldots$,*
- $\{f_1, f_2, f_3, \ldots\}$ *a set of* function symbols *equipped with a fixed (possibly zero) arity, denoted $f, g, h, \ldots$,*
- $\{\lambda_1, \lambda_2, \lambda_3, \ldots\}$ *a set of* binder symbols *equipped with a fixed (non-zero) arity, denoted $\lambda, \mu, \nu, \xi, \ldots$.*

Definition 3 (de Bruijn pre-metaterms). *The set of* de Bruijn pre-metaterms*, denoted $\mathcal{PMT}_{db}$, is defined by the following two-sorted grammar:*

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

We shall use $A, B, A_i, \ldots$ to denote de Bruijn pre-metaterms. The symbol $.[\![.]\!]$ is called *de Bruijn metasubstitution operator*. We assume the convention that $\mathtt{S}^0(1) = 1$ 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 $\underline{j}$.

We use $MVar(A)$ to denote the set of metavariables of the de Bruijn pre-metaterm A. An *X-based* t-metavariable is a t-metavariable of the form X_l for some label l, we say in that case that X is the *name of* X_l. We shall use $NMVar(A)$ to denote the set of names of metavariables in A. In order to say that a t-metavariable X_l occurs in a pre-metaterm A we write $X_l \in A$.

We shall need the notion of *metaterm* (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 $\mathtt{S}^i(1)$ correspond to bound variables. Indeed, pre-metaterms like $\xi(X_{\alpha\beta})$ and $\xi(\xi(\underline{4}))$ shall not make sense for us, and hence shall not be considered well-formed. Well-formed pre-metaterms shall be used to describe rewrite rules.

Definition 4 (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 $\mathcal{WF}(A) =_{def} \mathcal{WF}_{\epsilon}(A)$ and $\mathcal{WF}_l(A)$ is defined as follows:*

- $\mathcal{WF}_l(\mathtt{S}^j(1))$ *iff* $j+1 \leq |l|$
- $\mathcal{WF}_l(X_k)$ *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* $\alpha \notin l$ *s.t. for all* $1 \leq i \leq n$, $\mathcal{WF}_{\alpha l}(A_i)$
- $\mathcal{WF}_l(A_1[\![A_2]\!])$ *iff* $\mathcal{WF}_l(A_2)$ *and there exists* $\alpha \notin l$ *such that* $\mathcal{WF}_{\alpha l}(A_1)$

Example 1. Pre-metaterms $\xi(X_\alpha, \lambda(Y_{\beta\alpha}, \underline{2}))$ and $g(\lambda(\xi(h)))$ are metaterms, while $f(\underline{1}, \xi(X_\beta))$, $\lambda(\xi(X_{\alpha\alpha}))$ and $\xi(X_\alpha, X_\beta)$ (with $\alpha \neq \beta$) are not.

Definition 5 (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:*

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

We use $a, b, a_i, b_i, \ldots$ for de Bruijn terms and $E, F, \ldots$ for de Bruijn contexts. The *binder path number* of a context is the number of binders between the $\Box$ and the root. For example the binder path of $E = f(\underline{3}, \xi(\underline{1}, \lambda(\underline{2}, \Box, \underline{3})), \underline{2})$ is 2.

Remark that de Bruijn terms are also de Bruijn pre-metaterms, that is, $\mathcal{T}_{db} \subset \mathcal{PMT}_{db}$, although note that some de Bruijn terms may not be de Bruijn metaterms, i.e. may not be well-formed de Bruijn pre-metaterms, e.g. $\xi(\xi(\underline{4}))$.

The result of substituting a term b for the index $n \geq 1$ in a term a is denoted $a\{\!\{n \leftarrow b\}\!\}$. This is defined as usual [15].

We now recall the definition of rewrite rules, valuations, their validity, and reduction in $SERS_{DB}$.

Definition 6 (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*
- $NMVar(R) \subseteq NMVar(L)$
- *the metasubstitution operator $.[\![.]\!]$ does not occur in L*

Definition 7 (de Bruijn valuation). *A* de Bruijn valuation *$\overline{\kappa}$ is a (partial) function from t-metavariables to de Bruijn terms. A valuation $\overline{\kappa}$ determines in a unique way a function κ (also called valuation) on pre-metaterms as follows:*

$$\begin{array}{ll} \kappa\underline{n} =_{def} \underline{n} & \kappa f(A_1, \ldots, A_n) =_{def} f(\kappa A_1, \ldots, \kappa A_n) \\ \kappa X_l =_{def} \overline{\kappa} X_l & \kappa\xi(A_1, \ldots, A_n) =_{def} \xi(\kappa A_1, \ldots, \kappa A_n) \\ & \kappa(A_1[\![A_2]\!]) =_{def} \kappa(A_1)\{\!\{1 \leftarrow \kappa A_2\}\!\} \end{array}$$

We write $Dom(\kappa)$ for the set $\{X_l \mid \kappa X_l \text{ is defined}\}$, called the *domain* of κ.

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. A typical example is given by a rule like $\mathcal{C} : \xi(\xi(X_{\beta\alpha}))\longrightarrow \xi(\xi(X_{\alpha\beta}))$ where the X-occurrence on the *RHS* of the rule denotes a permutation of the binding context of the X-occurrence on the *LHS*.

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 a pre-cooking function[1] is used in [9] to avoid variable capture in the higher-order unification procedure. In *XRS* [24] the notions of binding arity and pseudo-binding arity are introduced to take into account the binder path number of rewrite rules. Our notion of "coherence" is implemented with *valid valuations* (cf. Definition 9) and it turns out to be more general than the solutions proposed in [9] and [24].

Definition 8 (Value function). *Let a be a de Bruijn term and l be a label of binder indicators. We define the value function $Value(l,a)$ as $Value^0(l,a)$ where*

$$\begin{aligned}
Value^i(l,\underline{n}) &=_{def} \begin{cases} \underline{n} & \text{if } n \le i \\ \mathtt{at}(l, n-i) & \text{if } 0 < n-i \le |l| \\ x_{n-i-|l|} & \text{if } n-i > |l| \end{cases} \\
Value^i(l, f(a_1,\ldots,a_n)) &=_{def} f(Value^i(l,a_1),\ldots Value^i(l,a_n)) \\
Value^i(l, \xi(a_1,\ldots,a_n)) &=_{def} \xi(Value^{i+1}(l,a_1),\ldots, Value^{i+1}(l,a_n))
\end{aligned}$$

It is worth noting that $Value^i(l,\underline{n})$ may give three different kinds of results. This is just a technical resource. Indeed, $Value(\alpha\beta, \xi(f(\underline{3},\underline{1}))) = \xi(f(\beta,\underline{1})) = Value(\beta\alpha, \xi(f(\underline{2},\underline{1})))$ and $Value(\epsilon, f(\xi(\underline{1}),\lambda(\underline{2}))) = f(\xi(\underline{1}),\lambda(x_1)) \neq f(\xi(\underline{1}),\lambda(\alpha)) = Value(\alpha, f(\xi(\underline{1}),\lambda(\underline{2})))$. 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 adequate variable names.

Definition 9 (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 $Value(l,\kappa X_l) = Value(l',\kappa X_{l'})$. 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 $Value(l,\kappa X_l) = Value(l',\kappa X_{l'})$.*

Example 2. Let us consider the de Bruijn rule $\mathcal{C} : \xi(\xi(X_{\beta\alpha}))\longrightarrow \xi(\xi(X_{\alpha\beta}))$. We have that $\kappa = \{X_{\beta\alpha}/\underline{2}, X_{\alpha\beta}/\underline{1}\}$ is valid since $Value(\beta\alpha,\underline{2}) = \alpha = Value(\alpha\beta,\underline{1})$.

As already mentioned the η-contraction rule $\lambda x.app(X,x)\longrightarrow X$ can be expressed in the $SERS_{DB}$ formalism as the rule $(\eta_{dB})\ \ \lambda(app(X_\alpha,\underline{1}))\longrightarrow X_\epsilon$. Our formalism, like other *HORS* in the literature, allows us to use rules like η_{dB} because valid valuations will test for coherence of values.

[1] The pre-cooking function translates a de Bruijn λ-term with metavariables into a $\lambda\sigma$-term by suffixing each metavariable X with as many explicit shift operators as the binder path number of the context obtained by replacing X by $\Box$. This avoids variable capture when the higher-order unification procedure finds solutions for the t-metavariables.

Definition 10 **($SERS_{DB}$-rewriting).** *Let $\mathcal{R}$ be a set of de Bruijn rewrite rules and a, b be de Bruijn terms. We say that a* $\mathcal{R}$-reduces or rewrites *to b, written $a \longrightarrow_{\mathcal{R}} b$, iff there is a de Bruijn rule $(L, R) \in \mathcal{R}$, a de Bruijn valuation κ valid for (L, R), and a de Bruijn context E such that $a = E[\kappa L]$ and $b = E[\kappa R]$.*

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

3 The First-Order framework

In this section we introduce the first-order formalism called Explicit Expression Reduction Systems (*ExERS*) used to translate higher-order rewriting systems based on de Bruijn indices into first-order ones.

Definition 11. *A* substitution declaration *is a (possibly empty) word over the alphabet $\{\mathtt{T}, \mathtt{S}\}$. The symbol $\mathtt{T}$ is used to denote terms and $\mathtt{S}$ to denote substitutions. A* substitution signature *is a set Γ_s of* substitution symbols *equipped with an arity n and a substitution declaration of length n. We use $\sigma : (w)$ where $w \in \{\mathtt{T}, \mathtt{S}\}^n$ if the substitution symbol σ has arity n and substitution declaration w. We use ϵ to denote the empty word.*

Definition 12 **(*ExERS* term algebra).** *An ExERS signature is a set $\Gamma = \Gamma_f \cup \Gamma_b \cup \Gamma_s$ where $\Gamma_f = \{f_1, \ldots, f_n\}$ is a set of function symbols, $\Gamma_b = \{\lambda_1, \ldots, \lambda_n\}$ is a set of binder symbols, Γ_s a substitution signature and Γ_f, Γ_b and Γ_s are pairwise disjoint. Both binder and function symbols come equipped with an arity. Given a set of (term) variables $\mathcal{V} = \{X_1, X_2, \ldots\}$, the term algebra of an ExERS of signature Γ generated by $\mathcal{V}$, is denoted by $\mathcal{T}$ and contains all the objects (denoted by letters o and p) generated by the following grammar:*

`indices`	$n ::= 1 \mid \mathtt{S}(n)$
`terms (T)`	$a ::= X \mid n \mid a[s] \mid f(a_1, \ldots, a_n) \mid \xi(a_1, \ldots, a_n)$
`substitutions (S)`	$s ::= \sigma(o_1, \ldots, o_n)$

where X ranges over $\mathcal{V}$, f over Γ_f, ξ over Γ_b, and σ over Γ_s. The arguments of σ are assumed to respect the sorts prescribed in its substitution declaration and function and binder symbols are assumed to respect their arities.

Letters $a, b, c, \ldots$ and $s, s_i, \ldots$ are used for terms and substitutions, respectively. The $.[.]$ operator is called the *substitution operator*. Binder symbols and substitution operators are considered as having binding power. We shall use $a[s]^n$ to abbreviate $a[s] \ldots [s]$ (n-times). Terms without occurrences of the substitution operator (resp. objects in $\mathcal{V}$) are called *pure* (resp. *ground*) terms. A *context* is a ground term with one (and only one) occurrence of a distinguished term variable called a "hole" (and denoted $\Box$). Letters $E, E_i, \ldots$ are used for contexts. The notion of *binder path number* is defined for pure contexts exactly as in the case of de Bruijn contexts.

The formalism of *ExERS* that we are going to use in order to encode higher-order rewriting consists of two sets of rewrite rules:

1. A set of *proper rewrite rules* governing the behaviour of the function and binder symbols in the signature.
2. A set of substitution rules, called the *substitution calculus* governing the behaviour of the substitution symbols in the signature, and used for propagating and performing/eliminating term substitutions.

Let us define these two concepts formally.

Definition 13 (Substitution macros). *Let Γ_s be a substitution signature. The following symbols not included in Γ_s are called* substitution macros*: $cons$: (TS), $lift$: (S), id : (ϵ) and $shift^j$: (ϵ) for $j \geq 1$. We shall abbreviate $shift^1$ by $shift$. Also, if $j \geq 0$ then $lift^j(s)$ stands for s if $j = 0$ and for $lift(lift^{j-1}(s))$ otherwise. Furthermore, $cons(a_1, \ldots, a_i, s)$ stands for $cons(a_1, \ldots cons(a_i, s))$.*

Definition 14 (Term rewrite and equational systems). *Let Γ be an ExERS signature. An equation is a pair of terms $L \doteq R$ over Γ such that L and R have the same sort and a* term rewrite rule *is a pair of terms (L, R) over Γ such that (1) L and R have the same sort, (2) the head symbol of the LHS of the rule is a function or a binder symbol, and (3) the set of variables of the LHS includes those of the RHS. An equational (resp. term rewrite) system is a set of equations (resp. term rewrite rules).*

Definition 15 (Substitution calculus). *A* substitution calculus *over an ExERS* signature *Γ consists of a set $\mathcal{W}$ of rewrite rules, and an interpretation of each substitution macro as some combination of substitution symbols from Γ_s of corresponding signature. Definition 16 shall require certain properties for these interpretations to be considered meaningful.*

An example of a substitution calculus is σ [1] with $cons(t, s) =_{def} t \cdot s$, $lift(s) =_{def} \underline{1} \cdot (s \circ \uparrow)$, $id =_{def} id$ and $shift^j =_{def} \uparrow \circ \ldots (\uparrow \circ \uparrow)$, where $\uparrow$ appears j times.

Definition 16 (Basic substitution calculus). *A substitution calculus $\mathcal{W}$ over Γ is said to be* basic *if the following conditions are satisfied:*

1. *$\mathcal{W}$ is complete (strongly normalizing and confluent) over the ground terms in $\mathcal{T}$. We use $\mathcal{W}(a)$ to indicate the unique $\mathcal{W}$-normal form of a.*
2. *$\mathcal{W}$-normal forms of ground terms are pure terms.*
3. *For each $f \in \Gamma_f$ and $\xi \in \Gamma_b$: $\mathcal{W}(f(a_1, \ldots, a_n)) = f(\mathcal{W}(a_1), \ldots, \mathcal{W}(a_n))$ and $\mathcal{W}(\xi(a_1, \ldots, a_n)) = \xi(\mathcal{W}(a_1), \ldots, \mathcal{W}(a_n))$.*
4. *Rules for propagating substitutions over function symbols and binders are contained in $\mathcal{W}$, for each $f \in \Gamma_f$ and $\xi \in \Gamma_b$:*

$$\begin{array}{ll} (func_f) & f(X^1, \ldots, X^n)[s] \longrightarrow f(X^1[s], \ldots, X^n[s]) \\ (bind_\xi) & \xi(X^1, \ldots, X^n)[s] \longrightarrow \xi(X^1[lift(s)], \ldots, X^n[lift(s)]) \end{array}$$

5. *For every substitution s, $\underline{1}[lift(s)] =_{\mathcal{W}} \underline{1}$.*
6. *For every substitution s and every $m \geq 0$, $\underline{m+1}[lift(s)] =_{\mathcal{W}} \underline{m}[s][shift]$.*

7. *For every term a and substitution s we have $\underline{1}[cons(a,s)] =_{\mathcal{W}} a$.*
8. *For every term a, substitution s, $m \geq 0$ we have $\underline{m+1}[cons(a,s)] =_{\mathcal{W}} \underline{m}[s]$.*
9. *For every $m, j \geq 1$ we have $\underline{m}[shift^j] =_{\mathcal{W}} \underline{m+j}$.*
10. *For every term a we have $a[id] =_{\mathcal{W}} a$.*

Example 3. The σ [1], $\sigma_{\Uparrow}$ [13] and ϕ [22] calculi are basic substitution calculi where the set of function and binder symbols are $\{app\}$ and $\{\lambda\}$, respectively.

The reader may have noted that the macro-based presentation of substitution calculi makes use of parallel substitutions (since $cons(.,.)$ has substitution declaration TS). Nevertheless, the results presented in this work may be achieved via a macro-based presentation using a simpler set of substitutions (such as for example the one used in [17]), where $scons(.)$ has substitution declaration T and the macro $shift^i$ is only defined for $i = 1$. Indeed, the expression $b[cons(a_1, \ldots, a_n, shift^j)]$ could be denoted by the expression

$$b[lift^n(shift)]^j[scons(a_1[shift]^{n-1})] \ldots [scons(a_n)]$$

Definition 17 (*ExERS* and *FExERS*). *Let Γ be an ExERS signature, $\mathcal{W}$ a basic substitution calculus over Γ and $\mathcal{R}$ a set of term rewrite rules. If each rule of $\mathcal{R}$ has sort T then $\mathcal{R}_{\mathcal{W}} =_{def} (\Gamma, \mathcal{R}, \mathcal{W})$ is called an Explicit Expression Reduction System (ExERS). If, in addition, the LHS of each rule in $\mathcal{R}$ contains no occurrences of the substitution operator $.[.]$ then $\mathcal{R}_{\mathcal{W}}$ is called a Fully Explicit Expression Reduction System (FExERS).*

Since reduction in $SERS_{DB}$ only takes place on terms, and first-order term rewrite systems will be used to simulate higher-order reduction, all the rules of a term rewrite system $\mathcal{R}$ are assumed to have sort T. However, rewrite rules of $\mathcal{W}$ may have any sort.

Example 4. Consider the signature Γ formed by $\Gamma_f = \{app\}$, $\Gamma_b = \{\lambda\}$ and Γ_s any substitution signature. Let $\mathcal{W}$ be a basic substitution calculus over Γ. Then for $\mathcal{R} : app(\lambda(X), Y) \longrightarrow_{\beta db} X[cons(Y, id)]$ we have that $\mathcal{R}_{\mathcal{W}}$ is an *FExERS*, and for $\mathcal{R}' : \mathcal{R} \cup \{\lambda(app(X[shift], \underline{1})) \longrightarrow_{\eta db} X\}$, $\mathcal{R}'_{\mathcal{W}}$ is an *ExERS*.

Reduction in an *ExERS* $\mathcal{R}_{\mathcal{W}}$ is first-order reduction in $\mathcal{R}$ modulo $\mathcal{W}$-equality. In contrast, reduction in a *FExERS* $\mathcal{R}_{\mathcal{W}}$ is just first-order reduction in $\mathcal{R} \cup \mathcal{W}$. Before defining these notions more precisely we recall the definition of assignment.

Definition 18 (Assignment (a.k.a. grafting)). *Let $\overline{\rho}$ be a (partial) function mapping variables in $\mathcal{V}$ to ground terms. We define an* assignment *ρ as the unique homeomorphic extension of $\overline{\rho}$ over the set $\mathcal{T}$.*

Definition 19 (Reduction and Equality). *Let o and p be two ground terms of sort T or S. Given a rewrite system $\mathcal{R}$, we say that o rewrites to p in one step, denoted $o \longrightarrow_{\mathcal{R}} p$, iff $o = E[\rho L]$ and $p = E[\rho R]$ for some assignment ρ, some context E and some rewrite rule (L, R) in $\mathcal{R}$. We shall use $\stackrel{*}{\longrightarrow}_{\mathcal{R}}$ to denote the reflexive transitive closure of the one-step rewrite relation.*

Given an equational system $\mathcal{E}$, we say that o equals p modulo $\mathcal{E}$ in one step, denoted $o =^1_{\mathcal{E}} p$, iff $o = E[\rho L]$ and $p = E[\rho R]$ for some assignment ρ, some context E and some equation $L \doteq R$ in $\mathcal{E}$. We use $=_{\mathcal{E}}$ to denote the reflexive symmetric transitive closure of $=^1_{\mathcal{E}}$, and say that o equals p modulo $\mathcal{E}$ if $o =_{\mathcal{E}} p$.

Definition 20 (*ExERS* and *FExERS*-rewriting). *Let $\mathcal{R}_{\mathcal{W}}$ be an ExERS, $\mathcal{R}'_{\mathcal{W}}$ a FExERS and o, p ground terms of sort* S *or* T. *We say that o $\mathcal{R}_{\mathcal{W}}$-*reduces or rewrites to *p, written $o \longrightarrow_{\mathcal{R}_{\mathcal{W}}} p$, iff $o \longrightarrow_{\mathcal{R}/\mathcal{W}} p$ (i.e. $o =_{\mathcal{W}} o' \longrightarrow_{\mathcal{R}} p' =_{\mathcal{W}} p$); and o $\mathcal{R}'_{\mathcal{W}}$-*reduces to *$p$, iff $o \longrightarrow_{\mathcal{R}' \cup \mathcal{W}} p$.*

3.1 Properties of Basic Substitution Calculi

This subsection takes a look at properties enjoyed by basic substitution calculi and introduces a condition called the *Scheme* [17]. Basic substitution calculi satisfying the scheme ease inductive reasoning when proving properties over them without compromising the genericity achieved by the macro-based presentation.

Lemma 1 (Behavior of Substitutions in Basic Substitution Calculi). *Let $\mathcal{W}$ be a basic substitution calculus and $m \geq 1$.*

1. *For all $n \geq 0$ and s in* S: $\underline{m}[lift^n(s)] =_{\mathcal{W}} \begin{cases} \underline{m-n}[s][shift]^n & \text{if } m > n \\ \underline{m} & \text{if } m \leq n \end{cases}$
2. *For all $n \geq m \geq 1$ and all terms $a_1, \ldots, a_n$: $\underline{m}[cons(a_1, \ldots, a_n, s)] =_{\mathcal{W}} a_m$*
3. *For all pure terms a, b and $m \geq 1$: $a\{\!\{m \leftarrow b\}\!\} =_{\mathcal{W}} a[lift^{m-1}(cons(b, id))]$.*

Definition 21 (The Scheme). *We say that a basic substitution calculus $\mathcal{W}$ obeys the* scheme *iff for every index $\underline{m}$ and every substitution symbol $\sigma \in \Gamma_s$ of arity q one of the following two conditions hold:*

1. *There exists a de Bruijn index $\underline{n}$, positive numbers $i_1, \ldots, i_r$ ($r \geq 0$) and substitutions $u_1, \ldots, u_k$ ($k \geq 0$) such that*
   - *$1 \leq i_1, \ldots, i_r \leq q$ and all the i_j's are distinct*
   - *for all $o_1, \ldots, o_q$ we have: $\underline{m}[\sigma(o_1, \ldots, o_q)] =_{\mathcal{W}} \underline{n}[o_{i_1}] \ldots [o_{i_r}][u_1] \ldots [u_k]$*
2. *There exists an index i ($1 \leq i \leq q$) such that for all $o_1, \ldots, o_q$ we have: $\underline{m}[\sigma(o_1, \ldots, o_q)] =_{\mathcal{W}} o_i$*

We assume these equations to be well-typed: whenever the first case holds, then $o_{i_1}, \ldots, o_{i_r}$ are substitutions, whenever the second case holds, o_i is of sort T.

Example 5. Example of calculi satisfying the scheme are σ, $\sigma_{\Uparrow}$, υ, f and d [17].

4 From Higher-Order to First-Order rewriting

In this section we give an algorithm, referred to as the Conversion Procedure, to translate any higher-order rewrite system in the formalism $SERS_{DB}$ to a first-order *ExERS*. The Conversion Procedure is somewhat involved since several conditions, mainly related to the labels of t-metavariables, must be met in order for a substitution to be admitted as *valid*. The idea is to replace all occurrences of t-metavariables X_l by a first-order variable X followed by an appropriate *index-adjusting explicit substitution* which computes valid valuations.

We first give the conversion rules of the translation, then we prove its properties in Section 4.1.

Definition 22 (Binding allowance). *Let M be a metaterm and $\{X_{l_1}, \ldots, X_{l_n}\}$ be the set of all the t-metavariables with name X occurring in M. Then, the* binding allowance of X in M, *noted* $\mathtt{Ba}_M(X)$, *is the set* $\bigcap_{i=1}^n l_i$. *Likewise, we define the* binding allowance of X in a rule (L, R), *written* $\mathtt{Ba}_{(L,R)}(X)$.

Example 6. Let $M = f(\xi(X_\alpha), \xi(\lambda(X_{\beta\alpha})), \nu(\xi(X_{\alpha\gamma})))$, then $\mathtt{Ba}_M(X) = \{\alpha\}$.

Definition 23 (Shifting index). *Let M be a metaterm, X_l a t-metavariable occurring in M, and i a position in l. The* shifting index determined by X_l in M at position i, *denoted* $\mathtt{Sh}(M, X_l, i)$, *is defined as*

$$\mathtt{Sh}(M, X_l, i) =_{def} |\{j \mid \mathtt{at}(l, j) \notin \mathtt{Ba}_M(X), j \in 1..i-1\}|$$

$\mathtt{Sh}(M, X_l, i)$ *is just the total number of binder indicators in l at positions $1..i-1$ that do not belong to* $\mathtt{Ba}_M(X)$ *(thus* $\mathtt{Sh}(M, X_l, 1)$ *is always* 0*). Likewise, we define the* shifting index determined by X_l in a rule (L, R) at position i, *written* $\mathtt{Sh}((L, R), X_l, i)$.

Example 7. Let $M = f(\xi(X_\alpha), \xi(\lambda(X_{\beta\alpha})), \nu(\xi(X_{\alpha\gamma})))$. Then $\mathtt{Sh}(M, X_{\beta\alpha}, 2) = 1$ and $\mathtt{Sh}(M, X_\alpha, 1) = \mathtt{Sh}(M, X_{\alpha\gamma}, 2) = 0$.

Definition 24 (Pivot). *Let (L, R) be a $SERS_{DB}$-rewrite rule and let suppose that $\{X_{l_1}, \ldots, X_{l_n}\}$ is the set of all X-based t-metavariables in (L, R). If* $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$, *then X_{l_j} for some $j \in 1..n$ is called an* (X-based) pivot *if*

1. *$|l_j| \leq |l_i|$ for all $i \in 1..n$, and*
2. *$X_{l_j} \in L$ whenever possible.*

A pivot set for a rewrite rule (L, R) *is a set of pivot t-metavariables, one for each name X in L such that* $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$. *This notion extends to a set of rewrite rules as expected.*

Note that Definition 24 admits the existence of more than one X-based pivot t-metavariable. A pivot set for (L, R) fixes a t-metavariable for each t-metavariable name having a non-empty binding allowance.

Example 8. Both t-metavariables $X_{\alpha\beta}$ and $X_{\beta\alpha}$ can be chosen as X-based pivot in the rewrite rule $\mathcal{R} : \mathit{Implies}(\exists(\forall(X_{\alpha\beta})), \forall(\exists(X_{\beta\alpha}))) \longrightarrow \mathit{true}$. In the rewrite rule $\mathcal{R}' : f(Y_\epsilon, \lambda(\xi(X_{\alpha\beta})), \nu(\lambda(X_{\beta\alpha}))) \longrightarrow \mu(X_\alpha, Y_\alpha)$ the t-metavariable X_α is the only possible X-based pivot.

Definition 25 (Conversion of t-metavariables). *Consider a $SERS_{DB}$-rewrite rule (L, R) and a pivot set for (L, R). We consider the following cases for every t-metavariable name X occurring in L:*

1. $\mathtt{Ba}_{(L,R)}(X)=\emptyset$. *Then replace each t-metavariable* X_l *in* (L,R) *by the metaterm* $X[\mathit{shift}^{|l|}]$, *and those t-metavariables* X_l *with* $l=\epsilon$ *simply by* X.
   This shall allow for example the rule $f(\lambda(app(X_\alpha,\underline{1}),X_\epsilon))\longrightarrow X_\epsilon$ *to be converted to* $f(\lambda(app(X[\mathit{shift}],\underline{1}),X))\longrightarrow X$.
2. $\mathtt{Ba}_{(L,R)}(X)=\{\beta_1,\ldots,\beta_m\}$ *with* $m>0$. *Let* X_l *be the pivot t-metavariable for* X *given by the hypothesis. We replace all occurrences of a t-metavariable* X_k *in* (L,R) *by the term* $X[cons(b_1,\ldots,b_{|l|},\mathit{shift}^j)]$ *where* $j=_{def}|k|+|l\setminus \mathtt{Ba}_{(L,R)}(X)|$ *and the* b_i*'s depend on whether* X_k *is a pivot t-metavariable or not. As an optimization and in the particular case that the resulting term* $X[cons(b_1,\ldots,b_{|l|},\mathit{shift}^j)]$ *is of the form* $cons(\underline{1},\ldots,\underline{|l|},\mathit{shift}^{|l|})$, *then we simply replace* X_k *by* X.
   The substitution $cons(b_1,\ldots,b_{|l|},\mathit{shift}^j)$ *is called the* index-adjusting substitution corresponding to X_k *and it is defined as follows:*
   (a) *if* X_k *is the pivot (hence* $l=k$*), then*
   $$b_i=\begin{cases}\underline{i} & \text{if } \mathtt{at}(l,i)\in \mathtt{Ba}_{(L,R)}(X)\\ \underline{|l|+1+\mathtt{Sh}((L,R),X_l,i)} & \text{if } \mathtt{at}(l,i)\notin \mathtt{Ba}_{(L,R)}(X)\end{cases}$$
   (b) *if* X_k *is not the pivot then*
   $$b_i=\begin{cases}\underline{\mathtt{pos}(\beta_h,k)} & \text{if } i=\mathtt{pos}(\beta_h,l) \text{ for some } \beta_h\in \mathtt{Ba}_{(L,R)}(X)\\ \underline{|k|+1+\mathtt{Sh}((L,R),X_l,i)} & \text{otherwise}\end{cases}$$

Note that for an index-adjusting substitution $X[cons(b_1,\ldots,b_{|l|},\mathit{shift}^j)]$ each b_i is a distinct de Bruijn index and less than or equal to j. Substitutions of this form have been called pattern substitutions in [10], where unification of higher-order patterns via explicit substitutions is studied.

Definition 26 (The Conversion Procedure). *Given a* $SERS_{DB}$ $\mathcal{R}$ *the following actions are taken:*

1. **Convert rules.** *The transformation of Definition 25 is applied to all rules in* $\mathcal{R}$ *prior selection of some set of pivot sets for* $\mathcal{R}$.
2. **Replace metasubstitution operator.** *All submetaterms of the form* $M[\![N]\!]$ *in* $\mathcal{R}$ *are replaced by the term substitution operator* $M[cons(N,id)]$.

Example 9. Below we present some examples of conversion of rules. We have fixed $\mathcal{W}$ to be the σ-calculus.

$SERS_{DB}$ *rule*	*Pivot selected*	*Transformed rule*
$\lambda(app(X_\alpha,\underline{1}))\longrightarrow X_\epsilon$	-	$\lambda(app(X[\uparrow],\underline{1}))\longrightarrow X$
$\lambda(\lambda(X_{\alpha\beta}))\longrightarrow\lambda(\lambda(X_{\beta\alpha}))$	$X_{\alpha\beta}$	$\lambda(\lambda(X))\longrightarrow\lambda(\lambda(X[\underline{2}\cdot\underline{1}\cdot(\uparrow\circ\uparrow)]))$
$f(\lambda(\lambda(X_{\alpha\beta})),\lambda(\lambda(X_{\beta\alpha})))\longrightarrow\lambda(X_\gamma)$	-	$f(\lambda(\lambda(X[\uparrow\circ\uparrow])),\lambda(\lambda(X[\uparrow\circ\uparrow])))\longrightarrow\lambda(X[\uparrow])$
$app(\lambda(X_\alpha),Z_\epsilon)\longrightarrow_{\beta_{db}} X_\alpha[\![Z_\epsilon]\!]$	X_α, Z_ϵ on *LHS*	$app(\lambda(X),Z)\longrightarrow X[cons(Z,id)]$

Let (L,R) be a $SERS_{DB}$-rule and P a pivot set for (L,R). We write $\mathcal{C}_P(L,R)$ for the result of applying the conversion of Definition 26 to (L,R) with pivot set P. We refer to $\mathcal{C}_P(L,R)$ as the *converted version of* (L,R) *via* P.

Note that if the $SERS_{DB}$-rewrite rule (L, R) which is input to the Conversion Procedure is such that for every name X in (L, R) there is a label l with *all* metavariables in (L, R) of the form X_l, then all X_l are replaced simply by X. This is the case of the rule β_{db} of Example 9.

Also, observe that if we replace our $cons(.,.)$ macro by a $scons(.)$ of substitution declaration `T` as defined in [17] then the "Replace metasubstitution operator" step in Definition 26 converts a metaterm of the form $M[\![N]\!]$ into $M[scons(N)]$, yielding first-order systems based on substitution calculi, such as $\lambda\upsilon$, which do not implement parallel substitution.

The resulting system of the Conversion Procedure is coded as an *ExERS*, a framework for defining first-order rewrite systems where $\mathcal{E}$-matching is used, $\mathcal{E}$ being an equational theory governing the behaviour of the index adjustment substitutions. Moreover, if it is possible, an *ExERS* may further be coded as a *FExERS* where reduction is defined on first-order terms and matching is just syntactic first-order matching, obtaining a *full first-order system.*

Definition 27 (First-order version of $\mathcal{R}$). *Let Γ be an ExERS signature and let $\mathcal{R}$ be a $SERS_{DB}$. Consider the system $fo(\mathcal{R})$ obtained by applying the Conversion Procedure to $\mathcal{R}$ and let $\mathcal{W}$ be a substitution calculus over Γ. Then the ExERS $fo(\mathcal{R})_{\mathcal{W}}$ is called a* first order-version *of $\mathcal{R}$.*

In what follows we shall assume given some fixed *basic* substitution calculus $\mathcal{W}$. Thus, given a $SERS_{DB}$ $\mathcal{R}$ we shall speak of *the* first-order version of $\mathcal{R}$. This requires considering pivot selection, an issue we take up next.

Assume given some rewrite rule (L, R) and different pivot sets P and Q for this rule. It is clear that $\mathcal{C}_P(L, R)$ and $\mathcal{C}_Q(L, R)$ shall not be identical. Nevertheless, we may show that the reduction relation generated by both of these converted rewrite rules is identical.

Proposition 1. *Let (L, R) be a $SERS_{DB}$-rewrite rule and let P and Q be different pivot sets for this rule. Then the rewrite relation generated by both $\mathcal{C}_P(L, R)$ and $\mathcal{C}_Q(L, R)$ are identical.*

4.1 Properties of the conversion

The Conversion Procedure satisfies two important properties: each higher-order rewrite step may be simulated by first-order rewriting (*simulation*) and rewrite steps in the first-order version of a higher-order system $\mathcal{R}$ can be projected in $\mathcal{R}$ (*conservation*).

Proposition 2 (Simulation). *Let $\mathcal{R}$ be an $SERS_{DB}$ and let $fo(\mathcal{R})_{\mathcal{W}}$ be its first-order version. Suppose $a \longrightarrow_{\mathcal{R}} b$ then*

1. *if $fo(\mathcal{R})_{\mathcal{W}}$ is an ExERS then $a \longrightarrow_{fo(\mathcal{R})/\mathcal{W}} b$.*
2. *if $fo(\mathcal{R})_{\mathcal{W}}$ is a FExERS then $a \longrightarrow_{fo(\mathcal{R})} \circ \overset{*}{\longrightarrow}_{\mathcal{W}} b$ where $\circ$ denotes relation composition.*

Proposition 3 (Conservation). *Let $\mathcal{R}$ be a $SERS_{DB}$ and $fo(\mathcal{R})_{\mathcal{W}}$ its first-order version with $\mathcal{W}$ satisfying the scheme. If $a \longrightarrow_{fo(\mathcal{R})_{\mathcal{W}}} b$ then $\mathcal{W}(a) \overset{*}{\longrightarrow}_{\mathcal{R}} \mathcal{W}(b)$.*

4.2 Essentially first-order *HORS*

This last subsection gives a very simple syntactical criterion that can be used to decide if a given higher-order rewrite system can be converted into a full first-order rewrite system (modulo an empty theory). In particular, we can check that many higher-order calculi in the literature, e.g. λ-calculus, verify this property.

Definition 28 (Essentially first-order *HORS*). *A $SERS_{DB}$ $\mathcal{R}$ is called* essentially first-order *if $fo(\mathcal{R})_{\mathcal{W}}$ is a FExERS for $\mathcal{W}$ a basic substitution calculus.*

Definition 29 (fo-condition). *A $SERS_{DB}$ $\mathcal{R}$ satisfies the* fo-condition *if every rewrite rule $(L, R) \in \mathcal{R}$ satisfies: for every name X in L let $X_{l_1}, \ldots, X_{l_n}$ be all the X-based t-metavariables in L, then*

1. *$l_1 = l_2 \ldots = l_n$ and (the underlying set of) l_1 is $\mathtt{Ba}_{(L,R)}(X)$, and*
2. *for all $X_k \in R$ we have $|k| \geq |l_1|$.*

In the above definition note that $l_1 = l_2 \ldots = l_n$ means that labels $l_1, \ldots, l_n$ must be *identical* (for example $\alpha\beta \neq \beta\alpha$). Also, by Definition 6, l_1 is simple.

Example 10. Consider the β_{db}-calculus: $app(\lambda(X_\alpha), Z_\epsilon) \longrightarrow_{\beta_{db}} X_\alpha[\![Z_\epsilon]\!]$. The β_{db}-calculus satisfies the fo-condition.

Proposition 4 puts forward the importance of the fo-condition. Its proof relies on a close inspection of the Conversion Procedure.

Proposition 4. *Let $\mathcal{R}$ be a $SERS_{DB}$ satisfying the fo-condition. Then $\mathcal{R}$ is essentially first-order.*

Note that many results on higher-order systems (e.g. perpetuality [19], standardization [21]) require *left-linearity* (a metavariable may occur at most once on the *LHS* of a rewrite rule), and *fully-extendedness or locality* (if a metavariable $X(t_1, \ldots, t_n)$ occurs on the *LHS* of a rewrite rule then $t_1, \ldots, t_n$ is the list of variables bound above it). The reader may find it interesting to observe that these conditions together seem to imply the fo-condition. A proof of this fact would require either developing the results of this work in the above mentioned *HORS* or via some suitable translation to the $SERS_{DB}$ formalism.

5 Conclusions and future work

This work presents an encoding of higher-order term rewriting systems into first-order rewriting systems modulo an equational theory. This equational theory takes care of the substitution process. The encoding has furthermore allowed us to identify in a simple syntactical manner, via the so-called fo-condition, a class of *HORS* that are fully first-order in that they may be encoded as first-order rewrite systems modulo an empty equational theory. This amounts to incorporating, into the first-order notion of reduction, not only the computation of substitutions but also the higher-order (pattern) matching process. It is fair to say that a higher-order rewrite system satisfying this condition requires a simple matching process, in contrast to those that do not satisfy this condition (such as the $\lambda\beta\eta$-calculus). Other syntactical restrictions, such as linearity and locality, imposed on higher-order rewrite systems in [19, 21] in order to reason about their properties can be related to the fo-condition in a very simple way. This justifies that the fo-condition, even if obtained very technically in this paper, may be seen as an interpretation of what a well-behaved higher-order rewrite system is.

Moreover, this encoding has been achieved by working with a general presentation of substitution calculi rather than dealing with some particular substitution calculus. Any calculus of explicit substitutions satisfying this general presentation based on macros will do.

Some further research directions are summarized below:

- As already mentioned, the encoding opens up the possibility of transferring results concerning confluence, termination, completion, evaluation strategies, implementation techniques, etc. from the first-order framework to the higher-order framework.
- Given a $SERS_{DB}$ $\mathcal{R}$ note that the *LHS*s of rules in $fo(\mathcal{R})$ may contain occurrences of the substitution operator (pattern substitutions). It would be interesting to deal with pattern substitutions and "regular" term substitutions (those arising from the conversion of the de Bruijn metasubstitution operator $.[\![.]\!]$) as different substitution operators at the object-level. This would neatly separate the explicit matching computation from that of the usual substitution replacing terms for variables.
- This work has been developed in a type-free framework. The notion of type is central to Computer Science. This calls for a detailed study of the encoding process dealing with typed higher-order rewrite systems such as *HRS* [23].
- The ideas presented in this paper could be used to relax conditions in [9] where only rewrite systems with atomic propositions on the *LHS*s of rules are considered.

Acknowledgements. We thank Bruno Guillaume for helpful remarks.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
3. L. Bachmair and N. Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.
4. Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.
5. E. Bonelli, D. Kesner, and A. Ríos. A de Bruijn notation for higher-order rewriting. In *Proc. of the Eleventh Int. Conference on Rewriting Techniques and Applications*, Norwich, UK, July 2000.

6. R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitutions. *Journal of Mathematical Structures in Computer Science*, 2000. To appear.
7. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.
8. N. Dershowitz and J-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.
9. G. Dowek, T. Hardin and C. Kirchner. Theorem proving modulo. Technical Report RR 3400, INRIA, 1998.
10. G. Dowek, T. Hardin, C. Kirchner and F. Pfenning. Unification via Explicit Substitutions: The Case of Higher-Order Patterns. Technical Report RR3591. INRIA. 1998.
11. G. Dowek, T. Hardin and C. Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157:183–235, 2000.
12. G. Dowek, T. Hardin and C. Kirchner. Hol-lambda-sigma: an intentional first-order expression of higher-order logic. *Mathematical Structures in Computer Science*, 11:1–25, 2001.
13. T. Hardin and J-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.
14. J-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, Trento, Italy, 1999.
15. F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *Proc. of the Int. Symposium on Programming Language Implementation and Logic Programming (PLILP)*, LNCS 982. 1995.
16. S. Kamin and J. J. Lévy. Attempts for generalizing the recursive path orderings. University of Illinois, 1980.
17. D. Kesner. Confluence of extensional and non-extensional lambda-calculi with explicit substitutions. *Theoretical Computer Science*, 238(1-2):183–220, 2000.
18. Z. Khasidashvili. Expression Reduction Systems. In *Proc. of I. Vekua Institute of Applied Mathematics*, volume 36, Tbilisi, 1990.
19. Z. Khasidashvili and M. Ogawa and V. van Oostrom. Uniform Normalization beyond Orthogonality. Submitted for publication, 2000.
20. P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn levels. In *Proc. of the Sixth Int. Conference on Rewriting Techniques and Applications*, LNCS 914, 1995.
21. P-A. Melliès. Description Abstraite des Systèmes de Réécriture. Thèse de l'Université Paris VII, December 1996.
22. C. Muñoz. A left-linear variant of $\lambda\sigma$. In Michael Hanus, J. Heering, and K. (Karl) Meinke, editors, *Proc. of the 6th Int. Conference on Algebraic and Logic Programming (ALP'97)*, LNCS 1298. 1997.
23. T. Nipkow. Higher-order critical pairs. in *Proc. of the Sixth Annual IEEE Symposium on Logic in Computer Science*, 1991.
24. B. Pagano. *Des Calculs de Susbtitution Explicite et leur application à la compilation des langages fonctionnels.* PhD thesis, Université Paris VI, 1998.
25. R. Pollack. Closure under alpha-conversion. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs (TYPES)*, LNCS 806. 1993.
26. H. Zantema. Termination of Term Rewriting by Semantic Labelling. Fundamenta Informaticae, volume 24, pages 89-105, 1995.