Expression Reduction Systems with Patterns

Julien Forest[1] and Delia Kesner[2]

[1] LRI (CNRS UMR 8623), Université Paris-Sud, France, forest@lri.fr.
[2] PPS (CNRS UMR 7126), Université Paris 7, France, kesner@pps.jussieu.fr.

Abstract. We introduce a new higher-order rewriting formalism, called *Expression Reduction Systems with Patterns (ERSP)*, where abstraction is not only allowed on variables but also on nested patterns. These patterns are built by combining standard *algebraic* patterns with *choice* constructors used to denote different possible structures allowed for an abstracted argument. In other words, the non deterministic choice between different rewriting rules which is inherent to classical rewriting formalisms can be lifted here to the level of patterns. We show that confluence holds for a reasonable class of systems and terms.

Introduction

Higher-order rewrite systems are able to combine formalisms coming from proof theory, such as λ-calculus, with formalisms arising in algebraic specifications, such as first-order rewrite systems. The main idea behind higher-order rewriting concerns the transformation of terms in the presence of *binding* mechanisms for variables and substitutions. Thus for example, functional and logic programming, equational reasoning, object-oriented programming, concurrent systems and theorem provers may be encoded by higher-order rewrite systems.

Many higher-order rewrite systems exist in the literature starting at the seminal work by J-W. Klop [15]. Many other interesting formalisms [4, 13, 17, 20, 22] were introduced later. The theory of higher-order rewriting is considerably more involved than that of first-order rewriting; many articles were devoted to the study of its foundations, applications, semantics and implementation.

In all the higher-order formalisms mentioned before the binding mechanism is only allowed on *variables*. However, most popular functional languages and proof assistants allow definitions by cases via pattern-matching mechanisms. Thus, a natural extension of higher-order rewriting consists in the use of binders for patterns so that a projection function like $\lambda\langle x, y\rangle.x$ would be also acceptable.

The *Pattern-Matching Calculus* [12], proposed as a theoretical framework to study pattern-matching in a *pure functional paradigm*, allows precisely this kind of binding mechanisms. Its evaluation process is given by the following generalization of the standard β-rule to the case of patterns:

$$(\beta_{PM}) \qquad app(\lambda\mathbb{X}.M, N) \longrightarrow M\{\mathbb{X} \texttt{ by } N\}$$

where $\mathbb{X}$ denotes a pattern and $\{\mathbb{X} \texttt{ by } N\}$ denotes a substitution resulting from the pattern-matching operation on the pattern $\mathbb{X}$ and the term N.

This calculus was later extended with *explicit operators* [6, 5, 9]; weak reduction was widely studied in [9]. Another language allowing abstractions on patterns is the ρ-calculus [8], for which typing [14] and explicit operators [7] were defined and analyzed.

In this paper we introduce a new higher-order formalism, called *Expression Reduction Systems with Patterns* (ERSP), where binding mechanisms are allowed on complex patterns. Our calculus constitutes an extension of ERS [13] and SERS [4] to the case of patterns, and a generalization of the Pattern-Matching Calculus to the case of general higher-order rewriting (and not only functional rewriting). ERSP patterns are defined as combinations of standard *algebraic* structures with special *choice* constructors used to denote different possible syntactic forms for any abstracted argument. Thus for example, the function which computes the length of a given list may be specified as the following ERSP term:

$$\lambda\, a\langle nil, cons(x,t)\rangle.a\langle 0, 1+len(t)\rangle$$

where λ is the classical function binder, a is a special variable used to identify the set of *choice* patterns nil and $cons(x,t)$ with the set of their corresponding continuations 0 and $1+len(t)$.

We carefully extend all the expected notions of rewriting to our framework, namely, terms, metaterms, rewrite rules, substitutions, reduction, etc. We then identify a class of ERSP, called *orthogonal l-constructor* systems, and a class of terms, called *l-constructor deterministic terms*, for which confluence holds. More precisely, reduction on this class of terms via this class of systems corresponds to reduction on ordinary terms (without patterns) in classical orthogonal higher-order systems [17, 20]. Much more, our confluence result turns out to give in particular a confluence result for SERS.

The paper is organized as follows. Sections 1 and 2 introduce the basic ingredients of the syntactic formalism ERSP. In Section 3 we develop an example of reduction in our framework. Section 4 is devoted to study a restriction of the class of ERSP so that confluence will follow (Section 5). We conclude and give many further research directions in Section 6. By lack of space we cannot provide here all the proofs but a complete version of this work can be found in [10].

1 Basic notions of the ERSP formalism

We consider a set $\mathcal{UV}$ of *usual variables* denoted $x, y, z, \ldots$, a set $\mathcal{CV}$ of *choice variables* denoted $a, b, c, \ldots$, a set $\mathcal{PV}$ of *pattern metavariables* denoted $\mathbb{X}, \mathbb{Y}, \ldots$, a set $\mathcal{TV}$ of *term metavariables* denoted $M, N, \ldots$, a set $\mathcal{F}$ of *function symbols* equipped with a fixed (possibly zero) arity, denoted $f, g, h, \ldots$, a set $\mathcal{B}$ of *binder symbols* denoted $\lambda, \mu, \nu, \ldots$. We assume all these sets to be denumerable and disjoint. When no special distinction is needed for the previous sets of variables and metavariables will use the symbols $\widehat{x}, \widehat{y}, \widehat{z}, \ldots$.

Metapatterns (p) and metaterms (t) are generated by the grammars:

$p ::=$	x	usual variable	$t ::=$	x	usual variable
$\mid$	$\mathbb{X}$	pattern metavariable	$\mid$	M	term metavariable
$\mid$	$f(p,\ldots,p)$	algebraic	$\mid$	$f(t,\ldots,t)$	algebraic
$\mid$	$a\langle p,\ldots,p\rangle$	choice	$\mid$	$a\langle t,\ldots,t\rangle$	case
$\mid$	$@(p,\ldots,p)$	contraction	$\mid$	$\mu p.t$	abstraction
$\mid$	$\_$	wildcard	$\mid$	$t\{p\ \texttt{by}\ t\}$	pattern-matching

The constructor $@()$ is varyadic, i.e. it has no fix arity. The constructor $a\langle\ \rangle$ is also varyadic, but with an arity different from 0. We assume that whenever a choice variable a appears inside t, then all its occurrences have the same arity : thus, a term like $\mu a\langle x\rangle.a\langle x,y\rangle$ is not allowed. The symbol { `by` } is called the pattern-matching constructor. The metaterms $\mu p.t$ and $t\{p\ \texttt{by}\ t'\}$ define bindings whose scope is t for all the (usual and choice) variables occurring in p.

A metapattern (resp. metaterm) is said to be a *pattern* (resp. *preterm*) if it contains no metavariables. A preterm is said to be a *term* if it contains no pattern-matching constructors.

We denote by $\mathcal{MV}(p)$ (resp. $\mathcal{V}ar(p)$) the set of all the *pattern* metavariables (resp. variables) appearing in a metapattern (resp. pattern) p. We denote by $\mathcal{MV}(t)$ the set of all the *term* metavariables appearing in t.

Definition 1. *A metapattern is called linear if each variable and metavariable appears at most once in it. We use the notation $p \in p'$ to say that the metapattern p appears inside the metapattern p'. A metaterm t is called p-linear iff every metapattern p in t is linear.*

Let us illustrate the use of our syntax by considering the `fibonacci` function specified by the equations `fib(0)=0, fib(1)=1` and `fib(x+2)=fib(x)+fib(x+1)`.

Using a choice variable a of arity 3 to encode the three different choices given by the previous specification, one possible specification of `fib` in our syntax is:

$$fib(M) \longrightarrow\ app(\lambda\, a\langle 0, s(0), s(s(x))\rangle.a\langle 0, s(0), app(fib,x) + app(fib, s(x))\rangle, M)$$

where *app* is the application symbol, λ is the classical function binder, and natural numbers are encoded by 0, $s(0)$, $s(s(0))$, ...

A position is a word over the alphabet $\mathbb{N}$; we use ϵ to denote the empty word. The *set of positions* of a metaterm t, denoted $\mathcal{POS}(t)$, is defined as usual [1] except for the term $s\{p\ \texttt{by}\ u\}$ for which we have $1.1.q \in \mathcal{POS}(s\{p\ \texttt{by}\ u\})$ if $q \in \mathcal{POS}(s)$ and $2.q \in \mathcal{POS}(s\{p\ \texttt{by}\ u\})$ if $q \in \mathcal{POS}(u)$ (see also [4] and [10]). The justification of this case comes from the fact that $s\{p\ \texttt{by}\ u\}$ is informally considered as "$app(\mu p.s, u)$" when reasoning about positions. The *submetaterm* of t at position p is written as $t|_p$. When $t|_p = u$, we will say that p is an *occurrence* of u in t.

The following notion is used to describe the set of variables/metavariables appearing along a given path which will play latter a role of "bound" objects in a terms/metaterms.

Definition 2 (Parameter Path). *Given a metaterm s and $p \in \mathcal{POS}(s)$, we define the parameter path of s at position q, written $\mathcal{PP}(s,q)$, as the following subset of variables and metavariables of s:*

$$\begin{array}{ll}
\mathcal{PP}(s,\epsilon) & = \emptyset \\
\mathcal{PP}(f(s_1,\ldots,s_n),i.q) & = \mathcal{PP}(s_i,q),\ \textit{for } i \in \{1\ldots n\} \\
\mathcal{PP}(a\langle s_1,\ldots,s_n\rangle,i.q) & = \mathcal{PP}(s_i,q),\ \textit{for } i \in \{1\ldots n\} \\
\mathcal{PP}(\mu p.s,1.q) & = \mathcal{V}ar(p) \cup \mathcal{MV}(p) \cup \mathcal{PP}(s,q) \\
\mathcal{PP}(u\{p \texttt{ by } v\},1.1.q) & = \mathcal{V}ar(p) \cup \mathcal{MV}(p) \cup \mathcal{PP}(u,q) \\
\mathcal{PP}(u\{p \texttt{ by } v\},2.q) & = \mathcal{PP}(v,q)
\end{array}$$

As an example, if $t = M\{g(\mathbb{X},x) \texttt{ by } \mu a\langle \mathbb{Y}, s(\mathbb{Y})\rangle.N\}$, then we have $\mathcal{PP}(t,2) = \emptyset$, $\mathcal{PP}(t,1.1) = \{\mathbb{X},x\}$, and $\mathcal{PP}(t,2.1) = \{\mathbb{Y},a\}$.

We assume that different "non parallel" metapatterns appearing on a same path cannot share (meta)variables. Thus for example, $\mu\mathbb{X}.\lambda\mathbb{X}.M$ or $\lambda x.\mu x.M$ are not allowed but the metaterm t given above is allowed. This is just a generalization of what is called "Barendregt's convention on bound variables".

The set of *free (meta)variables* of a metaterm t, written $\mathcal{FV}(t)$, is defined as usual. All the variables appearing in a metaterm t that are not free are called *bound variables*. Without loss of generality we assume the sets of free and bound variables to be disjoint. We work modulo α-conversion on preterms, so that renaming of bound variables is used when necessary to avoid clashes. Thus for example $\mu a\langle x,y,z\rangle.a\langle x,x,v\rangle =_\alpha \mu b\langle x',y',z'\rangle.b\langle x',x',v\rangle$.

Definition 3 (Well-formed metaterm). *A metaterm t is well-formed iff t has no free occurrences of choice/usual variables.*

The metaterms $\mu x.M$, $\mu\mathbb{X}.M$, $\mu x.f(M,x)$ and $\mu a\langle x,y\rangle.a\langle x,y\rangle$ are well-formed while $f(a\langle g,g\rangle)$ and $f(x)$ are not.

The following notion is used to talk about the free variables of a term which remain *after* a given choice on a choice variable.

Definition 4 (Localized Free Variables). *Given $a \in \mathcal{CV}$, $i \geq 1$ and a preterm t, the set $\mathcal{FV}^i_a(t)$ of* localized free variable *of t can be defined as usually done for the set of free variables except for the following case:*

$$\mathcal{FV}^i_a(a\langle t_1,\ldots,t_n\rangle) = \mathcal{FV}^i_a(t_i) \textit{ if } 1 \leq i \leq n$$

Indeed, $\mathcal{FV}^i_a(b\langle x,y,z\rangle) = \{b,z,x,y\}$ for any i and $\mathcal{FV}^1_a(a\langle x,y,z\rangle) = \{x\}$. Moreover, as we work modulo α-conversion we have $\mathcal{FV}^1_a(\mu a\langle x,y\rangle.a\langle f(x,z),u\rangle) = \mathcal{FV}^1_a(\mu b\langle x,y\rangle.b\langle f(x,z),u\rangle) = \{z,u\}$.

Definition 5 (Acceptable preterms). *Acceptability is the least relation on preterms containing the variables such that:*

- *If $t_1,\ldots,t_n$ are acceptable, then $f(t_1,\ldots,t_n)$ and $a\langle t_1,\ldots,t_n\rangle$ are acceptable for any $f \in \mathcal{F}$ and any $a \in \mathcal{CV}$.*

– *If t is acceptable and p is a pattern, then for all $a\langle p_1, \ldots, p_n\rangle \in p$, for all $i \in 1 \ldots n$, and for all $j \neq i$ such that $(\mathcal{FV}_a^j(t) \setminus \mathcal{V}ar(p_j)) \cap \mathcal{V}ar(p_i) = \emptyset$, we have that $\mu p.t$ is an acceptable term.*
– *If $\mu p.t$ and u are acceptable, then $t\{p \texttt{ by } u\}$ is acceptable.*

The role of acceptability is to prevent the creation of new free variables during evaluation. Indeed, the terms $\mu a\langle x, x\rangle.a\langle x, x\rangle$ and $\mu a\langle x, y\rangle.a\langle x, y\rangle$ are acceptable while $\mu a\langle x, y\rangle.b\langle x, y\rangle$ is not since $\mathcal{FV}_a^1(b\langle x, y\rangle) \setminus \mathcal{V}ar(x) = \{y, b\}$ and $\{y, b\} \cap \mathcal{V}ar(y) = \{y\}$. The term $\mu a\langle x, y\rangle.a\langle y, x\rangle$ is neither acceptable: if we choose the first branch x in the pattern $a\langle x, y\rangle$ we have then to consequently choose the first branch y in the term $a\langle y, x\rangle$, thus y becomes a *new* free variable.

Precontexts are preterms with one (and only one) occurrence containing a distinguished constant called a "hole" (and denoted $\Box$). A *context* is a precontext with no occurrence of the pattern-matching constructor. We remark that the notion of acceptability is not closed by precontexts as for example the preterm $a\langle x, y\rangle$ is acceptable but $\lambda a\langle y, x\rangle.a\langle x, y\rangle$ is not.

Definition 6 (Metasubstitutions/Substitutions). *A metasubstitution θ is a pair (θ_m, θ_v), with θ_m a denumerable set of pairs $\mathbb{X} \triangleright p$ and $M \triangleright t$, and θ_v a denumerable set of pairs $x \triangleright t$ and $a \triangleright i$, where t is a term, i is a natural number and p is a pattern. Application of θ to a (meta)variable $\widehat{x}$ is defined as $\theta\widehat{x} = o$ if $\widehat{x} \triangleright o \in \theta$, and $\theta\widehat{x} = \widehat{x}$, otherwise. We define id as the empty substitution, i.e. $id\widehat{x} = \widehat{x}$ for every $\widehat{x}$. The domain of θ is given by $Dom(\theta) = \{\widehat{x} \mid \widehat{x} \triangleright o \in \theta \text{ and } o \neq \widehat{x}\}$. A substitution θ is a metasubstitution such that $Dom(\theta_m) = \emptyset$. A metasubstitution $\theta = (\theta_m, \theta_v)$ is said to be well-formed iff $(\bigcup_{M \in Dom(\theta_m)} \mathcal{FV}(\theta_m M)) \cap Dom(\theta_v) = \emptyset$.*

The *union* of two well-formed metasubstitutions θ_1 and θ_2 is denoted by $\theta_1 \sqcup \theta_2$. This union is *only defined* if the resulting metasubstitution is well-formed and if for every (meta)variable $\widehat{x} \in Dom(\theta_1) \cap Dom(\theta_2)$ we have $\theta_1\widehat{x} = \theta_2\widehat{x}$.

We are now ready to define the notion of *pattern-matching*. This operation is not defined in general as a function from patterns and terms to substitutions but from patterns and terms to *sets* of substitutions. We will see latter how to ensure the uniqueness of this result.

Definition 7 (Pattern-matching). *For each pair (p, t), where p is a pattern and t is a term, we associate a set of substitutions as follows:*

$$
\begin{array}{lll}
id & \in \{\!\{ \_ \texttt{ by } t \}\!\} & \\
\{x \triangleright t\} & \in \{\!\{ x \texttt{ by } t \}\!\} & \\
\theta_1 \sqcup \ldots \sqcup \theta_n & \in \{\!\{ @(p_1, \ldots, p_n) \texttt{ by } t \}\!\} & \text{if } \theta_i \in \{\!\{ p_i \texttt{ by } t \}\!\} \\
\theta_1 \sqcup \ldots \sqcup \theta_n & \in \{\!\{ f(p_1 \ldots p_n) \texttt{ by } f(t_1 \ldots t_n) \}\!\} & \text{if } \theta_i \in \{\!\{ p_i \texttt{ by } t_i \}\!\} \\
\{a \triangleright i\} \sqcup \theta_i & \in \{\!\{ a\langle p_1 \ldots p_n\rangle \texttt{ by } t \}\!\} & \text{if } \theta_i \in \{\!\{ p_i \texttt{ by } t \}\!\}
\end{array}
$$

We remark that in the last three cases the result of $\{\!\{p \texttt{ by } t\}\!\}$ is defined only if $\sqcup$ is defined. Also, all the substitutions in $\{\!\{p \texttt{ by } t\}\!\}$ are well-formed since

they do not map metavariables. When $\{\!\{p \texttt{ by } t\}\!\}$ is a singleton we will make an abuse of notation by writing $\{\!\{p \texttt{ by } t\}\!\}$ to denote the only element of this set.

As an example of the previous definition, the pattern-matching $\{\!\{a\langle 0, x\rangle \texttt{ by } 0\}\!\}$ has two solutions: $\{a \triangleright 1\}$ and $\{a \triangleright 2, x \triangleright 0\}$. This comes from the fact that the pattern $a\langle 0, x\rangle$ contains two "overlapping" subpatterns 0 and x.

Definition 8 (Acceptable/linear metasubstitution). *A metasubstitution θ is said to be acceptable (resp. linear) iff for every (meta)variable $\widehat{x} \in Dom(\theta)$, $\theta\widehat{x}$ is acceptable (resp. linear).*

It is time to make the point w.r.t capture of variables in higher-order rewriting.

In CRS [15, 16] for example, a metaterm like $\lambda x.M(x)$ allows the (eventual) capture of the variable x while $\lambda x.M$ does not. In this formalism the β-rule has to be written as $app(\lambda x.M(x), N) \longrightarrow M(N)$ which does not correspond to the traditional way to express the β-rule.

In ERS [13] there is a metasubstitution operator which allows to express the β-rule in a more traditional way as $app(\lambda x.M, N) \longrightarrow M\{x/N\}$. The instantiation of the metavariable M may or may not capture the variable x. However, we cannot assume α-conversion on metaterms in this formalism: if we suppose $\lambda x.M =_\alpha \lambda y.M$, then the instantiation of M by x will give two non α-equivalent terms $\lambda x.x \neq_\alpha \lambda y.x$. In order to properly handle α-conversion on terms but not on metaterms two *different* levels of syntax are needed, and this is the approach taken in general in the ERS formalism (see also [4]).

To allow α-conversion on the level of terms but not on that of metaterms a special notion of instantiation is needed, so that application of a metasubstitution $\theta = (\theta_m, \theta_v)$ to a metaterm will be split into two different steps: θ_m is used as *first-order* replacement, so that capture of variables can be provoked, while θ_v is used as *higher-order* substitution, so that no capture of variables is possible.

Definition 9 (Applying a metasubstitution). *Given a metasubstitution $\theta = (\theta_m, \theta_v)$ and a metaterm t, the application of θ to t (or instantiation of t by θ) yields a set of terms, written $\theta(t)$, which is computed in two steps:*

1. *First compute the first-order replacement $\theta_m(t)$ obtaining a preterm s (in the case where $\theta_m(t)$ is not still a preterm the application is not defined)*
2. *Then compute the set of terms $\theta_v(s)$, where θ_v is a higher-order substitution which works modulo α-conversion defined as follows:*

$\theta_v x$	$\in \theta_v(x)$	*if* $x \in Dom(\theta_v)$
x	$\in \theta_v(x)$	*if* $x \notin Dom(\theta_v)$
$\mu p.t'$	$\in \theta_v(\mu p.t)$	*if* $t' \in \theta_v(t)$ *and no capture of variables holds*
$f(t'_1, \ldots, t'_n)$	$\in \theta_v(f(t_1, \ldots, t_n))$	*if* $t'_i \in \theta_v(t_i)$
t'_i	$\in \theta_v(a\langle t_1, \ldots, t_n\rangle)$	*if* $\theta_v a = i$ *and* $t'_i \in \theta_v(t_i)$

$$\begin{array}{lll} a\langle t'_1,\ldots,t'_n\rangle \in \theta_v(a\langle t_1,\ldots,t_n\rangle) & \textit{if } t'_i \in \theta_v(t_i) \textit{ and } a \notin \mathit{Dom}(\theta_v) \\ t' \qquad\qquad\quad \in \theta_v(t\{p \texttt{ by } u\}) & \textit{if } u' \in \theta_v(u), \theta'_v \in \{\!\{p \texttt{ by } u'\}\!\}, \\ & \quad t' \in (\theta'_v \sqcup \theta_v)(t) \\ & \quad \textit{and no capture of variables holds} \end{array}$$

When the metasubstitution θ is a substitution, we may make an abuse of notation by writing $\theta(t)$ instead of $\theta_v(t)$. We also remark that if a metaterm t has no pattern-matching constructor, then, if defined, $\theta(t)$ is a singleton.

Another interesting observation is that even when θ_v of some metasubstitution θ is empty, the second step of the previous procedure must be computed on preterms (which still have pattern-matching constructors to be eliminated).

Let us see how the application of a metasubstitution works on an example. Consider $\theta = (\theta_m, \theta_v)$, where $\theta_m = \{\mathbb{X}/a\langle x, f(z,y)\rangle, M/a\langle g(x,x), z\rangle, N/f(x,x)\}$ and $\theta_v = \emptyset$. In order to compute $\theta(M\{\mathbb{X} \texttt{ by } N\})$ we first compute $\theta_m(M\{\mathbb{X} \texttt{ by } N\})$ which gives the preterm $t = a\langle g(x,x), z\rangle\{a\langle x, f(z,y)\rangle \texttt{ by } f(x,x)\}$.

Now, since α-conversion is allowed on preterms, we obtain

$$t =_\alpha a\langle g(x',x'), z\rangle\{a\langle x', f(z,y)\rangle \texttt{ by } f(x,x)\} = t'$$

Now, the computation of $\{\!\{a\langle x', f(z,y)\rangle \texttt{ by } f(x,x)\}\!\}$ gives $\{\rho_1, \rho_2\}$, where $\rho_1 = \{a \triangleright 1, x' \triangleright f(x,x)\}$ and $\rho_2 = \{a \triangleright 2, z \triangleright x, y \triangleright x\}$, and thus, the second step of the application procedure finally gives a set

$$\theta_v(t') = \{g(f(x,x), f(x,x)), x\}$$

Lemma 1. *If t is acceptable and $\theta \in \{\!\{p \texttt{ by } t\}\!\}$, then θ is acceptable. Also, if t and θ are acceptable then so is $\theta(t)$.*

2 Rewrite rules and reduction relation

This section introduces the precise syntax used to specify rewrite rules in the ERSP formalism as well as the reduction relation associated to them.

Definition 10. *An Expression Reduction System with Patterns (ERSP) is a set of rewrite rules of the form $l \longrightarrow r$ (written also (l,r)) such that:*

- *l and r are well-formed metaterms,*
- *the first symbol (called head symbol) in l is in $\mathcal{F} \cup \mathcal{B}$,*
- *$\mathcal{MV}(r) \subseteq \mathcal{MV}(l)$,*
- *$\mathcal{FV}(r) \subseteq \mathcal{FV}(l)$, and*
- *l contains no occurrence of the pattern-matching constructor.*

Thus for example, the rule $app(\lambda\mathbb{X}.M, N) \longrightarrow M\{\mathbb{X} \texttt{ by } N\}$ given in the introduction, which generalizes the classical β-rule to the case of patterns, belongs to our framework.

In order to be able to guarantee that no free variable is "generated" during reduction the following notion will be necessary.

Definition 11 (Path condition). *Let M be a term variable and t be a metaterm. We consider all the occurrences $p_1, \ldots, p_n$ of M in t and their corresponding parameter paths $l_1, \ldots, l_n$. A metasubstitution θ is said to have the path condition property for M in t iff:*

$$\forall \widehat{x} \in \mathcal{FV}(\theta_m M), (\forall 1 \leq i \leq n, \widehat{x} \in \theta_m l_i) \vee (\forall 1 \leq i \leq n, \widehat{x} \notin \theta_m l_i)$$

where the notation $\theta_m l$ denotes the set $\bigcup_{\widehat{x} \in l} \theta_m \widehat{x}$.

This notion is extended to rewrite rules by saying that θ has the path condition for M in (l, r) iff it has the path condition for M in $f(l, r)$, where f is any binary function symbol. This trick is used to consider a rule as a unique "tree".

The classical example of path condition which is not satisfied for a rewrite rule is given by the η-rule of the λ-calculus (see for example [13, 4]). Another rule in the same spirit but using patterns is $\lambda f(\mathbb{X}).M \longrightarrow M$. The metasubstitution $\theta = \{\mathbb{X} \triangleright x, M \triangleright x\}$ does not satisfy the path condition for M in this rule.

We now define the set of "good" substitutions to instantiate rewrite rules. For that we remark that given a rewrite rule $l \longrightarrow r$, the metaterm l does not contain the pattern-matching constructor, so that for any metasubstitution θ the term $\theta(l)$ is a singleton.

Definition 12 (Admissible metasubstitution for metaterms/rules). *A metasubstitution θ is admissible for a metaterm t iff*

- *$\theta(t)$ contains only acceptable terms*
- *θ has the path condition for every term metavariable appearing in t.*

A metasubstitution θ is admissible for a rule (l, r) iff θ is admissible for $f(l, r)$, where f is any binary function symbol.

We remark that this definition implies that given a rule (l, r) both $\theta(l)$ and $\theta(r)$ are defined, so in particular all the pattern/term metavariables in l are also in $Dom(\theta_m)$.

Definition 13 (Admissible reduction relation). *Let $\mathcal{R}$ be a ERSP. We say that s rewrites to t, written $s \longrightarrow_{\mathcal{R}} t$ (or $s \xrightarrow{a}_{\mathcal{R}} t$ when the distinction must be done), iff there exists a rule $(l, r) \in \mathcal{R}$, a well-formed admissible metasubstitution θ for (l, r) and a context C such that $s = C[\theta(l)]$ and $t \in C[\theta(r)]$.*

Even if the relation $\longrightarrow_{\mathcal{R}}$ is defined on any kind of terms, the reduction can only take place on acceptable subterms.

As expected, the relation reduction enjoys good preservation properties.

Lemma 2. *Assume $s \longrightarrow_{\mathcal{R}} t$. Then $\mathcal{FV}^i_a(t) \subseteq \mathcal{FV}^i_a(s)$ (for any a and any i) and $\mathcal{FV}(t) \subseteq \mathcal{FV}(s)$. Also, if s is acceptable, then so is t.*

3 A complete example

We consider in this section the well known higher-order function *map* which takes a function f and a list l and returns the result of applying f to each element of l. This function can be specified as the following Ocaml [18] program:

```
#let rec map(f,l) =  match l with
                      Nil -> Nil
                    | Cons(h,t) -> Cons(f(h),map(f,t));;
```

It can also be specified as the following ERSP rewrite rule:

$$map(\mu\mathbb{X}.F, L) \longrightarrow \\ a\langle nil, cons(F\{\mathbb{X} \texttt{ by } h\}, map(\mu\mathbb{X}.F, t))\rangle\{a\langle nil, cons(h,t)\rangle \texttt{ by } L\}$$

Let us see how this implementation of *map* works on a concrete example. Suppose that we represent natural numbers with constructors 0 and s and let us consider $pred =_{def} \mu b\langle 0, s(n)\rangle.b\langle 0, n\rangle$ in order to denote the predecessor function on natural numbers. Using the metasubstitution $\theta = (\theta_m, \theta_v)$, where $\theta_m = \{\mathbb{X} \triangleright b\langle 0, s(n)\rangle, F \triangleright b\langle 0, n\rangle, L \triangleright cons(0, cons(s(s(0)), nil))\}$ and $\theta_v = \emptyset$ to fire the previous rewrite rule, we can construct the following derivation:

$$t_1 = map(pred, cons(0, cons(s(s(0)), nil))) \longrightarrow \\ t_2 = cons(0, map(pred, cons(s(s(0)), nil)))$$

Indeed, the term t_1 is an instance of the left-hand side of the previous rule. In order to obtain t_2 we have to apply θ to the right-hand side of the previous rule. For that, we first instantiate $\{a\langle nil, cons(h,t)\rangle \texttt{ by } L\}$ with θ_m, then since $\theta_v(\theta_m(L)) = \theta_m(L)$, we can compute the pattern-matching operation $\{\!\{a\langle nil, cons(h,t)\rangle \texttt{ by } cons(0, cons(s(s(0)), nil))\}\!\}$. We then obtain a substitution $\theta'_v = \{a \triangleright 2, h \triangleright 0, t \triangleright cons(s(s(0)), nil)\}$. Now, we have to instantiate $a\langle nil, cons(F\{\mathbb{X} \texttt{ by } h\}, map(\mu\mathbb{X}.F, t))\rangle$ with θ_m, then proceed with the application of θ'_v to this last instantiation. The only delicate part is the one concerning the submetaterm $F\{\mathbb{X} \texttt{ by } h\}$. We have $\theta_m(F\{\mathbb{X} \texttt{ by } h\}) = b\langle 0, n\rangle\{b\langle 0, s(n)\rangle \texttt{ by } h\} = t'$ and $\theta'_v(t') = b\langle 0, n\rangle\{\!\{b\langle 0, s(n)\rangle \texttt{ by } 0\}\!\} = 0$.

The reader may verify that this sequence of operations finally leads to the term t_2. Similarly, we can then continue the reduction till $cons(0, cons(s(0), nil))$.

4 Towards a subclass of confluent ERSP

The following two sections are devoted to the study of confluence for a certain class of ERSP which are called the *orthogonal l-constructor* ERSP, and a certain class of terms, which are called *l-constructor deterministic terms*. Intuitively, an orthogonal ERSP is *left-linear* and *not overlapping*. Sufficiency of orthogonality for confluence in first and higher-order rewrite systems is well-known [1]. A constructor ERSP is a system $\mathcal{R}$ where the set of function symbols is partitioned

into two different subsets, namely, the set of *constructors*, which cannot be reduced, and the set of *defined symbols*, which cannot be matched. As an example, let us consider the following system which is not a constructor ERSP.

$$\mathcal{R}: \quad a \longrightarrow b,\ app(\mu a.c, M) \longrightarrow c\{a \texttt{ by } M\}$$

The term $app(\mu a.c, a)$ can be reduced to both $app(\mu a.c, b)$ and c which are not joinable. Thus, $\mathcal{R}$ turns out to be non confluent.

Unfortunately, *orthogonal l-constructor* ERSP do not immediately guarantee confluence as the rule $app(\lambda\mathbb{X}.M, N) \longrightarrow M\{\mathbb{X} \texttt{ by } N\}$ shows: the term $t = app(\lambda a\langle x, y\rangle.a\langle 0, 1\rangle, 3)$ has two non-joinable reducts 0 and 1 by this unique rule.

The reason is that t contains two "overlapping" patterns x and y inside the choice pattern $a\langle x, y\rangle$. The failure of the confluence property in this case is completely natural since the term t corresponds, informally, to a "non-orthogonal" first-order rewriting system. It is then clear that we have to get rid of this class of terms in order to get a confluence result, this will be done by introducing the notion of *l-constructor deterministic* terms.

We are know ready to give a formal definition of all these notions.

Definition 14 (L-constructor system). *$\mathcal{R}$ is said to be l-constructor iff*

- *The set $\mathcal{F}$ of function symbols can be partitioned into two sets $\mathcal{F}_c$ and $\mathcal{F}_d$, called respectively constructors and defined symbols, such that:*
  - *Each defined symbol is the head of some left-hand side of $\mathcal{R}$.*
  - *All the function symbols in metapatterns of $\mathcal{R}$ are constructors.*
- *For every rule $(l, r) \in \mathcal{R}$, both l and r are p-linear metaterms.*

The system $\mathcal{R}_1 = \{\beta_{PM}\} \cup \{0 + N \longrightarrow N, s(M) + N \longrightarrow s(M + N)\}$ is l-constructor. The system $\mathcal{R}_2 = \{\mu f(\mathbb{X}).M \longrightarrow M, f(0) \longrightarrow 0\}$ is not l-constructor since the function symbol f appears as the head symbol of some rule and inside a metapattern of $\mathcal{R}$. The system $\mathcal{R}_3 = \{\mu f(\mathbb{X}, \mathbb{X}).0 \longrightarrow 0\}$ is not l-constructor since $\mu f(\mathbb{X}, \mathbb{X}).0$ is not p-linear.

Definition 15 (L-constructor objects). *Given a l-constructor system $\mathcal{R}$, we say that a metapattern is l-constructor iff it is linear and all its function symbols are constructors of $\mathcal{R}$. A l-constructor metaterm contains only l-constructor metapatterns. A metasubstitution θ is said to be l-constructor iff $\theta(\widehat{x})$ is l-constructor for any $\widehat{x} \in Dom(\theta)$.*

As an example concerning our previous system $\mathcal{R}_1$, we can observe that the metapattern $s(\mathbb{X})$ is l-constructor but $\mathbb{X} + \mathbb{Y}$ is not since the symbol $+$ is not a constructor function symbol.

We remark that if p is a l-constructor pattern and $\{\!\{p \texttt{ by } t\}\!\}$ is defined then all its elements are l-constructor substitutions.

Definition 16 (L-constructor reduction relation). *If $\mathcal{R}$ is a l-constructor system, we say that s constructor rewrites to t (written $s \xrightarrow{c}_{\mathcal{R}} t$) iff there exists a rewrite rule $(l, r) \in \mathcal{R}$, a well-formed l-constructor metasubstitution θ admissible for (l, r) and a context C such that $s = C[\theta(l)]$ and $t \in C[\theta(r)]$.*

As an example, given the previous system $\mathcal{R}_1$, we have $0+0 \xrightarrow{c}_{\mathcal{R}_1} 0$ but we do not have $t = app(\lambda(0+0).3, 0+0) \xrightarrow{c}_{\mathcal{R}_1} 3$ (even if we have $t \xrightarrow{a}_{\mathcal{R}_1} 3$) since the term t is not an l-constructor term.

One can remark that whenever t is a l-constructor preterm and θ is a l-constructor substitution w.r.t. $\mathcal{FV}(t)$, then $\theta(t)$ contains only l-constructor terms. Also as expected, l-constructor terms are preserved during reduction.

Lemma 3 (Preservation of l-constructor terms). *If $\mathcal{R}$ is l-constructor, s is l-constructor and $s \xrightarrow{c}_{\mathcal{R}} t$, then t is l-constructor.*

Definition 17 (Left linear systems). *A rewrite rule $l \longrightarrow r$ is said to be left linear iff l contains at most one occurrence of any term metavariable. A system is left linear if all its rule are left linear.*

As an example, the rule $f(M, M) \longrightarrow 3$ is not left linear while $f(M) \longrightarrow g(M, M)$ and $\mu x.f(x, x) \longrightarrow 0$ are.

Definition 18 (Redexes and overlapping redexes). *A term t is said to be a redex if it is an instance of some left-hand side of rule that is $t = \theta(l)$ for some rule (l, r). A rewrite system is said to be non-overlapping iff*

- *Whenever a redex $\theta(l_j)$ contains another redex $\theta'(l_i)$, then $\theta'(l_i)$ must be contained in $\theta(M)$ for some term metavariable M of l_j.*
- *Likewise whenever a redex $\theta(l)$ properly contains another redex instance $\theta'(l)$ of the same rule.*

Definition 19 (Orthogonal systems). *A rewrite system $\mathcal{R}$ is said to be orthogonal iff $\mathcal{R}$ is left-linear and non-overlapping.*

As an example, the system $\{f(\mu x.x) \longrightarrow 0, \mu\mathbb{X}.y \longrightarrow 1\}$ is overlapping : the redex $f(\mu y.y) = \theta(f(\mu x.x))$ contains the redex $\mu y.y = \theta'(\mu\mathbb{X}.y)$. The system $\{f(\mu\mathbb{X}.M) \longrightarrow 0, \lambda\mathbb{Z}.N \longrightarrow g(2)\}$ is orthogonal.

We now introduce *l-constructor deterministic terms* for which the class of orthogonal l-constructor ERSP will be confluent. Let us start by the following notion.

Definition 20 (Overlapping patterns). *Two patterns p and q are said to be overlapping iff there exists a term t s.t. both $\{\!\{p \texttt{ by } t\}\!\}$ and $\{\!\{q \texttt{ by } t\}\!\}$ are defined.*

The patterns $f(\_, x)$ and $f(y, g(0))$ are overlapping. Also $a\langle 0, s(x)\rangle$ and $b\langle s(0), s(s(\_))\rangle$ are overlapping.

Definition 21 (Deterministic patterns/terms). *A pattern p is said to be deterministic iff whenever $a\langle p_1, \ldots, p_n\rangle$ appears inside p, then for all $i \neq j$ the patterns p_i and p_j are not overlapping. A term t is said to be a deterministic iff t is acceptable and for every pattern p appearing in t, p is deterministic.*

Thus for example, $b\langle s(0), s(s(\_))\rangle$ is deterministic but $b\langle s(0), s(\_)\rangle$ is not. This definition implies that whenever p is a deterministic pattern, then there exists *at most* one substitution θ belonging to $\{\!\{p \texttt{ by } t\}\!\}$.

Definition 22 (Deterministic metasubstitution for metaterms/rules). *A metasubstitution θ is said to be deterministic for a metaterm t iff*

- *θ is admissible for t,*
- *$\theta(t)$ contains only one deterministic term,*

Finally, θ is deterministic for a rule (l, r) iff θ is a deterministic for $f(l, r)$, where f is any binary function symbol.

A *metasubstitution* θ is *deterministic* iff $\theta\widehat{x}$ is deterministic $\forall \widehat{x} \in Dom(\theta)$. Deterministic terms are stable by deterministic substitutions. Also, the substitution obtained by performing a pattern-matching operation on a deterministic pattern p and a deterministic term t turns out to be deterministic.

Definition 23 (Deterministic reduction relation). *Given a system $\mathcal{R}$, we say that s deterministicly rewrites to t (written $s \xrightarrow{d}_{\mathcal{R}} t$) iff there exists a rewrite rule $(l, r) \in \mathcal{R}$, a well-formed deterministic metasubstitution θ for (l, r) and a context C such that $s = C[\theta(l)]$ and $s = C[\theta(r)]$.*

From now on we use the notation $\xrightarrow{c,d}_{\mathcal{R}}$ to denote $\xrightarrow{c}_{\mathcal{R}} \cap \xrightarrow{d}_{\mathcal{R}}$. As expected, orthogonal systems allows us to preserve deterministic terms.

Lemma 4 (Preservation of deterministic terms). *Given an orthogonal system $\mathcal{R}$, if s is deterministic and $s \xrightarrow{d}_{\mathcal{R}} t$, then t is deterministic.*

5 The confluence proof

This section is dedicated to show that the relation $\xrightarrow{c,d}_{\mathcal{R}}$ is confluent for orthogonal l-constructor ERSP, that is, orthogonal l-constructor ERSP are confluent on l-constructor deterministic terms. This confluence property can only be proved on the set of acceptable l-constructor deterministic terms. The proof uses a technique due to Tait and Martin-Löf [2] and can be summarized in four steps:

- We define a parallel reduction relation denoted $\ggg_{c,d}$.
- We prove that $\ggg_{c,d}{}^*$ and $\xrightarrow{c,d}{}^*$ are the same relation.
- We show, using Takahashi terms [19], that $\ggg_{c,d}$ has the diamond property on the set of acceptable l-constructor deterministic terms.
- We conclude by the fact that the diamond property implies confluence.

In order to define the $\ggg_{c,d}$ reduction relation, we first need to extend relations on terms to relations on substitutions.

Notation 1 Given any relation $\rightsquigarrow$ between terms and given two metasubstitutions θ and θ', we write $\theta \rightsquigarrow \theta'$ when $Dom(\theta) = Dom(\theta')$, θ and θ' coincide on the sets of choice and pattern variables, and for every $M \in Dom(\theta)$ we have $\theta(M) \rightsquigarrow \theta'(M)$ and $x \in Dom(\theta)$ we have $\theta(x) \rightsquigarrow \theta'(x)$.

In order to relate two sets S and T of metaterms/metasubstitutions via any binary relation $\rightsquigarrow$, we will write $S \rightsquigarrow T$ iff $\forall s \in S, \exists t \in T, s \rightsquigarrow t$ and $\forall t \in T, \exists s \in S, s \rightsquigarrow t$. For any unary relation Φ on metaterms/metasubstitutions, we will write $\Phi(S) = \bigcup_{s \in S} \Phi(s)$.

We can now define the parallel relation as follows:

Definition 24 (The simultaneous relation $\ggg_a$).

- $x \ggg_a x$.
- *If* $s_1 \ggg_a s'_1, \ldots, s_m \ggg_a s'_m$, *then* $f(s_1, \ldots, s_m) \ggg_a f(s'_1, \ldots, s'_m)$, $\mu p.s_1 \ggg_a \mu p.s'_1$ *and* $a\langle s_1, \ldots, s_m\rangle \ggg_a a\langle s'_1, \ldots, s'_m\rangle$.
- *If* $\theta \ggg_a \rho$, *then* $\theta(l) \ggg_a \rho(r)$ *for any rewrite rule* $l \longrightarrow r$ *such that* θ *is admissible for it.*

We will denote $s \ggg_c t$ *(resp.* $s \ggg_d t$*) if the metasubstitution* θ *used in the Definition 24 is not only admissible but also constructor (resp. deterministic). We write* $\ggg_{c,d}$ *to denote the relation* $\ggg_c \cap \ggg_d$.

Given $\ggg \in \{\ggg_a, \ggg_c, \ggg_d\}$, then for every term s we have $s \ggg s$. Also, if $s \ggg s'$ and s is not a redex, then the root symbols of s and s' are the same. Moreover, if also $s = \mathcal{G}(s_1, \ldots, s_m)$, then $s' = \mathcal{G}(s'_1, \ldots, s'_m)$, where $s_i \ggg s'_i$, for $\mathcal{G}$ a function symbol, a case or a binder symbol.

We are now ready to proceed with the second step of Tait and Martin-Löf's technique which consists in showing that the reflexive-transitive closures of $\xrightarrow{c,d}$ and $\ggg_{c,d}$ are the same relation.

We can show by induction on l-constructor patterns that pattern-matching is stable by term reduction.

Lemma 5. *Let* $\mathcal{R}$ *be a l-constructor ERSP,* p *a l-constructor pattern and* t *a term such that* $t \xrightarrow{c}{}^*_{\mathcal{R}} t'$. *If* $\{\!\{p \texttt{ by } t\}\!\}$ *is defined, then* $\{\!\{p \texttt{ by } t'\}\!\}$ *is also defined and* $\{\!\{p \texttt{ by } t\}\!\} \xrightarrow{c}{}^*_{\mathcal{R}} \{\!\{p \texttt{ by } t'\}\!\}$.

The previous Lemma does not holds if the pattern p is not *l-constructor*. Indeed, let $\mathcal{R} = \{a \longrightarrow b\}$ and let us take the non-linear pattern $p_1 = f(x, x)$ and the non-constructor pattern $p_2 = a$. We have that $\{\!\{f(x, x) \texttt{ by } f(a, a)\}\!\}$ is defined and $f(a, a) \xrightarrow{c}_{\mathcal{R}} f(a, b)$ but $\{\!\{f(x, x) \texttt{ by } f(a, b)\}\!\}$ is not defined. Also, $\{\!\{a \texttt{ by } a\}\!\}$ is defined and $a \xrightarrow{c}_{\mathcal{R}} b$ but $\{\!\{a \texttt{ by } b\}\!\}$ is not defined.

We can now show by induction on metaterms that reduction of metasubstitutions is stable by application:

Lemma 6. *Let* $\mathcal{R}$ *be an l-constructor ERSP and* θ, ρ *two well-formed l-constructor metasubstitutions such that* $\theta \xrightarrow{c}{}^*_{\mathcal{R}} \rho$. *If for any l-constructor metaterm* t *such that* θ *has the path condition for* t, $\theta(t)$ *is defined and contains only acceptable l-constructor terms, then the same happens for* $\rho(t)$ *and* $\theta(t) \xrightarrow{c}{}^*_{\mathcal{R}} \rho(t)$.

Lemma 6 allows us to obtain the following fundamental property.

Lemma 7. *Let* $\mathcal{R}$ *be a l-constructor ERSP. If* $\theta \xrightarrow{c}{}^*_{\mathcal{R}} \rho$, *and* θ *is a well-formed l-constructor admissible metasubstitution for* (l, r), *then* $\theta(l) \xrightarrow{c}{}^*_{\mathcal{R}} \rho(r)$ *and* ρ *is admissible for* (l, r).

We are now able to conclude the second step of our confluence proof:

Theorem 1 (Equivalence between $\ggg_{c,d}{}^*$ and $\xrightarrow{c,d}{}^*_{\mathcal{R}}$). *If $\mathcal{R}$ is a l-constructor ERSP, then $s \xrightarrow{c,d} t$ implies $s \ggg_{c,d} t$ and $s \ggg_{c,d} t$ implies $s \xrightarrow{c,d}{}^*_{\mathcal{R}} t$.*

Proof. The first implication holds in a more general case, namely, that $s \longrightarrow s'$ implies $s \ggg s'$ for $\longrightarrow \in \{\xrightarrow{a}, \xrightarrow{c}, \xrightarrow{d}\}$ and *any* ERSP $\mathcal{R}$. This can be shown by induction on the definition of $\ggg$. The second implication can be shown by induction on the structure of s using Lemma 7.

We are now going to prove the diamond property for the relation $\ggg_{c,d}$. For that, we associate a term $\#(s)$ to every term s such that every time $s \ggg_{c,d} s'$, for some s', we automatically deduce $s' \ggg_{c,d} \#(s)$. Thus, given two different terms s' and s'' such that $s \ggg_{c,d} s'$ and $s \ggg_{c,d} s''$, we obviously obtain a unique term $\#(s)$ which allows us to close the diagram with $s' \ggg_{c,d} \#(s)$ and $s'' \ggg_{c,d} \#(s)$. The diamond property will immediately follow.

Definition 25 (Takahashi terms $\#(s)$). *Given a ERSP $\mathcal{R}$ and a term s we define its associated Takahashi term $\#(s)$ by induction as follows:*

- *If $s = x$, then $\#(x) = x$.*
- *If $s = f(s_1, \ldots, s_m)$ (resp. $s = \mu p.s'$ or $a\langle s_1, \ldots, s_m\rangle$) and s is not a deterministic instance of a left-hand side of rule, then $\#(s) = f(\#(s_1), \ldots, \#(s_m))$ (resp. $\#(s) = \mu p.\#(s')$ and $\#(s) = a\langle\#(s_1), \ldots, \#(s_m)\rangle$).*
- *If s is an instance $\theta(l)$ of the left-hand side l of some rule $l \longrightarrow r$, where $\theta = (\theta_m, \theta_v)$ is deterministic for (l, r), then $\#(s) = \#(\theta)(r)$, where $\#(\theta) = (\#(\theta)_m, \#(\theta)_v)$ verifies $\#(\theta)_m(M) = \#(\theta_m(M))$ and $\#(\theta)_v(y) = \#(\theta_v(y))$.*

We can show by induction on terms the following two properties about $\ggg$.

Lemma 8. *The term $\#(s)$ is uniquely defined in every orthogonal ERSP.*

Lemma 9. *Let $\mathcal{R}$ be an orthogonal ERSP, let $\ggg \in \{\ggg_a, \ggg_c, \ggg_d\}$ and let $s \ggg s'$. If $s = \mathcal{G}(s_1, \ldots, s_m)$ and $s' = \mathcal{G}(s'_1, \ldots, s'_m)$ (where $\mathcal{G}$ is a function, binder symbol, case or variable) and $s_i \ggg s'_i$, then if s is a redex $\theta(l)$, s' is a also redex $\rho(l)$, for some ρ such that $\theta \ggg \rho$.*

Lemma 10. *If $\mathcal{R}$ is l-constructor and orthogonal and $s \ggg_{c,d} s'$, then $s' \ggg_{c,d} \#(s)$.*

Proof. By induction on the definition of $s \ggg_{c,d} s'$ using Lemma 9.

Corollary 1. *If $\mathcal{R}$ is an l-constructor orthogonal ERSP, then the relation $\ggg_{c,d}$ is confluent on deterministic constructor acceptable terms.*

Proof. It is well-known that a relation having the diamond property is confluent [1] so that it is sufficient to show that $\ggg$ has the diamond property. Let us consider $s \ggg s'$ and $s \ggg s''$. By Lemma 10 we can close the diagram with $s' \ggg \#(s)$ and $s'' \ggg \#(s)$.

By Theorem 1 and Corollary 1 we can now conclude with the main result of this section, namely,

Theorem 2. *Let $\mathcal{R}$ be an l-constructor orthogonal ERSP. The relation $\xrightarrow{c,d}_{\mathcal{R}}$ is confluent on acceptable constructor deterministic terms.*

6 Conclusion and further work

We have introduced a new *Higher-Order* formalism, called ERSP, in which the abstraction operation is not only allowed on variables but also on more complex patterns. This formalism can be seen as an extension of ERS [13] and SERS [4] to the case of patterns and an extension of [12] to the case of non functional rewriting rules. Many simple notions in the mentioned previous works do not trivially extend in our case: on one hand the complexity of ERSP does not only appear at the level of metaterms but also at the level of terms, on the other hand, binders are not always so simple as in the case of λ-calculus. We carefully extend all the expected notions of rewriting to our framework, namely, terms, metaterms, rewrite rules, substitutions, reduction, etc. The resulting formalism is able to model pattern-matching function/proof definitions.

The more technical part of this work is the identification of a class of ERSP which can be proved to be confluent on an appropriate set of terms. Our confluence result gives in particular a confluence result for SERS.

As mentioned in the introduction, ERSP and ρ-calculus [8] are closely related. The main difference between both formalisms lies in the class of syntactic patterns which are considered: our approach is mainly driven by the set of patterns appearing in functional languages and theorem provers, namely, algebraic patterns with choice constructors, while their approach includes other higher-order patterns (rules in their formalism) not really used in implementation of programming languages.

Many future directions remain to be explored. The first one consists in the definition of implementation languages given by "explicit" versions of this formalism, where both pattern matching and substitution operators are integrated to the syntax. This would result in generalizations of calculi defined in [5, 9].

Also, a formal comparison between ERSP and ρ-calculus must be done. In particular, it would be interesting to know if every system in the ρ-calculus can be expressed in a ERSP system and vice-versa. Another interesting question concerns confluence since confluent ρ-systems are characterized via a special *reduction strategy* while confluence is ensured in our case by syntactic restrictions.

Typing is another feature which remains as further work. It is however interesting to remark that the pioneer work on pattern calculi [12] which inspired the definition of ERSP was built, via the Curry-Howard isomorphism, on a computational interpretation of Gentzen sequent calculus for intuitionistic minimal logic. As a consequence, each ERSP pattern constructor comes from the interpretation of some *left* logical rule of Gentzen calculus. It is nevertheless less evident how to associate a Curry-Howard style interpretation to the entire ERSP syntax.

Last but not least, strong normalization of ERSP has to be studied. Indeed, proof techniques to guarantee termination of higher-order formalisms are not straightforward [3, 11, 21] and they do not extend immediately to our case.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984. Revised Edition.
3. F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-Data-Type Systems. *Theoretical Computer Science*, 277, 2001.
4. E. Bonelli, D. Kesner, and A. Ríos. A de Bruijn notation for higher-order rewriting. In *11th RTA*, *LNCS* 1833. 2000.
5. S. Cerrito and D. Kesner. Pattern matching as cut elimination. *Theoretical Computer Science*. To appear.
6. S. Cerrito and D. Kesner. Pattern matching as cut elimination. In *14th LICS*. IEEE. 1999.
7. H. Cirstea. *Calcul de réécriture : fondements et applications*. Thèse de doctorat, Université Henri Poincaré - Nancy 1, 2000.
8. H. Cirstea and C. Kirchner. ρ-calculus, the rewriting calculus. In *5th CCL*, 1998.
9. J. Forest. A weak calculus with explicit operators for pattern matching and substitution. In *13th RTA*, *LNCS* 2378. 2002.
10. J. Forest and D. Kesner. Expression Reduction Systems with Patterns, 2003. Available on `http://www.pps.jussieu.fr/~kesner/papers/`.
11. J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *14th LICS*. IEEE. 1999.
12. D. Kesner, L. Puel, and V. Tannen. A Typed Pattern Calculus. *Information and Computation*, 124(1), 1996.
13. Z. Khasidashvili. Expression reduction systems. In *Proceedings of IN Vekua Institute of Applied Mathematics*, volume 36, Tbilisi, 1990.
14. C. Kirchner, H. Cirstea and L. Liquori. The Rho Cube. In *FOSSACS'01*, *LNCS* 2030. 2001.
15. J.-W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, 1980. PhD Thesis.
16. J.-W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science* 121, 1993.
17. T. Nipkow. Higher-order critical pairs. In *6th LICS*. IEEE. 1991.
18. The Objective Caml language, `http://caml.inria.fr/`.
19. M. Takahashi. Parallel Reductions in lambda-Calculus. *Journal of Symbolic Computation*, 7(2), 1989.
20. V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: the higher-order case. In *3rd LFCS*, *LNCS* 813. 1994.
21. F. van Raamsdonk. On termination of higher-order rewriting. In *12th RTA*, *LNCS* 2051. 2001.
22. D. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1993.