

# Confluence of Pattern-Based Calculi

Horatiu Cirstea and Germain Faure

Université Nancy 2 & Université Henri Poincaré & LORIA  
BP 239, F-54506 Vandoeuvre-lès-Nancy France  
`first.last@loria.fr`

**Abstract** Different pattern calculi integrate the functional mechanisms from the  $\lambda$ -calculus and the matching capabilities from rewriting. Several approaches are used to obtain the confluence but in practice the proof methods share the same structure and each variation on the way pattern-abstractions are applied needs another proof of confluence.

We propose here a generic confluence proof where the way pattern-abstractions are applied is axiomatized. Intuitively, the conditions guarantee that the matching is stable by substitution and by reduction.

We show that our approach directly applies to different pattern calculi, namely the lambda calculus with patterns, the pure pattern calculus and the rewriting calculus. We also characterize a class of matching algorithms and consequently of pattern-calculi that are not confluent.

## Introduction

Pattern matching, *i.e.* the ability to discriminate patterns is one of the main basic mechanisms human reasoning is based on. This concept is present since the beginning of information processing modeling. Instances of it can be traced back to pattern recognition and it has been extensively studied when dealing with strings [16], trees [10] or feature objects [1].

It is somewhat astonishing that one of the most commonly used models of computation, the lambda calculus, uses only trivial pattern matching. This has been extended, initially for programming concerns, either by the introduction of patterns in lambda calculi [21], or by the introduction of matching and rewrite rules in functional programming languages. There are several formalisms that address the integration of pattern matching capabilities with the lambda calculus; we can mention the  $\lambda$ -calculus with patterns [25], the rewriting calculus [6], the pure pattern calculus [11] and the  $\lambda$ -calculus with constructors [2].

Each of these pattern-based calculi differs on the way patterns are defined and on the way pattern-abstractions are applied. Thus, patterns can be simple variables like in the  $\lambda$ -calculus, algebraic terms like in the algebraic  $\rho$ -calculus [7], special (static) patterns that satisfy certain (semantic or syntactic) conditions like in the  $\lambda$ -calculus with patterns or dynamic patterns that can be instantiated and possibly reduced like in the pure pattern calculus and some versions of the  $\rho$ -calculus. The underlying matching theory strongly depends on the form of the patterns and can be syntactic, equational or more sophisticated [11,4].

Although some of these calculi just extend the  $\lambda$ -calculus by allowing pattern-abstractions instead of variable-abstractions the confluence of these formalisms is lost when no restrictions are imposed.

Several approaches are then used to recover confluence. One of these techniques consists in syntactically restricting the set of patterns and then showing that the reduction relation is confluent for the chosen subset. This is done for example in the  $\lambda$ -calculus with patterns and in the  $\rho$ -calculus (with algebraic patterns). The second technique considers a restriction of the initial reduction relation (that is, a strategy) to guarantee that the calculus is confluent on the whole set of terms. This is done for example in the pure pattern calculus where the matching algorithm is a partial function (whereas any term is a pattern).

Nevertheless we can notice that in practice, the proof methods share the same structure and that each variation on the way pattern-abstractions are applied needs another proof of confluence. There is thus a need for a more abstract and more modular approach in the same spirit as in [19,12]. A possible way to have a unified approach for proving the confluence is the application of the general and powerful results on the confluence of higher-order rewrite systems [15,18,23]. Although these results have already been applied for some particular pattern-calculi [5] the encoding seems to be rather complex for some calculi and in particular for the general setting proposed in this paper. Moreover, it would be interesting to have a framework where the expressiveness and (confluence) properties of the different pattern calculi can be compared.

In this paper, we show that all the pattern-based calculi using a unitary matching algorithm can be expressed in a general calculus parameterized by a function that defines the underlying matching algorithm and thus the way pattern-abstractions are applied. This function can be instantiated (implemented) by a unitary matching algorithm as in [6,11] but also by an anti-pattern matching algorithm [13] or it can be even more general [17]. We propose a generic confluence proof where the way pattern-abstractions are applied is axiomatized. Intuitively, the sufficient conditions to ensure confluence guarantee that the (matching) function is stable by substitution and by reduction.

We apply our approach to several classical pattern calculi, namely the  $\lambda$ -calculus with patterns, the pure pattern calculus and the  $\rho$ -calculus. For all these calculi, we give the encodings in the general framework and we obtain proofs of confluence. This approach does not provide confluence proofs for free but it establishes a proof methodology and isolates the key points that make the calculi confluent. It can also point out some matching algorithms that although natural at the first sight can lead to non-confluent reductions in the corresponding calculus.

*Outline of the paper* In Section 1 we give the syntax and semantics of the dynamic pattern  $\lambda$ -calculus. The hypotheses under which the calculus is confluent and the main theorems are stated in Section 2. A non-confluent calculus is given at the end of this section. In Section 3 we give the encoding of different pattern-calculi and the corresponding confluence proofs. Section 4 concludes and gives some perspectives to this work.

|            |                          |               |
|------------|--------------------------|---------------|
| $A, B ::=$ | $x$                      | (Variable)    |
|            | $  c$                    | (Constant)    |
|            | $  \lambda_{\theta} A.B$ | (Abstraction) |
|            | $  A B$                  | (Application) |

**Figure1.** Syntax of the (core) dynamic pattern  $\lambda$ -calculus

## 1 The Dynamic Pattern $\lambda$ -calculus

In this section, we first define the syntax and the operational semantics of the core dynamic pattern  $\lambda$ -calculus. We then give the general definition of the dynamic pattern  $\lambda$ -calculus.

### 1.1 Syntax

The syntax of the core dynamic pattern  $\lambda$ -calculus is defined in Figure 1. It consists of variables (denoted by  $x, y, z, \dots$ ), constants (denoted  $a, b, c, d, e, f, \dots$ ), abstractions and applications. In an abstraction of the form  $\lambda_{\theta} A.B$  we call the term  $A$  the pattern and the term  $B$  the body. The set  $\theta$  is a subset of the set of variables of  $A$  and represents the set of variables bound by the abstraction. This set is often omitted when it is exactly the set of free variables of the pattern. We sometimes use an algebraic notation  $f(A_1, \dots, A_n)$  for the term  $((f A_1) \dots) A_n$ .

Comparing to the  $\lambda$ -calculus we abstract not only on variables but on general terms and the set of variables bound by an abstraction is not necessarily the same as the set of (free) variables of the corresponding pattern. We say thus that the patterns are dynamic since they can be instantiated and possibly reduced.

**Definition 1 (Free and bound variables).** *The set of free and bound variables of a term  $A$ , denoted  $\mathbf{fv}(A)$  and  $\mathbf{bv}(A)$ , are defined inductively by:*

$$\begin{array}{llll}
 \mathbf{fv}(c) \triangleq \emptyset & \mathbf{bv}(c) \triangleq \emptyset & \mathbf{fv}(x) \triangleq \{x\} & \mathbf{bv}(x) \triangleq \emptyset \\
 \mathbf{fv}(A B) \triangleq \mathbf{fv}(A) \cup \mathbf{fv}(B) & & \mathbf{fv}(\lambda_{\theta} A.B) \triangleq (\mathbf{fv}(A) \cup \mathbf{fv}(B)) \setminus \theta & \\
 \mathbf{bv}(A B) \triangleq \mathbf{bv}(A) \cup \mathbf{bv}(B) & & \mathbf{bv}(\lambda_{\theta} A.B) \triangleq \mathbf{bv}(A) \cup \mathbf{bv}(B) \cup \theta & 
 \end{array}$$

When the set of free variables of a term is empty we say that the term is closed. Otherwise, it is open.

In what follows we work modulo  $\alpha$ -conversion, that is two terms that are  $\alpha$ -convertible are not distinguishable. Equality modulo  $\alpha$ -equivalence is denoted here by  $\equiv$ . We adopt Barendregt's *hygiene-convention* [3], *i.e.* free and bound variables have different names.

A *substitution* is a partial function from variables to terms (we use post-fix notation for substitution application). We denote by  $\sigma = \{x_1 \leftarrow A_1, \dots, x_n \leftarrow A_n\}$  the substitution that maps each variable  $x_i$  to a term  $A_i$ . The set  $\{x_1, \dots, x_n\}$  is called the domain of  $\sigma$  and is denoted by  $\mathcal{D}om(\sigma)$ . The range of a substitution  $\sigma$ , denoted by  $\mathcal{R}an(\sigma)$ , is the union of the sets  $\mathbf{fv}(x\sigma)$  where  $x \in \mathcal{D}om(\sigma)$ . The composition of two substitutions  $\sigma$  and  $\tau$  is denoted  $\sigma \circ \tau$  and defined as usually,

that is  $x(\sigma \circ \tau) = (x\tau)\sigma$ . We denote by  $\text{id}$  the empty substitution. The restriction of (the domain of) a substitution  $\sigma$  to a set of variables  $\theta$  is denoted by  $\sigma|_{\theta}$ .

**Definition 2 (Substitution).** *The application of a substitution  $\sigma$  to a term  $A$  is inductively defined by*

$$\begin{aligned} x\sigma &\triangleq A && \text{if } \sigma = \{\dots, x \leftarrow A, \dots\} \\ y\sigma &\triangleq y && \text{if } y \notin \text{Dom}(\sigma) \\ (\lambda_{\theta} A_1. A_2)\sigma &\triangleq \lambda_{\theta}(A_1\sigma).(A_2\sigma) \\ (A_1 A_2)\sigma &\triangleq (A_1\sigma)(A_2\sigma) \end{aligned}$$

In the abstraction case, we take the usual precautions to avoid variable captures.

## 1.2 Operational Semantics

The operational semantics of the core dynamic pattern  $\lambda$ -calculus is given by a single reduction rule that defines the way pattern-abstractions are applied. This rule, given in Figure 2, is parameterized by a partial function, denoted  $\text{Sol}(A \leftarrow_{\theta} B)$ , that takes as parameters two terms  $A$  and  $B$  and a set  $\theta$  of variables and returns a substitution.

In most of the cases this function corresponds to a pattern-matching algorithm but it can be even more general [17,13].

*Example 1 (Syntactic matching).* We consider matching problems of the form  $A \ll^{?} B$  and conjunctions of such problems built with the (associative, idempotent and with neutral element) operator  $\wedge$ . The symbol  $\mathbb{F}$  represents a matching problem without solution. The following set of terminating and confluent rules can be used to solve a (non-linear) syntactic matching problem:

$$\begin{aligned} A \ll^{?} A \wedge M &\rightarrow M \\ f(A_1, \dots, A_n) \ll^{?} f(A'_1, \dots, A'_n) \wedge M &\rightarrow \bigwedge_{i=1}^n (A_i \ll^{?} A'_i) \wedge M \\ f(A_1, \dots, A_n) \ll^{?} g(A'_1, \dots, A'_m) \wedge M &\rightarrow \mathbb{F} && f \neq g \\ f(A_1, \dots, A_n) \ll^{?} x \wedge M &\rightarrow \mathbb{F} \\ (x \ll^{?} A) \wedge (x \ll^{?} A') \wedge M &\rightarrow \mathbb{F} && A \neq A' \end{aligned}$$

We can define  $\text{Sol}(A \leftarrow_{\theta} B)$  as the function that normalizes the matching problem  $A \ll^{?} B$  w.r.t. the above rewrite rules and according to the obtained result returns:

- nothing (*i.e.* is not defined) if  $\text{fv}(A) \neq \theta$  or if the result is  $\mathbb{F}$ ,
- the substitution  $\{x_i \leftarrow A_i\}_{i \in I}$  if the result is of the form  $\bigwedge_{i \in I \neq \emptyset} x_i \ll^{?} A_i$ ,
- $\text{id}$ , if the result is empty.

In Section 2.2 we analyze the confluence of the relation induced by this rule and more precisely of its compatible [3] and transitive closure.

**Definition 3 (Compatible relation).** *A relation  $\mapsto_{\mathcal{R}}$  on the set of terms of the core dynamic pattern  $\lambda$ -calculus is said to be compatible if for all terms  $A$ ,  $B$  and  $C$  s.t.  $A \mapsto_{\mathcal{R}} B$  we have  $AC \mapsto_{\mathcal{R}} BC$ ,  $\lambda_{\theta} A.C \mapsto_{\mathcal{R}} \lambda_{\theta} B.C$ ,  $CA \mapsto_{\mathcal{R}} CB$  and  $\lambda_{\theta} C.A \mapsto_{\mathcal{R}} \lambda_{\theta} C.B$ .*

|             |   |   |
|-------------|---|---|
| ( $\beta$ ) | $(\lambda_{\theta} A.B)C \quad \rightarrow$ | $B\sigma$<br>where $\sigma = \mathcal{Sol}(A \ll_{\theta} C)$ |
|-------------|---|---|

**Figure 2.** Operational semantics of the core dynamic pattern  $\lambda$ -calculus

Different instances of the core dynamic pattern  $\lambda$ -calculus are obtained when we give concrete definitions to  $\mathcal{Sol}$ . For example, the  $\lambda$ -calculus can be seen as the core dynamic pattern  $\lambda$ -calculus such that

$$\sigma = \mathcal{Sol}(A \ll_{\theta} C) \text{ iff } A \text{ is a variable } x, \theta = \{x\} \text{ and } A\sigma \equiv C$$

*Example 2 (Case branchings).* Consider a pattern-based calculus with a case construct denoted using  $|$  (a.k.a. a `match` operator as in functional programming languages). It can be encoded in the core dynamic pattern  $\lambda$ -calculus as follows:

$$(A_1 \mapsto B_1 | \dots | A_n \mapsto B_n) C \quad \triangleq \quad (\lambda_x (A_1 \hookrightarrow B_1 / \dots / A_n \hookrightarrow B_n).x) C$$

where  $x$  is a fresh variable, the symbols  $\hookrightarrow$  and  $/$  are constants of the core dynamic pattern  $\lambda$ -calculus (infix notation) and the function  $\mathcal{Sol}$  may be defined by

$$\mathcal{Sol}((A_1 \hookrightarrow B_1 / \dots / A_n \hookrightarrow B_n) \ll_x A_i \sigma) \quad \triangleq \quad \{x \leftarrow B_i \sigma\}$$

Some pattern-calculi come with additional features and cannot be expressed as instances of the core dynamic pattern  $\lambda$ -calculus. For example, the pure pattern calculus [11] and some versions of the rewriting calculus [7] reduce the application of a pattern-abstraction to a special term when the corresponding matching problem has not and will never have a solution. In the rewriting calculus [7] there is also a construction that aggregates terms and then distributes them over applications. These calculi as well as their encodings are briefly presented in Section 3.

We define thus the dynamic pattern  $\lambda$ -calculus as the core dynamic pattern  $\lambda$ -calculus extended by a set of rewrite rules. As for  $\mathcal{Sol}$  this set, denoted  $\xi$ , is not made precise and can be considered as a parameter of the calculus. It can include for example some rules to reduce particular pattern-abstractions to a special constant representing a definitive matching failure or some extra rules describing the distributivity of certain symbols (like the structure operator of the  $\rho$ -calculus) over the applications.

In what follows,  $\mapsto_{\beta}$  denotes the compatible closure of the relation induced by the rule  $\beta$  and  $\mapsto_{\beta}^*$  denotes the transitive closure of  $\mapsto_{\beta}$ . Similarly, we will denote by  $\mapsto_{\beta \cup \xi}$  the compatible closure of the relation induced by the rules  $\beta$  and  $\xi$ .  $\mapsto_{\beta \cup \xi}^*$  denotes the transitive closure of  $\mapsto_{\beta \cup \xi}$ .

## 2 The Confluence of the Dynamic Pattern $\lambda$ -calculus

The calculus is not confluent when no restrictions are imposed on the function  $\mathcal{Sol}$ . This is for example the case when we consider the decomposition of ap-

plications containing free active variables. (the term  $(\lambda_x x a.x) ((\lambda_y y.y) a)$  can be reduced either to  $\lambda_y y.y$  or to  $(\lambda_x x a.x) a$  which are not joinable) or when we deal with non-linear patterns (see Section 2.3). Nevertheless, the confluence is recovered for some specific definitions of  $\mathcal{Sol}$  like the one used in Section 1.2 when defining the  $\lambda$ -calculus as a core dynamic pattern  $\lambda$ -calculus.

In this section we give some sufficient conditions that guarantee the confluence of the core dynamic pattern  $\lambda$ -calculus. Intuitively, the hypotheses introduced in Section 2.1 under which we prove the confluence of the calculus guarantee the coherence between  $\mathcal{Sol}$  and the underlying relation of the calculus. The obtained results can then be generalized for a dynamic pattern  $\lambda$ -calculus with an extended set of rules  $\xi$  that satisfies some classical coherence conditions.

We use here a proof method introduced by Martin-Löf that consists in defining a so-called parallel reduction that, intuitively, can reduce all the redexes initially present in the term and that is strongly confluent (even if the one-step reduction is not) under some hypotheses.

**Definition 4 (Parallel reduction).** *The parallel reduction is inductively defined on the set of terms as follows:*

$$\frac{}{A \dashrightarrow A} \quad \frac{A \dashrightarrow A' \quad B \dashrightarrow B'}{AB \dashrightarrow A'B'} \quad \frac{A \dashrightarrow A' \quad B \dashrightarrow B'}{\lambda_\theta A.B \dashrightarrow \lambda_\theta A'.B'}$$

$$\frac{A \dashrightarrow A' \quad B \dashrightarrow B' \quad C \dashrightarrow C'}{(\lambda_\theta A.B)C \dashrightarrow B'\sigma'} \text{ IF } \sigma' \in \mathcal{Sol}(A' \ll_\theta C')$$

Note that the parallel reduction is compatible. Moreover, we should note that this definition of parallel reduction does not coincide with the classical notion of developments. For example, if we use a  $\mathcal{Sol}$  function that computes the substitution solving the matching between its two arguments (for example, like in Example 1 but without using the last rule) then we have

$$\frac{f x \dashrightarrow f x \quad x \dashrightarrow x \quad (\lambda y.f y) a \dashrightarrow f a}{(\lambda(f x).x)((\lambda y.f y) a) \dashrightarrow a}$$

The substitution  $\{x \leftarrow a\}$  solves the syntactic matching between the terms  $f x$  and  $f a$  and thus, even if the initial term contains no head redex it can still be reduced using the parallel reduction.

We extend the definition of parallel reduction to substitutions having the same domain by setting  $\sigma \dashrightarrow \sigma'$  if for all  $x$  in the domain of  $\sigma$ , we have  $x\sigma \dashrightarrow x\sigma'$ .

## 2.1 Stability of $\mathcal{Sol}$

*Preservation of free variables* First of all, when defining a higher-order calculus it is natural to ask that the set of free variables is preserved by reduction (some free variables can be lost but no free variables can appear during reduction). For example, the free variables of the term  $(\lambda_\theta A.B)C$  should include the ones of the term  $B\sigma$  with  $\sigma = \mathcal{Sol}(A \ll_\theta C)$ . Thus, the substitution  $\sigma$  should instantiate

$$\begin{array}{l}
\forall A, C, A', C' \\
\mathbf{H}_0 : \quad \text{Sol}(A \ll_{\theta} C) = \sigma \quad \Longrightarrow \quad \begin{cases} \text{Dom}(\sigma) = \theta \\ \text{Ran}(\sigma) \subseteq \text{fv}(C) \cup (\text{fv}(A) \setminus \theta) \end{cases} \\
\mathbf{H}_1 : \quad \text{Sol}(A \ll_{\theta} C) = \sigma \quad \Longrightarrow \quad \begin{cases} \forall \tau, \text{Var}(\tau) \cap \theta = \emptyset, \\ \text{Sol}(A\tau \ll_{\theta} C\tau) = (\tau \circ \sigma)_{|\theta} \end{cases} \\
\mathbf{H}_2 : \quad \begin{cases} \text{Sol}(A \ll_{\theta} C) = \sigma \\ A \mapsto A' \quad C \mapsto C' \end{cases} \quad \Longrightarrow \quad \begin{cases} \text{Sol}(A' \ll_{\theta} C') = \sigma' \\ \sigma \mapsto \sigma' \end{cases}
\end{array}$$

**Figure 3.** Conditions to ensure confluence of the core dynamic pattern  $\lambda$ -calculus

all the variables bound (by the abstraction) in  $B$ , that is, all the variables in  $\theta$ . Moreover, the free variables of  $\sigma$  should already be present in  $C$  or free in  $A$ . These conditions are enforced by the hypothesis  $\mathbf{H}_0$  in Figure 3.

If we think of  $\text{Sol}$  as a unitary matching algorithm, the examples that do not verify  $\mathbf{H}_0$  are often peculiar algorithms (for example the function that returns the substitution  $\{x \leftarrow y\}$  for any problem). When considering non-unitary matching (not handled in this paper), there are several examples that do not verify  $\mathbf{H}_0$ . For instance, the algorithms solving higher-order matching problems or matching problems in non-regular theories (*e.g.*, such that  $x \times 0 = 0$ ) do not verify  $\mathbf{H}_0$ .

*Stability by substitution* In the core dynamic pattern  $\lambda$ -calculus, when a pattern-abstraction is applied the argument may be open. One can wait for the argument to be instantiated and only then compute the corresponding substitution (if it exists) and reduce the application. On the other hand, one might not want to sequentialize the reduction but to perform the reduction as soon as possible. Nevertheless, the same result should be obtained for both reduction strategies. This is enforced by the hypothesis  $\mathbf{H}_1$  in Figure 3.

If we consider that  $\text{Sol}$  performs a naive matching algorithm that does not take into account the variables in  $\theta$  and such that  $\text{Sol}(a \ll_{\emptyset} b)$  has no solution and  $\text{Sol}(x \ll_{\emptyset} y) = \{x \leftarrow y\}$ , then the hypothesis  $\mathbf{H}_1$  is clearly not verified (take  $\tau = \{x \leftarrow a, y \leftarrow b\}$ ).

*Stability by reduction* When applying a pattern-abstraction, the argument may also be not fully reduced. Once again, if  $\text{Sol}$  succeeds and produces a substitution  $\sigma$  then subsequent reductions should not lead to a definitive failure for  $\text{Sol}$ . Moreover, the substitution that is eventually obtained should be derivable from  $\sigma$ . This is formally defined in hypothesis  $\mathbf{H}_2$ .

The function  $\text{Sol}$  proposed in Example 1 does not satisfy this hypothesis. If we take  $I \triangleq (\lambda y.y)$  then  $\text{Sol}(f(x,x) \ll_x f(II,II)) = \{x \leftarrow II\}$  but  $\text{Sol}(f(x,x) \ll_x f(II,I))$  has no solution. Similarly, this hypothesis is not satisfied by a matching algorithm that allows the decomposition of applications containing a so-called free *active* variable (*i.e.* a variable in applicative position). For example, we can have  $\text{Sol}(xa \ll_x (\lambda y.y)a) = \{x \leftarrow \lambda y.y\}$  but  $\text{Sol}(xa \ll_x a)$  has no solution for any classical (first-order) matching algorithm.

## 2.2 Sufficient Conditions for the Confluence

In this section, we first prove the confluence of the core dynamic pattern  $\lambda$ -calculus under the hypotheses  $\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2$ . The proof uses the standard techniques of parallel reduction first introduced by Tait and Martin-Löf. We first show that the reflexive and transitive closure of the parallel and one-step reductions are the same. Then we show that the parallel reduction has the diamond property and we deduce the confluence of the one-step reduction.

The three hypotheses given in the previous section are used for showing the strong confluence of the parallel reduction and in particular the Lemma 2. On the other hand, we can show that the reflexive and transitive closure of  $\dashv\vdash$  is equal to  $\mapsto_{\beta}$  independently of the properties of  $Sol$ .

**Lemma 1.** *The following inclusions hold.  $\mapsto_{\beta} \subseteq \dashv\vdash \subseteq \mapsto_{\beta}$ .*

The next fundamental lemma and the diamond property of the parallel reduction are obtained by induction and are used for the confluence theorem.

**Lemma 2 (Fundamental lemma).** *For all terms  $C$  and  $C'$  and for all substitutions  $\sigma$  and  $\sigma'$ , such that  $C \dashv\vdash C'$  and  $\sigma \dashv\vdash \sigma'$  we have  $C\sigma \dashv\vdash C'\sigma'$ .*

**Lemma 3 (Diamond property for  $\dashv\vdash$ ).** *The relation  $\dashv\vdash$  satisfies the diamond property that is, for all terms  $A, B$  and  $C$  if  $A \dashv\vdash B$  and  $A \dashv\vdash C$  then there exists a term  $D$  such that  $B \dashv\vdash D$  and  $C \dashv\vdash D$ .*

**Theorem 1 (Confluence).** *The core dynamic pattern  $\lambda$ -calculus with  $Sol$  satisfying  $\mathbf{H}_0, \mathbf{H}_1$  and  $\mathbf{H}_2$  is confluent.*

As we have already said most of the pattern-calculi extend the basic  $\beta$  rule (or its equivalent) by a set of rules. We will state here the conditions that should be imposed in order to prove the confluence of the dynamic pattern  $\lambda$ -calculus, conditions that turn out to be satisfied by most of the different calculi that can be expressed as instances of the dynamic pattern  $\lambda$ -calculus.

We show in this section that the confluence of extensions of the core dynamic pattern  $\lambda$ -calculus with an appropriate set of rules is easy to deduce using Yokouchi-Hikita's lemma [26] (see also [8]).

**Lemma 4 (Yokouchi-Hikita).** *Let  $\mathcal{R}$  and  $\mathcal{S}$  be two relations defined on the same set  $\mathcal{T}$  of terms such that*

- $\mathcal{R}$  is strongly normalizing and confluent,
- $\mathcal{S}$  has the diamond property,
- for all  $A, B$  and  $C$  in  $\mathcal{T}$  such that  $A \mapsto_{\mathcal{R}} B$  and  $A \mapsto_{\mathcal{S}} C$  then there exists  $D$  such that  $B \mapsto_{\mathcal{R}^* \mathcal{S} \mathcal{R}^*} D$  and  $C \mapsto_{\mathcal{R}^*} D$  (Yokouchi-Hikita's diagram).

*Then the relation  $\mathcal{R}^* \mathcal{S} \mathcal{R}^*$  is confluent.*

**Theorem 2.** *The dynamic pattern  $\lambda$ -calculus is confluent when*

- $Sol$  satisfies  $\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2$ ,
- the set  $\xi$  of reduction rules is strongly normalizing and confluent,
- the relations  $\dashv\vdash$  and  $\xi$  satisfy Yokouchi-Hikita's diagram.

*Proof.* Notice that  $\mapsto_{\beta \cup \xi}$  and  $\mapsto_{\xi} \dashv\vdash \mapsto_{\xi}$  are equal (consequence of Lemma 1) and apply Yokouchi-Hikita's lemma with  $\dashv\vdash$  and the relation induced by  $\xi$ .  $\square$

### 2.3 Encoding Klop's Counter-example Using Linear Patterns

The different results we give below may seem to be limited because the conditions imposed on the matching algorithm are strong. Nevertheless, these conditions are respected by most of the pattern-calculi we have explored and relaxing them leads to classical counter-examples for the confluence.

For example, if the matching can be performed on active variables then non-confluent reductions can be obtained in both the lambda-calculus with patterns [25] and in the rewriting calculus [7]. Similarly, non-linear patterns lead to non-confluent reductions that are variations of the Klop's counter-example [14] for higher-order systems dealing with non-linear matching. This is why in the  $\lambda$ -calculus with patterns or in the  $\rho$ -calculus we only consider *linear* patterns and in the pure pattern calculus matching against non-linear terms always fails.

When using dynamic patterns containing variables that are not bound in the abstraction the confluence hypotheses should be carefully verified. More precisely, the behavior of non-linear rules can be encoded using *linear* and dynamic patterns. Consequently, Klop's counter-example can be encoded in the corresponding calculus that is therefore non-confluent.

**Proposition 1 (Non-confluence).** *The core dynamic pattern  $\lambda$ -calculus with  $Sol$  such that for all terms  $A$  and  $B$  and for some constant  $d$*

$$\begin{aligned} Sol(A \leftarrow_{\emptyset} A) &= \text{id} \\ Sol(x \leftarrow_x A) &= \{x \leftarrow A\} \\ Sol(d(x, y) \leftarrow_{x,y} d(A, B)) &= \{x \leftarrow A, y \leftarrow B\} \end{aligned} \quad \textit{is not confluent.}$$

*Proof.* It is sufficient to remark that we can encode non-linear patterns using dynamic linear patterns as follows:

$$\lambda_x(d x x).\clubsuit \quad \triangleq \quad \lambda_{x,y}(d x y).(\lambda_{\emptyset} x.\clubsuit)y$$

where  $\clubsuit$  denotes an arbitrary term. Then we adapt the encoding of Klop's counter-example in the core dynamic pattern  $\lambda$ -calculus.  $\square$

As a consequence we obtain that any pattern-calculus defined using a function  $Sol$  that satisfies the conditions in Proposition 1 is not confluent. This is somewhat surprising since the last two computations are clearly satisfied by any classical syntactic matching algorithm and the first one seems to be a reasonable choice. On the other hand, ignoring the information given by the set  $\theta$  of bound variables when performing matching can lead to strange behaviors and in particular it allows for an encoding of the equality of terms.

The conditions of Proposition 1 are not satisfied by the specific calculi considered in this paper. More precisely,  $Sol(A \leftarrow_{\emptyset} A)$  does not return the identity for all terms but only for a subset that is defined either statically (syntactical restrictions on patterns) like in the  $\lambda$ -calculus with patterns or the rewriting calculus or dynamically (only a subset of terms can be matched) like in the pure pattern calculus. In all these cases the matchable terms from the corresponding subsets cannot be used to encode Klop's counter-example.

|  |
|--|
| $(\beta_{\lambda_P}) \quad (\lambda A.B)(A\sigma) \quad \rightarrow \quad B\sigma$ |
|--|

**Figure 4.** Operational semantics of the  $\lambda$ -calculus with patterns

### 3 Instantiations of the Dynamic Pattern $\lambda$ -calculus

In this section, we give some instantiations of the dynamic pattern  $\lambda$ -calculus. For the sake of simplicity, we only give the key points for each of the pattern based calculi. All these calculi have been proved confluent under appropriate conditions; we give here confluence proofs based on these conditions and using the general confluence proof for the dynamic pattern  $\lambda$ -calculus. This latter approach does not provide confluence proofs for free but gives a proof methodology that focuses on the fundamental properties of the underlying matching that can be thus seen as the key issue of pattern based calculi.

#### 3.1 $\lambda$ -calculus with Patterns

The  $\lambda$ -calculus with patterns was introduced in [25] as an extension of the  $\lambda$ -calculus with pattern-matching facilities. The set of terms is parameterized by a set of patterns  $\Phi$  on which we can abstract.

The syntax of the  $\lambda$ -calculus with patterns is thus the one of the core dynamic pattern  $\lambda$ -calculus but where patterns are taken in a given set and abstractions always bind all the (free) pattern variables. Its operational semantics is given by the rule in Figure 4.

Instead of considering syntactical restrictions, we can equivalently consider that a matching problem  $A \rightsquigarrow_{\theta} A\sigma$  has a solution only when  $A \in \Phi$ . The  $\lambda$ -calculus with patterns can thus be seen as an instance of the core dynamic pattern  $\lambda$ -calculus.

The calculus is not confluent (see [25] Ex. 4.18) in general but some restrictions can be imposed on the set of patterns to recover confluence. This restriction is called the rigid pattern condition (RPC) and can be defined using the parallel reduction of the calculus. The definition allows for patterns which are extensionally but not intensionally rigid, such as  $\Omega xy$  with  $\Omega = (\lambda x.xx)(\lambda x.xx)$ . We choose a (less general) syntactical characterization [25] that excludes these pathological cases.

**Definition 5 (RPC).** *The set of terms satisfying RPC is the set of all terms of the  $\lambda$ -calculus which*

- *are linear (each free variable occur at most once),*
- *are in normal form,*
- *have no active variables (i.e., no sub-terms of the form  $xA$  where  $x$  is free).*

Note that reformulating **H<sub>2</sub>** in the particular case of a *Sol* function that performs matching on closed patterns leads to a condition close to the original RPC (the parallel reduction used in the definition of RPC is slightly different).

Nevertheless, the hypothesis  $\mathbf{H}_2$  allows the matching to be performed on patterns that are not in normal form (or not reducible to themselves) while this is not the case for the RPC.

*Example 3.* Pairs and projections can be encoded in the  $\lambda$ -calculus with patterns by directly matching on the pair encoding.

$$\begin{aligned} (\lambda(\lambda z.(z x) y).x)(\lambda z.(z A) B) &\mapsto_{\beta_{\lambda P}} x\{x \leftarrow A, y \leftarrow B\} \\ &\equiv A \end{aligned}$$

**Proposition 2.** *The  $\lambda$ -calculus with patterns is confluent if the patterns are taken in the set defined by the RPC.*

*Proof.* The hypotheses  $\mathbf{H}_0$  and  $\mathbf{H}_1$  follow immediately. To prove  $\mathbf{H}_2$ , we can remark that if  $P\sigma \dashv\vdash B$  with  $P \in \text{RPC}$  then  $\exists B', \sigma'$  s.t.  $B' \equiv P\sigma'$  with  $\sigma \dashv\vdash \sigma'$ . This proves that a redex cannot overlap with  $P$  in  $P\sigma$  if  $P \in \text{RPC}$  and thus that the condition  $\mathbf{H}_2$  is satisfied. We conclude the proof by applying Thm. 1.  $\square$

### 3.2 Rewriting Calculus

The rewriting calculus was introduced in [6] to make explicit all the ingredients of rewriting. There are several versions of the calculus but we focus here on the  $\rho_{\rightarrow}$ -calculus [7] that uses a left-distributive structure operator and a special constant representing matching failures.

Note that the syntax used here is slightly different from the original presentations that introduce matching constraints. Nevertheless, the use of matching constraints is crucial only when considering explicit matching and substitution application. We also omit the type related aspects of this calculi and consider a syntactic matching (in the original version, the reduction rules of  $\rho_{\rightarrow}$ -calculus are parameterized by a matching theory  $\mathbb{T}$  but the confluence is proved for the syntactic version). The syntax of the  $\rho_{\rightarrow}$ -calculus is given in Figure 5.

The *patterns* are linear algebraic terms (*i.e.* terms constructed only with variables, constants and application). A *structure* is a collection of terms that can be seen either as a set of rewrite rules or as a set of results. The symbol `stk` can be considered as the special constant representing a redex whose underlying matching problem is unsolvable.

We consider a superposition relation  $\not\sqsubseteq$  between (patterns and) terms whose aim is to characterize a broad class of matching equations that have not and will never have a solution (*i.e.* independently of subsequent instantiations and reductions). Thus, if  $P \not\sqsubseteq A$  then  $\forall \sigma_1, \sigma_2, \forall A', A\sigma_1 \mapsto A' \Rightarrow P\sigma_2 \not\equiv A'$ . This definition is clearly undecidable but syntactic characterizations can be given [7].

The  $\rho_{\rightarrow}$ -calculus handles uniformly matching failures and eliminates them when they are not significant for the computation. The semantics of the  $\rho_{\rightarrow}$ -calculus is given in Figure 5.

We can consider `\}` and `stk` as constants of the dynamic pattern  $\lambda$ -calculus and thus we can see the syntax of the  $\rho_{\rightarrow}$ -calculus as an instance of the syntax of the

| <u>Syntax of the <math>\rho_{\rightarrow}</math>-calculus</u>    |   |  |   |
|--|---|--|---|
| <b>Patterns</b>  | $P ::= x \mid \text{stk} \mid c(P, \dots, P)$                           |  | <i>(variables occur only once in any P)</i> |
| <b>Terms</b>   | $A ::= c \mid \text{stk} \mid x \mid \lambda P.A \mid A A \mid A \wr A$ |  |   |
| <u>Semantics of the <math>\rho_{\rightarrow}</math>-calculus</u> |   |  |   |
| $(\beta_{\rho_{\text{stk}}})$                                    | $(\lambda P.A) B \rightarrow A\sigma$                                   |  | with $P\sigma = B$                          |
| $(\delta)$   | $(A \wr B) C \rightarrow A C \wr B C$                                   |  |   |
| $(\text{stk})$   | $(\lambda P.A) B \rightarrow \text{stk}$                                |  | if $P \not\sqsubseteq B$                    |
| $(\text{stk})$   | $\text{stk} \wr A \rightarrow A$  |  |   |
| $(\text{stk})$   | $A \wr \text{stk} \rightarrow A$  |  |   |
| $(\text{stk})$   | $\text{stk} A \rightarrow \text{stk}$                                   |  |   |

**Figure5.** The  $\rho_{\rightarrow}$ -calculus

dynamic pattern  $\lambda$ -calculus. The rule  $(\beta_{\rho_{\text{stk}}})$  can be considered as an instance of the  $(\beta)$  rule of the dynamic pattern  $\lambda$ -calculus.

$$(\beta_{\rho}) \quad (\lambda P.A)B \rightarrow A\sigma \quad \text{if } \sigma = \text{Sol}(P \ll B)$$

where  $\text{Sol}(P \ll B)$  has a solution only when  $P$  is a pattern. Since patterns are linear algebraic terms then the  $\text{Sol}$  function can be implemented using first-order linear matching *à la* Huet (see Example 1).

**Proposition 3.** *The  $\rho_{\rightarrow}$ -calculus with linear algebraic patterns is confluent.*

*Proof.* The patterns of the  $\rho_{\rightarrow}$ -calculus satisfy the RPC. The confluence of the  $(\beta_{\rho_{\text{stk}}})$  rule is thus obtained as in Prop. 2. The relation  $\delta \cup \text{stk}$  induces a terminating relation. It is also locally confluent (all critical pairs are joinable). Moreover, the relations  $\dashrightarrow_{\beta_{\rho_{\text{stk}}}}$  and  $(\delta \cup \text{stk})$  satisfy the Yokouchi-Hikita's diagram (easy induction of the structure of terms). Thm. 2 concludes the proof.  $\square$

### 3.3 Pure Pattern Calculus

In the  $\lambda$ -calculus, data structures such as pairs of lists can be encoded. Although the  $\lambda$ -calculus supports some functions that act uniformly on data structures, it cannot support operations that exploit characteristics common to all data structures such as an update function that traverses any data structures to update its atoms. In the pure pattern calculus [11] where any term can be a pattern the focus is put on the dynamic matching of data structures.

The syntax of the pure pattern calculus is the same as the one of the dynamic pattern  $\lambda$ -calculus (except that the pure pattern calculus defines a single constructor).

Pattern-abstractions are applied using a particular matching algorithm. Although the original paper uses a single rule to describe application of pattern-abstractions we present it here using two rules. First, a rule that is an instance

| <u><math>\phi</math>-data structures and <math>\phi</math>-matchable forms</u> |                           |               |  |
|--|---------------------------|---------------|--|
| $D ::= x (x \in \phi) \mid c \mid D A$   |                           |               |  |
| $E ::= D \mid \lambda_{\theta} A.B$  |                           |               |  |
| where $A$ and $B$ are arbitrary terms  |                           |               |  |
| <u>Semantics of the pure pattern calculus</u>                                  |                           |               |  |
| $(\beta_{pc})$   | $(\lambda_{\theta} A.B)C$ | $\rightarrow$ | $B\sigma$<br>if $\sigma = Sol(A \ll_{\theta} C)$   |
| $(\beta_{pc}^{stk})$   | $(\lambda_{\theta} A.B)C$ | $\rightarrow$ | $\lambda x.x$<br>if none = $Sol(A \ll_{\theta} C)$ |

**Figure 6.** The pure pattern calculus

of the  $(\beta)$  rule of the dynamic pattern  $\lambda$ -calculus. Secondly, a rule that reduces the corresponding pattern-abstraction application to the identity (the motivation for this second rule is given in [11]) when the pattern-matching does not succeed.

The matching algorithm of the pure pattern calculus is based on the notions of  $\phi$ -data structures (denoted  $D$ ) and  $\phi$ -matchable forms (denoted  $E$ ) that are given in Figure 6.

The operational semantics of the pure pattern calculus is given in Figure 6 where the partial function  $Sol$  is defined by the following equations that are applied respecting the order below

$$\begin{aligned}
Sol(x \ll_{\theta} A) &= \{x \leftarrow A\} \quad \text{if } x \in \theta \\
Sol(c \ll_{\theta} c) &= \text{id} \\
Sol(A_1 A_2 \ll_{\theta} B_1 B_2) &= Sol(A_1 \ll_{\theta} B_1) \uplus Sol(A_2 \ll_{\theta} B_2) \\
&\quad \text{if } A_1 A_2 \text{ is a } \theta\text{-data structure} \\
&\quad \text{if } B_1 B_2 \text{ is a data structure} \\
Sol(A_1 \ll_{\theta} B_1) &= \text{none} \\
&\quad \text{if } A_1 \text{ is a } \theta\text{-matchable form} \\
&\quad \text{if } B_1 \text{ is a matchable form}
\end{aligned}$$

Note that the union  $\uplus$  is only defined for substitutions of disjoint domains and that the union of none and  $\sigma$  is always none.

At first sight, the matching algorithm may seem surprising because one decomposes application syntactically whereas it is a higher-order symbol. This is sound because the decomposition is done only on data-structures, which consist of head normal forms.

*Example 4.* [11] Define  $\mathbf{elim} \triangleq \lambda x. \lambda y. (xy).y$  to be the generic eliminator. For example, suppose given two constants  $\mathbf{Cons}$  and  $\mathbf{Nil}$  representing list constructs. We define the function  $\mathbf{singleton} \triangleq \lambda x. (\mathbf{Cons} \ x \ \mathbf{Nil})$  and we check that

$$\begin{aligned}
\mathbf{elim} \ \mathbf{singleton} &\equiv (\lambda x. \lambda y. (xy).y) (\lambda x. (\mathbf{Cons} \ x \ \mathbf{Nil})) \\
&\mapsto_{\rho_{ppc}} \lambda (\mathbf{Cons} \ y \ \mathbf{Nil}).y
\end{aligned}$$

**Proposition 4.** *The pure pattern calculus is confluent.*

*Proof.* The hypothesis  $\mathbf{H}_0$  is true. The hypotheses  $\mathbf{H}_1$  and  $\mathbf{H}_2$  are not surprisingly intermediate results in [11]. In particular, Lemma 7 [11] states that the function  $Sol$  is stable by substitution and Lemma 8 [11] proves that the function  $Sol$  is stable by reduction (when it returns a substitution and when it returns none). We use this property to prove that the relation  $(\beta_{pc}^{stk})$  is locally confluent (simple induction). It is trivially terminating, and thus confluent. The fact that the relations  $\dashv\vdash_{\beta_{pc}}$  and  $(\beta_{pc}^{stk})$  verify Yokouchi-Hikita’s diagram is obtained again by a simple induction. The only interesting case is for the term  $(\lambda A_0.A_1)A_2$  but it is easy to conclude using the stability by reduction of the function  $Sol$ .  $\square$

## 4 Conclusions and Future Work

We propose here a different formulation for different pattern-based calculi and we use the confluence properties of the general formalism to give alternative confluence proofs for these calculi. The general confluence proof uses the standard techniques of parallel reduction of Tait and Martin-Löf and Yokouchi-Hikita’s lemma. The method proposed by M. Takahashi [22] gives in general more elegant and shorter proofs by using the notion of developments. Reformulating the hypotheses  $\mathbf{H}_0$ ,  $\mathbf{H}_1$  and  $\mathbf{H}_2$  and adapting the proofs of this paper is easy but given a pattern-calculus the reformulated hypotheses are often more difficult to prove than for the original case.

Moreover, we show that the proof of confluence of the  $\rho$ -calculus is easy to deduce from our general result as soon as the structure operator has no equational theory. Nevertheless, if one wants to switch to non-unitary matching (and this is very useful in practice [24,20]) then the  $\beta$  rule should return a collection of results and the structure operator should be (at least) associative and commutative. This extension is syntactically and semantically non-trivial and opens new challenging problems as mentioned in [9]. Nevertheless, we think that this work is a good starting point for the study of the confluence properties of pattern-calculi with non-unitary matching.

*Acknowledgments* We would like to thank C. Kirchner for useful interactions and comments on this work and D. Kesner for the many discussions we have been having on pattern-calculi.

## References

1. H. Ait-Kaci, A. Podelski, and G. Smolka. A feature constraint system for logic programming with entailment. *Theoretical Computer Science*, 122(1-2):263–283, 1994.
2. A. Arbiser, A. Miquel, and A. Ríos. A lambda-calculus with constructors. In *Term Rewriting and Applications – RTA’06*, volume 4098 of *Lecture Notes in Computer Science*, pages 181–196, Seattle, USA, August 2006. Springer.

3. H. Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.
4. G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure patterns type systems. In *Principles of Programming Languages - POPL'03*, volume 38. ACM, January 2003.
5. C. Bertolissi and C. Kirchner. The rewriting calculus as a combinatory reduction system. In *Foundations of Software Science and Computation Structures - FoSSaCS'07*, Lecture Notes in Computer Science, Braga, Portugal, March 2007. Springer.
6. H. Cirstea and C. Kirchner. The rewriting calculus — Part I and II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9:427–498, May 2001.
7. H. Cirstea, L. Liquori, and B. Wack. Rewriting calculus with fixpoints: Untyped and first-order systems. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs - TYPES'03*, volume 3085 of *Lecture Notes in Computer Science*, pages 147–161, Torino, Italy, May 2003. Springer.
8. P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM (JACM)*, 43(2):362–397, 1996.
9. G. Faure and A. Miquel. A categorical semantics of the parallel lambda-calculus. 2007. Submitted.
10. C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.
11. B. Jay and D. Kesner. Pure pattern calculus. In *European Symposium on Programming - ESOP'06*, volume 3924 of *Lecture Notes in Computer Science*, pages 100–114, Vienna, Austria, March 2006. Springer.
12. D. Kesner. Confluence of extensional and non-extensional lambda-calculi with explicit substitutions. *Theoretical Computer Science*, 238(1-2):183–220, 2000.
13. C. Kirchner, R. Kopetz, and P.-E. Moreau. Anti-pattern matching. In *European Symposium on Programming - ESOP'07*, Lecture Notes in Computer Science, Braga, Portugal, March 2007. Springer.
14. J. W. Klop. *Combinatory Reduction Systems*. Ph.D. thesis, Mathematisch Centrum, Amsterdam, 1980.
15. J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.
16. D. E. Knuth, J. Morris, and V. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.
17. L. Liquori, F. Honsell, and M. Lenisa. A framework for defining logical frameworks. In *Electronic Notes in Theoretical Computer Science.*, volume 172, 2007.
18. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.
19. P.-A. Melliès. Axiomatic rewriting theory VI residual theory revisited. In *Rewriting Techniques and Applications - RTA'02*, volume 2378 of *Lecture Notes in Computer Science*, pages 24–50, Copenhagen, Denmark, July 2002. Springer.

20. J. Meseguer. A logical theory of concurrent objects and its realization in the maude language.
21. S. Peyton-Jones. *The implementation of functional programming languages*. Prentice Hall, Inc., 1987.
22. M. Takahashi. Parallel reductions in  $\lambda$ -calculus. *Information and Compututation*, 118:120–127, 1995.
23. Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. M. Bezem, J. W. Klop and R. de Vrijer, eds.
24. The Tom Team. The Tom language. <http://tom.loria.fr/>.
25. V. van Oostrom. Lambda calculus with patterns. Technical report, Vrije Universiteit, Amsterdam, November 1990.
26. H. Yokouchi and T. Hikita. A rewriting system for categorical combinators with multiple arguments. *SIAM Journal on Computing*, 19(1):78–97, February 1990.