On the Implementation of Dynamic Patterns

Thibaut Balabonski
Laboratoire PPS, CNRS and Université Paris Diderot
thibaut.balabonski@pps.jussieu.fr

The evaluation mechanism of pattern matching with dynamic patterns is modelled in the *Pure Pattern Calculus* by one single meta-rule. This contribution presents a refinement which narrows the gap between the abstract calculus and its implementation. A calculus is designed to allow reasoning on matching algorithms. The new calculus is proved to be confluent, and to simulate the original *Pure Pattern Calculus*. A family of new, matching-driven, reduction strategies is proposed.

Introduction: Dynamic Patterns

Pattern matching is a basic mechanism used to deal with algebraic data structures in functional programming languages. It allows to define a function by reasoning on the shape of the arguments. For instance, define a binary tree to be either a single data or a node with two subtrees (code on the left, in ML-like syntax). Then a function on binary trees may be defined by reasoning on the shapes generated by these two possibilities (code on the right).

```
type 'a tree =
        | Data 'a
        | Node of 'a tree * 'a tree
```

```
let f t = match t with
        | Data d            ->  <code1>
        | Node (Data d) r   ->  <code2>
        | Node l r          ->  <code3>
```

An argument given to the function `f` is first compared to (or **matched** against) the shape `Data d` (called a **pattern**). In case of success, the occurrences of `d` in <code1> are replaced by the corresponding part of the argument, and <code1> is executed. In case of failure of this first matching (the argument is not a data) the argument is matched against the second pattern, and so on until a matching succeeds or there is no pattern left.

One limit of this approach is that patterns are fixed expressions mentioning explicitly the constructors to which they can apply, which restricts polymorphism and reusability of the code. This can be improved by allowing patterns to be parametrised: one single function can be specialised in various ways by instantiating the parameters of its patterns by different constructors or even by functions building patterns. For instance in the following code, the function `f` would take an additional parameter **p** which would then be used to define the first two patterns. In this case, instantiating **p** with the constructor `Data` would yield the same function as before, but any other function building a pattern can be used for **p**!

```
let f  p t = match t with
           |  p d             ->  <code1>
           | Node ( p d) r    ->  <code2>
           | Node l r         ->  <code3>
```

However, introducing parameters and functions inside patterns deeply modifies their nature: they become dynamic objects that have to be evaluated. This disrupts the matching algorithms and introduces new evaluation behaviours. This paper intends to give tools to study these extended evaluation possibilities.

E. Bonelli (Ed.): 5th International Workshop
on Higher-Order Rewriting (HOR'10)
EPTCS 49, 2011, pp. 16–30, doi:10.4204/EPTCS.49.2

The *Pure Pattern Calculus* (*PPC*) of B. Jay and D. Kesner [JK09, Jay09] models the behaviour of dynamic patterns by using a meta-level notion of pattern matching. The present contribution analyses the content of the meta pattern matching of *PPC* (reviewed in Section 1), and proposes an explicit pattern matching calculus (Section 2) which is confluent, which simulates *PPC*, and which allows the description of new reduction strategies (Section 3.1). An extension of the explicit calculus is then discussed (Section 3.2) before a conclusion is drawn.

1 The Pure Pattern Calculus

This section only reviews some key aspects of *PPC*. Please refer to [JK09] for a complete story with more examples. The syntax of *PPC* is close to the one of λ-calculus. The main difference is the replacement of the abstraction over a variable $\lambda x.b$ by an abstraction over a pattern (with a list of matching variables) written $[\theta]p \to b$. There is also a new distinction between **variable** occurrences x and **matchable** occurrences $\hat{x}$ of a name x. Variable occurrences are usual variables which may be substituted while matchable occurrences are immutable and used as matching variables or constructors.

$$t \quad ::= \quad x \mid \hat{x} \mid tt \mid [\theta]t \to t \qquad\qquad \textit{PPC} \text{ Terms}$$

where θ is a list of names. Letter a (resp. b, p) is used to indicate a term in position of **a**rgument (resp. function **b**ody, **p**attern).

As pictured below, in the abstraction $[\theta]p \to b$ the list of names θ binds matchable occurrences in the pattern p and variable occurrences in the body b. Substitution of free variables and α-conversion are deduced (see [JK09] for details on *PPC*, or Figures 1 and 2 for a formal definition in an extended setting).

$$[x]\ x\,\hat{x} \to x\,\hat{x} \quad =_\alpha \quad [y]\ x\,\hat{y} \to y\,\hat{x}$$

One of the features of *PPC* is the use of a single syntactic application for two different meanings: the term t_1t_2 may represent either the usual **functional application** of a function t_1 to an argument t_2 or the construction of a data structure by **structural application** of a constructor to one or more arguments. The latter is invariant: any structural application is forever a data structure, whereas the functional application may be evaluated or instantiated someday (and then turn into anything else, including a structural application).

The simplest notion of pattern matching is syntactic: an argument a matches a pattern p if and only if there is a substitution σ such that $a = p^\sigma$. However, with arbitrary patterns, this solution generates non-confluent calculi [vOo90]. To recover confluence, syntactic matching can be used together with a restriction on patterns, as for instance the *rigid pattern condition* of the lambda-calculus with patterns [KvOdV08]. The alternative solution of *PPC* allows a priori any term to be a pattern, and checks the validity of patterns only a posteriori, when pattern matching is performed. In particular, the restriction on patterns applies only once the evaluation of the pattern is completed. This allows a greater freedom of evaluation and a greater polymorphism of patterns, and hence a greater expressivity.

This is done by a more subtle notion of matching, called **compound matching**, which tests whether patterns and arguments are in a so-called **matchable form**. A matchable form denotes a term which is understood as a value, or in other words a term whose current form is stable and then allows matching. Matchable forms are described in *PPC* at the meta-level by the following grammar:

$$\begin{array}{lcll} d & ::= & \hat{x} \mid dt & \quad \textit{PPC} \text{ data structures} \\ m & ::= & d \mid [\theta]t \to t & \quad \textit{PPC} \text{ matchable forms} \end{array}$$

Compound matching is then defined (still at the meta-level) by the following equations, taken in order.

$$
\begin{array}{rcll}
\{\!\{a /_\theta \hat{x}\}\!\} & := & \{x \mapsto a\} & \text{if } x \in \theta \\
\{\!\{\hat{x} /_\theta \hat{x}\}\!\} & := & \{\} & \text{if } x \notin \theta \\
\{\!\{a_1 a_2 /_\theta p_1 p_2\}\!\} & := & \{\!\{a_1 /_\theta p_1\}\!\} \uplus \{\!\{a_2 /_\theta p_2\}\!\} & \text{if } a_1 a_2 \text{ and } p_1 p_2 \text{ are matchable forms} \\
\{\!\{a /_\theta p\}\!\} & := & \bot & \text{if } a \text{ and } p \text{ are matchable forms, otherwise} \\
\{\!\{a /_\theta p\}\!\} & := & \texttt{wait} & \text{otherwise}
\end{array}
$$

Its result, called a **match** and denoted by ρ, may be a substitution (written σ), a matching failure (written $\bot$) or the special value `wait`. The latter case represents undefined cases of matching, when the pattern or the argument has still to be evaluated or instantiated before being matched.

Decomposition of compound patterns in the equations above is associated with an operation $\uplus$ of disjoint union which ensures linearity of patterns: no matching variable should be used twice in the same pattern, or confluence would be broken [Klo80]. Its formal definition is:

- $\uplus$ is commutative.
- $\bot \uplus \rho = \bot$ for any ρ (even `wait`).
- $\texttt{wait} \uplus \rho = \texttt{wait}$ for $\rho \neq \bot$.
- $\sigma_1 \uplus \sigma_2 = \bot$ if domains of σ_1 and σ_2 overlap.
- $\sigma_1 \uplus \sigma_2$ is the union of σ_1 and σ_2 otherwise.

Finally, *PPC* has to deal with a problem related to the dynamics of patterns: a matching variable may be erased from a pattern during its evaluation. In this case, no part of the argument would be bound to this matching variable and then no term would be substituted to the corresponding variable. Hence free variables would not be preserved, which would make reduction ill-defined (see Example 1). This is avoided in *PPC* by a last (meta-level) test, called *check*: the result $\{a /_\theta p\}$ of the matching of a against p is defined as follows.

- if $\{\!\{a /_\theta p\}\!\} = \bot$ then $\{a /_\theta p\} = \bot$.
- if $\{\!\{a /_\theta p\}\!\} = \sigma$ with $dom(\sigma) \neq \theta$ then $\{a /_\theta p\} = \bot$.
- if $\{\!\{a /_\theta p\}\!\} = \sigma$ with $dom(\sigma) = \theta$ then $\{a /_\theta p\} = \sigma$.

Remark that $\{a /_\theta p\}$ is not defined if $\{\!\{a /_\theta p\}\!\} = \texttt{wait}$.
Finally, the reduction $\longrightarrow_{PPC}$ of *PPC* is defined by a unique reduction rule (applied in any context):

$$([\theta]p \to b)a \quad \longrightarrow_{\beta_m} \quad b^{\{a /_\theta p\}}$$

where for any b and σ the expression b^σ denotes the application of the substitution σ to the term b, and $b^\bot$ denotes some fixed closed normal term $\bot$.

Example 1. *Let t be a* PPC *term. The redex $([x]\hat{c}\hat{x} \to x)\ (\hat{c}t)$ reduces to t: the constructor $\hat{c}$ matches itself and the matchable $\hat{x}$ is associated to t. On the other hand, $([x,y]\hat{c}\hat{x} \to xy)\ (\hat{c}t)$ reduces to $\bot$: whereas the compound matching is defined and successful, the check fails since there is no match for y and the result would be ty where y appears as a free variable. The redex $([x]\hat{c}\hat{x} \to x)\ (\hat{c})$ also reduces to $\bot$ since a constructor will never match a structural application. And last, $([x]y\hat{x} \to x)\ (\hat{c}t)$ is not a redex since the pattern $y\hat{x}$ has to be instantiated.*

2 Explicit Matching

This section defines the *Pure Pattern Calculus with Explicit Matching* (PPC_{EM}), a calculus which gives an account of all the steps of a pattern matching process of *PPC*. The first point discussed is the identification of structural application (Section 2.1). An explicit calculus is then fully detailed (Section 2.2) and some of its basic properties are proved (Section 2.3). Explicit formulations of simpler pattern calculi already appear in [CK04, For02, CFK04].

2.1 Explicit Data Structures

Firstly, a new syntactic construct is introduced to discriminate between functional and structural applications (as in [FMS06] for the rewriting calculus for instance). Any application is supposed functional *a priori*, and two reduction rules propagate structural information. The explicit structural application of t to u is written $t \bullet u$.

$$\begin{array}{rcll} t & ::= & x \mid \hat{x} \mid tt \mid t \bullet t \mid [\theta]t \to t & PPC_\bullet \text{ terms} \\ d & ::= & \hat{x} \mid t \bullet t & PPC_\bullet \text{ data structures} \end{array}$$

$$\begin{array}{rcl} \hat{x}\, t & \longrightarrow_\bullet & \hat{x} \bullet t \\ (t_1 \bullet t_2)\, t_3 & \longrightarrow_\bullet & (t_1 \bullet t_2) \bullet t_3 \end{array}$$

The identity morphism embeds *PPC* into $PPC_\bullet$. The subset of $PPC_\bullet$ defined by *PPC* is referred to as the set of pure terms. On the other hand, a "forgetful" morphism maps $PPC_\bullet$ terms back to *PPC* terms (or pure terms):

$$\begin{array}{rcl} [\![x]\!] & := & x \\ [\![\hat{x}]\!] & := & \hat{x} \\ [\![t_1 t_2]\!] & := & [\![t_1]\!][\![t_2]\!] \\ [\![t_1 \bullet t_2]\!] & := & [\![t_1]\!][\![t_2]\!] \\ [\![[\theta]p \to b]\!] & := & [\theta][\![p]\!] \to [\![b]\!] \end{array}$$

Some $PPC_\bullet$ data structures are not mapped to data structures of *PPC*, for instance $([\theta]p \to b) \bullet a$. However, for any pure term t, if $t \longrightarrow_\bullet^* t'$ and t' is a $PPC_\bullet$ data structure, then t is a *PPC* data structure (proof by induction on t). One can also observe that for every *PPC* data structure t, there exists a reduction $t \longrightarrow_\bullet^* t'$ with t' a $PPC_\bullet$ data structure. Call **well-formed** a term t such that $[\![t]\!] \longrightarrow_\bullet^* t$.

2.2 Explicit Pattern Matching

Another new syntactic object has to be introduced to represent an ongoing matching operation. The basic information contained in such an object are: the list of matching variables, a partial result recording what has already been computed, and a representation of what has still to be solved.

This new object is called **matching** and is written $\langle \theta | \mu | \Delta \rangle$ with θ a list of names, μ a **decided match** (that means, $\perp$ or a substitution), and Δ the collection of submatchings that have still to be solved (a multiset of pairs of terms). For now on, we will consider only decided matches, written μ (`wait` does not exist as such in PPC_{EM}).

The complete new grammar is:

$$\begin{array}{rcll} t & ::= & x \mid \hat{x} \mid tt \mid t \bullet t \mid [\theta]t \to t \mid t\, \langle \theta | \mu | \Delta \rangle & PPC_{EM} \text{ terms} \\ d & ::= & \hat{x} \mid t \bullet t & PPC_{EM} \text{ data structures} \\ m & ::= & d \mid [\theta]t \to t & PPC_{EM} \text{ matchable forms} \end{array}$$

The set of free names of a term t is $fn(t) = fv(t) \cup fm(t)$.

Free variables

$$
\begin{array}{rcl}
fv(x) & := & \{x\} \\
fv(\hat{x}) & := & \emptyset \\
fv(t_1 t_2) & := & fv(t_1) \cup fv(t_2) \\
fv(t_1 \bullet t_2) & := & fv(t_1) \cup fv(t_2) \\
fv([\theta]p \to b) & := & fv(p) \cup (fv(b) \setminus \theta) \\
fv(t\,\langle\theta|\mu|\Delta\rangle) & := & (fv(t) \setminus \theta) \cup fv(codom(\mu)) \cup fv(\Delta)
\end{array}
$$

Free matchables

$$
\begin{array}{rcl}
fm(x) & := & \emptyset \\
fm(\hat{x}) & := & \{x\} \\
fm(t_1 t_2) & := & fm(t_1) \cup fm(t_2) \\
fm(t_1 \bullet t_2) & := & fm(t_1) \cup fm(t_2) \\
fm([\theta]p \to b) & := & (fm(p) \setminus \theta) \cup fm(b) \\
fm(t\,\langle\theta|\mu|\Delta\rangle) & := & fm(t) \cup fm(codom(\mu)) \cup fm(\pi_1(\Delta)) \cup (fm(\pi_2(\Delta)) \setminus \theta)
\end{array}
$$

where if $\Delta = (a_1, p_1)...(a_n, p_n)$ then $fm(\pi_1(\Delta)) = \bigcup_i fm(a_i)$ and $fm(\pi_2(\Delta)) = \bigcup_i fm(p_i)$.

Figure 1: Free names of a PPC_{EM} term

$$
\begin{array}{rcll}
x^\sigma & := & \sigma_x & x \in dom(\sigma) \\
x^\sigma & := & x & x \notin dom(\sigma) \\
\hat{x}^\sigma & := & \hat{x} & \\
(tu)^\sigma & := & t^\sigma u^\sigma & \\
(t \bullet u)^\sigma & := & t^\sigma \bullet u^\sigma & \\
([\theta]p \to b)^\sigma & := & ([\theta]p^\sigma \to b^\sigma) & \theta \cap (dom(\sigma) \cup fn(\sigma)) = \emptyset \\
(t\,\langle\theta|\mu|\Delta\rangle)^\sigma & := & t^\sigma\,\langle\theta|\mu^\sigma|\Delta^\sigma\rangle & \theta \cap (dom(\sigma) \cup fn(\sigma)) = \emptyset
\end{array}
$$

where in Δ^σ (resp. μ^σ) the substitution propagates in all terms of Δ (resp. of the codomain of μ).

Figure 2: Substitution in PPC_{EM}

Initialisation

$$([\theta]p \rightarrow b)a \quad \longrightarrow_B \quad b\langle\theta|\emptyset|(a,p)\rangle$$

Structural application

$$\begin{array}{rcl} \hat{x}\, t & \longrightarrow_\bullet & \hat{x} \bullet t \\ (t_1 \bullet t_2)\, t_3 & \longrightarrow_\bullet & (t_1 \bullet t_2) \bullet t_3 \end{array}$$

Matching

Since Δ has been defined as a multiset of pairs of terms, its elements are not ordered. In the following rules $(a,p)\Delta$ denotes the (multiset) union of Δ with the singleton $\{(a,p)\}$.

The first three matching rules are for successful matching steps.

$$\begin{array}{rcll} b\langle\theta|\mu|(a,\hat{x})\Delta\rangle & \longrightarrow_m & b\langle\theta|\mu \uplus \{x \mapsto a\}|\Delta\rangle & \text{if } x \in \theta \text{ and } fn(a) \cap \theta = \emptyset \\ b\langle\theta|\mu|(\hat{x},\hat{x})\Delta\rangle & \longrightarrow_m & b\langle\theta|\mu|\Delta\rangle & \text{if } x \notin \theta \\ b\langle\theta|\mu|(a_1 \bullet a_2, p_1 \bullet p_2)\Delta\rangle & \longrightarrow_m & b\langle\theta|\mu|(a_1,p_1)(a_2,p_2)\Delta\rangle & \end{array}$$

The last six matching rules are for failure, and could be summed up as "for any other matchable forms a and p, let $b\langle\theta|\mu|(a,p)\Delta\rangle$ reduce to $b\langle\theta|\bot|\Delta\rangle$".

$$\begin{array}{rcll} b\langle\theta|\mu|(\hat{y},\hat{x})\Delta\rangle & \longrightarrow_m & b\langle\theta|\bot|\Delta\rangle & \text{if } x \notin \theta \text{ and } x \neq y \\ b\langle\theta|\mu|(a_1 \bullet a_2,\hat{x})\Delta\rangle & \longrightarrow_m & b\langle\theta|\bot|\Delta\rangle & \text{if } x \notin \theta \\ b\langle\theta|\mu|([\theta_a]p_a \rightarrow b_a,\hat{x})\Delta\rangle & \longrightarrow_m & b\langle\theta|\bot|\Delta\rangle & \text{if } x \notin \theta \\ b\langle\theta|\mu|(\hat{x}, p_1 \bullet p_2)\Delta\rangle & \longrightarrow_m & b\langle\theta|\bot|\Delta\rangle & \\ b\langle\theta|\mu|([\theta_a]p_a \rightarrow b_a, p_1 \bullet p_2)\Delta\rangle & \longrightarrow_m & b\langle\theta|\bot|\Delta\rangle & \\ b\langle\theta|\mu|(a, [\theta_p]p_p \rightarrow b_p)\Delta\rangle & \longrightarrow_m & b\langle\theta|\bot|\Delta\rangle & \end{array}$$

Resolution

$$\begin{array}{rcll} b\langle\theta|\sigma|\emptyset\rangle & \longrightarrow_r & b^\sigma & \text{if } dom(\sigma) = \theta \quad (\textbf{substitution rule}) \\ b\langle\theta|\sigma|\emptyset\rangle & \longrightarrow_r & \bot & \text{if } dom(\sigma) \neq \theta \\ b\langle\theta|\bot|\Delta\rangle & \longrightarrow_r & \bot & \end{array}$$

Figure 3: Rules of PPC_{EM}

A pure term of PPC_{EM} is a term without any structural application or matching (that means a *PPC* term). As in *PPC*, the symbol $\perp$ used as a term denotes a fixed closed pure normal term.

Free variables and matchables are defined in Figure 1 as a natural extension of *PPC* mechanisms to explicit matching. Similarly, a notion of (meta-level) substitution is deduced from this definition (Figure 2). Finally, a notion of α-conversion is associated, and from now, on it is supposed that all bound names in a term are different, and disjoint from free names.

New rules for matching are of three kinds: an *initialisation rule* $\longrightarrow_B$ which triggers a new matching operation, several *matching rules* $\longrightarrow_m$ corresponding to all possible elementary matching steps and three *resolution rules* $\longrightarrow_r$ that apply the result of a completed matching. The complete set of rules of PPC_{EM} is given in Figure 3.

Reduction $\longrightarrow_{EM}$ of PPC_{EM} is defined by application of any rule of $\longrightarrow_B$, $\longrightarrow_\bullet$, $\longrightarrow_m$ or $\longrightarrow_r$ in any context. The subsystem $\longrightarrow_p = \longrightarrow_\bullet \cup \longrightarrow_m \cup \longrightarrow_r$ computes (when possible) already existing pattern matchings but does not create new ones.

2.3 Confluence and Simulation properties

This section states and proves four theorems on basic properties of PPC_{EM} and its links with *PPC*. The first one is a result on the normalization of already existing pattern matchings.

Theorem 1. *$\longrightarrow_p$ is confluent and strongly normalizing.*

Proof.

- We define two well-founded orders $\prec_\mathscr{N}$ and $\prec_\mathscr{S}$, whose lexicographic product contains ${}_p\!\longleftarrow$. This will enforce strong normalization.
  - $\prec_\mathscr{N}$ sorts terms with respect to the nesting of matchings. It is based on an over-approximation of the depth of potentially nested matchings (matchings that are syntactically nested or that may become such after some substitutions). For any lists of names θ_i, decided matches μ_i, and lists of pairs of terms Δ_i, the sequence $\langle\theta_1|\mu_1|\Delta_1\rangle;\ldots;\langle\theta_n|\mu_n|\Delta_n\rangle$ is called a potentially nested chain of length n if for each $i\in\{1\ldots n-1\}$ one of these conditions holds:
    - **Nesting:** $\langle\theta_{i+1}|\mu_{i+1}|\Delta_{i+1}\rangle$ appears in Δ_i or in the codomain of μ_i.
    - **Potential nesting:** a variable of θ_{i+1} appears in Δ_i or in the codomain of μ_i.

    The set of maximal chains of a term t is the set of all potentially nested chains that can be built using the matchings appearing in t and that can not be extended (neither by the left nor by the right) using other matchings of t. For this extraction, remember that all bound names in t are supposed to be different, and disjoint from free names. The depth of t is the multiset of the lengths of the maximal chains of t.

    Example 2. *Write* $t = \hat{c}\,\langle\emptyset|\emptyset|(x,\hat{c})(x,\hat{c})\rangle\,\langle x|x\mapsto y\,\langle y|\emptyset|(\hat{c},\hat{y})\rangle\,|\emptyset\rangle$. *The term t contains three matchings and has one maximal chain of length 3, which is*

    $$\langle\emptyset|\emptyset|(x,\hat{c})(x,\hat{c})\rangle\,;\,\langle x|x\mapsto y\,\langle y|\emptyset|(\hat{c},\hat{y})\rangle\,|\emptyset\rangle\,;\,\langle y|\emptyset|(\hat{c},\hat{y})\rangle$$

    The reduction $t\ \longrightarrow_r\ t' = \hat{c}\,\langle\emptyset|\emptyset|(y_1\,\langle y_1|\emptyset|(\hat{c},\hat{y_1})\rangle,\hat{c})\,(y_2\,\langle y_2|\emptyset|(\hat{c},\hat{y_2})\rangle,\hat{c})\rangle$ *yields a new term t' which still contains three matchings (one was reduced and disappeared but another one was duplicated) and admits two maximal chains of length 2, namely*

    $$\langle\emptyset|\emptyset|(y_1\,\langle y_1|\emptyset|(\hat{c},\hat{y_1})\rangle,\hat{c})\,(y_2\,\langle y_2|\emptyset|(\hat{c},\hat{y_2})\rangle,\hat{c})\rangle\,;\,\langle y_1|\emptyset|(\hat{c},\hat{y_1})\rangle$$

    $$\langle\emptyset|\emptyset|(y_1\,\langle y_1|\emptyset|(\hat{c},\hat{y_1})\rangle,\hat{c})\,(y_2\,\langle y_2|\emptyset|(\hat{c},\hat{y_2})\rangle,\hat{c})\rangle\,;\,\langle y_2|\emptyset|(\hat{c},\hat{y_2})\rangle$$

The usual order on natural integers gives a well-founded order on the lengths of potentially nested chains. $\prec_{\mathscr{N}}$ is defined as the multiset extension of this order, applied to the depths of terms. It strictly decreases for any reduction by the substitution rule, and is less or equal for any other reduction.

– $\prec_{\mathscr{S}}$ is the natural order on the size of terms, defined as follows:

$$\begin{array}{rcl} \mathscr{S}(x) & := & 1 \\ \mathscr{S}(\hat{x}) & := & 1 \\ \mathscr{S}(t_1 t_2) & := & \mathscr{S}(t_1)+\mathscr{S}(t_2)+2 \\ \mathscr{S}(t_1 \bullet t_2) & := & \mathscr{S}(t_1)+\mathscr{S}(t_2)+1 \\ \mathscr{S}([\theta]p \to b) & := & \mathscr{S}(p)+\mathscr{S}(b) \\ \mathscr{S}(b\langle\theta|\mu|\Delta\rangle) & := & \mathscr{S}(b)+\mathscr{S}(\bot)+\sum_{x\in dom(\mu)}\mathscr{S}(\mu_x)+\sum_{(a,p)\in_k\Delta}k(\mathscr{S}(a)+\mathscr{S}(p)) \end{array}$$

where we write $e \in_k \Delta$ when the element e appears in the multiset Δ with multiplicity k. $\prec_{\mathscr{S}}$ strictly decreases for any reduction except by the substitution rule.

- Matching rules generate some critical pairs, most of which are trivially convergent. The most subtle case is the reduction of a non linear matching:

$$\langle\theta|\mu \uplus \{x\mapsto a_1\}|(a_2,\hat{x})\Delta\rangle \quad {}_p\longleftarrow \quad \langle\theta|\mu|(a_1,\hat{x})(a_2,\hat{x})\Delta\rangle \quad \longrightarrow_p \quad \langle\theta|\mu\uplus\{x\mapsto a_2\}|(a_1,\hat{x})\Delta\rangle$$

Since $\uplus$ is a disjoint union of substitutions, both sides can be reduced to $\langle\theta|\bot|\Delta\rangle$.
Finally, $\longrightarrow_p$ is weakly confluent, and then confluent by Newman's Lemma [Ter03].

□

The second theorem states the confluence of $\longrightarrow_{EM}$. Since the reduction of PPC_{EM} is defined by several rules, the result does not fall into the modular framework of [JK09]. It is proved here directly by the Tait and Martin-Löf's technique. The main construction of the proof is the definition (in Figure 4) of a parallel reduction relation $\Longrightarrow$ enjoying the diamond property (Lemma 3). The relation $\Longrightarrow$ is first linked to $\longrightarrow_{EM}$ in Lemma 1.

Lemma 1. $\longrightarrow_{EM} \;\subseteq\; \Longrightarrow \;\subseteq\; \longrightarrow^*_{EM}$

Proof.

- $\longrightarrow_{EM} \;\subseteq\; \Longrightarrow$ by induction on the definition of $\longrightarrow_{EM}$.
- $\Longrightarrow \;\subseteq\; \longrightarrow^*_{EM}$ by induction on the definition of $\Longrightarrow$.

□

Lemma 2. *If $t \Longrightarrow t'$ and $\sigma \Longrightarrow \sigma'$ then $t^\sigma \Longrightarrow t'^{\sigma'}$.*

Proof. By induction on the derivation of $t \Longrightarrow t'$. □

Lemma 3. $\Longleftarrow\!\!\Longrightarrow \;\subseteq\; \Longrightarrow\Longleftarrow$

Proof. Suppose $t_1 \Longleftarrow t \Longrightarrow t_2$. Induction on the derivations of $t \Longrightarrow t_1$ and $t \Longrightarrow t_2$:

- If one of the reductions is by "Id", the conclusion is immediate.
- If one reduction is by a "Cgr" rule, and the other by a "Cgr", "Init", "Struct", or "Match" rule, then the induction hypothesis applies straightforwardly.

Id.

$$t \Longrightarrow t$$

Cgr.

$$\frac{t_1 \Longrightarrow t_1' \qquad t_2 \Longrightarrow t_2'}{t_1 t_2 \Longrightarrow t_1' t_2'} \qquad \frac{t_1 \Longrightarrow t_1' \qquad t_2 \Longrightarrow t_2'}{t_1 \bullet t_2 \Longrightarrow t_1' \bullet t_2'} \qquad \frac{p \Longrightarrow p' \qquad b \Longrightarrow b'}{[\theta]p \to b \Longrightarrow [\theta]p' \to b'}$$

$$\frac{b \Longrightarrow b' \qquad \mu \Longrightarrow \mu' \qquad \Delta \Longrightarrow \Delta'}{b\,\langle\theta|\mu|\Delta\rangle \Longrightarrow b'\,\langle\theta|\mu'|\Delta'\rangle}$$

Init.

$$\frac{p \Longrightarrow p' \qquad b \Longrightarrow b' \qquad a \Longrightarrow a'}{([\theta]p \to b)a \Longrightarrow b'\,\langle\theta|\emptyset|(a',p')\rangle}$$

Struct.

$$\frac{t \Longrightarrow t'}{\hat{x}\,t \Longrightarrow \hat{x} \bullet t'} \qquad \frac{t_1 \Longrightarrow t_1' \qquad t_2 \Longrightarrow t_2' \qquad t_3 \Longrightarrow t_3'}{(t_1 \bullet t_2)\;t_3 \Longrightarrow (t_1' \bullet t_2') \bullet t_3'}$$

Match.

$$\frac{b \Longrightarrow b' \qquad \mu \Longrightarrow \mu' \qquad a \Longrightarrow a' \qquad \Delta \Longrightarrow \Delta'}{b\,\langle\theta|\mu|(a,\hat{x})\Delta\rangle \Longrightarrow b'\,\langle\theta|\mu' \uplus \{x \mapsto a'\}|\Delta'\rangle}\; x \in \theta, fn(a) \cap \theta = \emptyset$$

$$\frac{b \Longrightarrow b' \qquad \mu \Longrightarrow \mu' \qquad \Delta \Longrightarrow \Delta'}{b\,\langle\theta|\mu|(\hat{x},\hat{x})\Delta\rangle \Longrightarrow b'\,\langle\theta|\mu'|\Delta'\rangle}\; x \notin \theta$$

$$\frac{b \Longrightarrow b' \qquad \mu \Longrightarrow \mu' \qquad \Delta \Longrightarrow \Delta' \qquad a_i \Longrightarrow a_i' \qquad p_i \Longrightarrow p_i'}{b\,\langle\theta|\mu|(a_1 \bullet a_2, p_1 \bullet p_2)\Delta\rangle \Longrightarrow b'\,\langle\theta|\mu'|(a_1', p_1')(a_2', p_2')\Delta'\rangle}$$

$$\frac{b \Longrightarrow b' \qquad \Delta \Longrightarrow \Delta'}{b\,\langle\theta|\mu|(a,p)\Delta\rangle \Longrightarrow b'\,\langle\theta|\bot|\Delta'\rangle}\; a \text{ and } p \text{ other matchable forms}$$

Res.

$$\frac{b \Longrightarrow b' \qquad \sigma \Longrightarrow \sigma'}{b\,\langle\theta|\sigma|\emptyset\rangle \Longrightarrow (b')^{\sigma'}}\; dom(\sigma) = \theta \qquad \frac{dom(\sigma) \neq \theta}{b\,\langle\theta|\sigma|\emptyset\rangle \Longrightarrow \bot} \qquad b\,\langle\theta|\bot|\Delta\rangle \Longrightarrow \bot$$

As in Figure 3, the last "Match" rule could be explicited in six fail rules.
Parallel reduction is straightforwardly extended:

- to decided matches (μ) by applying $\Longrightarrow$ to all terms in the codomain of a substitution (with moreover $\bot \Longrightarrow \bot$).
- to multisets of pairs of terms (Δ) by applying $\Longrightarrow$ to all terms.

Figure 4: Definition of parallel reduction relation $\Longrightarrow$

- If one reduction is by a “Cgr” rule and the other by a “Res” rule, there is one non trivial case: suppose $t_1 \langle\theta|\sigma_1|\emptyset\rangle \Longleftarrow t\langle\theta|\sigma|\emptyset\rangle \Longrightarrow t_2^{\sigma_2}$. By induction hypothesis there are t_3 and σ_3 such that $t_1 \Longrightarrow t_3 \Longleftarrow t_2$ and $\sigma_1 \Longrightarrow \sigma_3 \Longleftarrow \sigma_2$. Then we can derive $t_1 \langle\theta|\sigma_1|\emptyset\rangle \Longrightarrow t_3^{\sigma_3}$. Finally, by Lemma 2 we conclude that $t_2^{\sigma_2} \Longrightarrow t_3^{\sigma_3}$.
- If both reductions are by a “Init” rule, then the induction hypotheses apply straightforwardly.
- Idem for two “Struct” or two “Match” rules.
- Case where both reductions are by a “Res” rule. Reductions to $\perp$ are straightforward. Then consider the following case: $t_1^{\sigma_1} \Longleftarrow t\langle\theta|\sigma|\emptyset\rangle \Longrightarrow t_2^{\sigma_2}$. By induction hypotheses $t_1 \Longrightarrow t_3 \Longleftarrow t_2$ and $\sigma_1 \Longrightarrow \sigma_3 \Longleftarrow \sigma_2$. By Lemma 2 $t_1^{\sigma_1} \Longrightarrow t_3^{\sigma_3} \Longleftarrow t_2^{\sigma_2}$.

□

Theorem 2. *PPC_{EM} is confluent.*

Proof. Since $\Longrightarrow$ has the diamond property (Lemma 3), its transitive closure $\Longrightarrow^*$ also enjoys the diamond property ([Ter03]). Moreover Lemma 1 implies $\longrightarrow_{EM}^* = \Longrightarrow^*$, and then $\longrightarrow_{EM}^*$ enjoys the diamond property. Finally, $\longrightarrow_{EM}$ is confluent. □

The last two theorems establish a link between the calculus with explicit matching PPC_{EM} and the original implicit PPC.

Lemma 4. *If $\{\!\{a /_\theta\ p\}\!\} = \mu$ with μ a decided match, then for any μ_0 and Δ there are μ' with $[\![\mu']\!] = \mu$ and a reduction*

$$\langle\theta|\mu_0|(a,p)\Delta\rangle\ (\longrightarrow_\bullet \cup \longrightarrow_m)^*\ \langle\theta|\mu_0 \uplus \mu'|\Delta\rangle$$

Proof. Induction on $\{\!\{a /_\theta\ p\}\!\}$.

- $\{\!\{a /_\theta\ \hat{x}\}\!\}$ with $x \in \theta$ or $\{\!\{\hat{x} /_\theta\ \hat{x}\}\!\}$ with $x \notin \theta$: immediate.
- $\{\!\{aa_0 /_\theta\ pp_0\}\!\}$ with aa_0 and pp_0 matchable forms. Hence $a = a_n...a_1$ and $p = p_m...p_1$ with a_n and p_m constructors. Then $a_n...a_1a_0 \longrightarrow_\bullet^* a_n \bullet ... \bullet a_1 \bullet a_0$ and $p_m...p_1p_0 \longrightarrow_\bullet^* p_m \bullet ... \bullet p_1 \bullet p_0$. Suppose $n \geq m$, then $\{\!\{aa_0 /_\theta\ pp_0\}\!\} = \{\!\{a_m...a_n /_\theta\ p_n\}\!\} \uplus \{\!\{a_{n-1} /_\theta\ p_{n-1}\}\!\} \uplus ... \uplus \{\!\{a_0 /_\theta\ p_0\}\!\}$ and $\langle\theta|\mu_0|(a_n \bullet ... \bullet a_0, p_m \bullet ... \bullet p_0)\Delta\rangle \longrightarrow_m^* \langle\theta|\mu_0|(a_m \bullet ... \bullet a_n, p_n)(a_{n-1}, p_{n-1})...(a_0, p_0)\Delta\rangle$. Case on $p_n = \hat{x}$:
  - If $x \in \theta$ then the matching reduces to $\langle\theta|\mu_0 \uplus \{x \mapsto a_m \bullet ... \bullet a_n\}|(a_{n-1}, p_{n-1})...(a_0, p_0)\Delta\rangle$.
  - If $x \notin \theta$ then the matching reduces to $\langle\theta|\mu_0'|(a_{n-1}, p_{n-1})...(a_0, p_0)\Delta\rangle$ with $\mu_0' = \mu_0$ or $\mu_0' = \perp$.

  In any of these two cases, the induction hypothesis gives the conclusion. In the case where $m > n$, the same method allows to derive a reduction to $\perp$.
- Cases of matching failure: for instance $\{\!\{\hat{x} /_\theta\ \hat{y}t\}\!\}$. The following reduction gives the conclusion: $\langle\theta|\mu_0|(\hat{x}, \hat{y}t)\Delta\rangle \longrightarrow_\bullet \langle\theta|\mu_0|(\hat{x}, \hat{y} \bullet t)\Delta\rangle \longrightarrow_m \langle\theta|\perp|\Delta\rangle$.

□

Theorem 3. *For any terms t and t' of* PPC*, if $t \longrightarrow_{PPC} t'$ then $t \longrightarrow_{EM}^* t'$.*

Proof. Suppose $t \longrightarrow_{PPC} t'$. There is a context $C[]$ such that $t = C[(([\theta]p \to b)a] \longrightarrow_{PPC} C[b'] = t'$ and $\{\!\{a /_\theta\ p\}\!\} = \mu$ with μ a decided match.
By Lemma 4 $([\theta]p \to b)a \longrightarrow_B b\langle\theta|\emptyset|(a,p)\rangle\ (\longrightarrow_\bullet \cup \longrightarrow_m)^*\ b\langle\theta|\mu|\emptyset\rangle$.

Case on μ:

- If $\mu = \bot$ then $b' = \bot$ and $b\,\langle\theta|\bot|\emptyset\rangle \longrightarrow_r \bot$.
- Else $\mu = \sigma$ and:
  - If $dom(\sigma) = \theta$ then $b' = b^\sigma$ and $b\,\langle\theta|\sigma|\emptyset\rangle \;\longrightarrow_r\; b^\sigma$.
  - Else $b' = \bot$ and $b\,\langle\theta|\bot|\emptyset\rangle \;\longrightarrow_r\; \bot$.

□

The map $[\![\cdot]\!]$ is naturally extended to any PPC_{EM} term, set of PPC_{EM} terms and decided match, as well as the notion of well-formedness. Then, for any μ and Δ not containing any explicit matching, define the semantics of the matching $\langle\theta|\mu|\Delta\rangle$ by:

$$[\![\theta|\mu|\Delta]\!] = [\![\mu]\!] \uplus \left(\biguplus_{(a,p)\in\Delta} \{\!\{[\![a]\!] /_\theta [\![p]\!]\}\!\} \right)$$

Note that the semantics can be `wait`.

Lemma 5. *For any well-formed μ, μ', Δ and Δ' which do not contain any explicit matching, if $\langle\theta|\mu|\Delta\rangle \longrightarrow_m \langle\theta|\mu'|\Delta'\rangle$ or $\langle\theta|\mu|\Delta\rangle \longrightarrow_\bullet \langle\theta|\mu'|\Delta'\rangle$ then $[\![\theta|\mu|\Delta]\!] = [\![\theta|\mu'|\Delta']\!]$.*

Proof. Case on the reduction rules. □

Lemma 6 ([JK09]). *If $t \longrightarrow_{PPC} t'$, then $t^\sigma \longrightarrow_{PPC} t'^\sigma$.*

Let t be a PPC_{EM} term, and t' the unique normal form of t by $\longrightarrow_p$. Write $t\!\downarrow$ and call purification of t the term $[\![t']\!]$. Note that the purification may not be a pure term if there is an unsolvable matching in it.

Theorem 4. *For any well-formed terms t and t' of PPC_{EM}, if $t \longrightarrow_{EM} t'$ and $t\!\downarrow$ and $t'\!\downarrow$ are pure, then $t\!\downarrow = t'\!\downarrow$ or $t\!\downarrow \longrightarrow_{PPC} t'\!\downarrow$.*

Proof. Induction on $t \longrightarrow_{EM} t'$.

- Case $t = ([\theta]p \to b)a \;\longrightarrow_B\; b\,\langle\theta|\emptyset|(p,a)\rangle = t'$. The term $t'\!\downarrow$ is pure, then there is a sequence $b\!\downarrow \langle\theta|\emptyset|(p\!\downarrow, a\!\downarrow)\rangle \;(\longrightarrow_\bullet \cup \longrightarrow_m)^*\; b\!\downarrow \langle\theta|\mu|\Delta\rangle \longrightarrow_r t''$ where $[\![t'']\!] = t'\!\downarrow$ and where $\Delta = \emptyset$ or $\mu = \bot$. By Lemma 5, $[\![\mu]\!] = \{\!\{a\!\downarrow \;/_\theta\; p\!\downarrow\}\!\}$. Then, by case on matching resolution, $t\!\downarrow \longrightarrow_{PPC} [\![t'']\!] = t'\!\downarrow$.
- Other base cases: if $t \longrightarrow_p t'$, then $t\!\downarrow = t'\!\downarrow$.
- Case $t = b\,\langle\theta|\mu|\Delta\rangle \longrightarrow_{EM} b'\,\langle\theta|\mu|\Delta\rangle = t'$. The term $t\!\downarrow$ is pure. Then $\langle\theta|\mu|\Delta\rangle \longrightarrow_p^* \langle\theta|\mu'|\Delta'\rangle$ where $\Delta' = \emptyset$ or $\mu' = \bot$. If $\mu' = \bot$ or $dom(\mu') \neq \theta$, then $t\!\downarrow = t'\!\downarrow = \bot$. Suppose $\Delta' = \emptyset$ and $\mu' = \sigma$ with $dom(\sigma) = \theta$. Hence $t\!\downarrow = (b\!\downarrow)^\sigma$ and $t'\!\downarrow = (b'\!\downarrow)^\sigma$. By induction hypothesis $b\!\downarrow \longrightarrow_{PPC} b'\!\downarrow$, and then by Lemma 6 $t\!\downarrow \longrightarrow_{PPC} t'\!\downarrow$.
- Other inductive cases are straightforward.

□

This section introduced the new calculus PPC_{EM} for explicit matching with dynamic patterns, and proved its confluence. It also expressed a bidirectional simulation between PPC and PPC_{EM}: first any reduction of PPC is reflected in PPC_{EM} by a sequence. On the other hand, a reduction of PPC_{EM} can be mapped on zero or one step of PPC if and only if its source and its target are well-formed and can be purified. Next section discusses how this new calculus can be used.

3 Discussion

3.1 Reduction Strategies

Pattern matching raises at least two new issues concerning reduction strategies (*i.e.* the evaluation order of programs). One is related to the order in which pattern matching steps are performed, the other concerns the amount of evaluation of the pattern and of the argument performed before pattern matching is solved.

Some remarks about the order of pattern matching steps.
PPC_{EM} uses a multiset as the third component of a matching $\langle\theta|\mu|\Delta\rangle$ to represent all the remaining work. The calculus is thus able to cover all the possible orders of pattern matching steps. A particular strategy may be enforced by giving more structure to the multiset Δ and by adapting the matching reduction rules.

Example 3. *Suppose that Δ is now a* list *of pairs of terms, and $(a,p)\Delta$ denotes the usual "cons": it builds the list whose head is (a,p) and whose tail is Δ. Then the rules of Figure 3 implement a depth-first, left-to-right pattern matching algorithm.*

Example 4. *Now assume the list structure of Example 3 and replace the right member of the reduction rule $\langle\theta|\mu|(a_1 \bullet a_2, p_1 \bullet p_2)\Delta\rangle \longrightarrow_m \langle\theta|\mu|(a_1,p_1)(a_2,p_2)\Delta\rangle$ by $\langle\theta|\mu|\Delta(a_1,p_1)(a_2,p_2)\rangle$. Then pattern matching is done in a completely different order!*

More generally, if some permutations of the elements of Δ are allowed, lots of richer matching behaviours may be described in PPC_{EM}.

Pattern and argument evaluation: what is needed?
In *PPC*, a naive evaluation strategy for a term $([\theta]p \to b)a$ could be: evaluate the pattern p and the argument a, then solve the matching (atomically). As the usual call-by-value, this solution may perform unneeded evaluation of the argument, for instance in parts that are not reused in the body b of the function. The most basic solution to this problem, call-by-name, allows the substitution of non-evaluated arguments. But how can such a solution be described in a pattern calculus?

In the context of pattern matching, some evaluation of the argument has to be done before pattern matching is solved. However the exact amount of needed evaluation depends on the pattern. Hence pattern matching enforces some kind of call-by-value where the notion of value is context-sensitive. Moreover, even the evaluation of the pattern may depend on the argument!

This makes the description of a strategy performing a minimal evaluation of the dynamic pattern and the argument rather difficult. One may keep for the object-level a compact formalism like *PPC* by defining complex meta-level operations finely parametrised by terms. This is done in [KLR10] to describe standard reductions in a simpler pattern calculus. In contrast to this solution, we want to show here how the richer syntax of PPC_{EM} allows a simple description of such a reduction strategy.

Indeed PPC_{EM} allows to interleave pattern and argument reduction with pattern matching steps. This finer control allows for instance an easy definition of a "matching-driven" reduction, as pictured in Figure 5.

The idea here is to trigger pattern matchings as soon as possible. Then the pattern and the argument are evaluated until they become matchable, and one or more pattern matching steps are performed before the story goes on. A formal definition of a strategy implementing this picture is by restricting the reduction under a context to the only four rules given in Figure 6.

Moreover, it can be checked that the list structure of Example 3 associated with the rules of Figure 3 and the context rules of Figure 6 gives a deterministic reduction strategy for PPC_{EM} (which means that any term has at most one authorised redex).

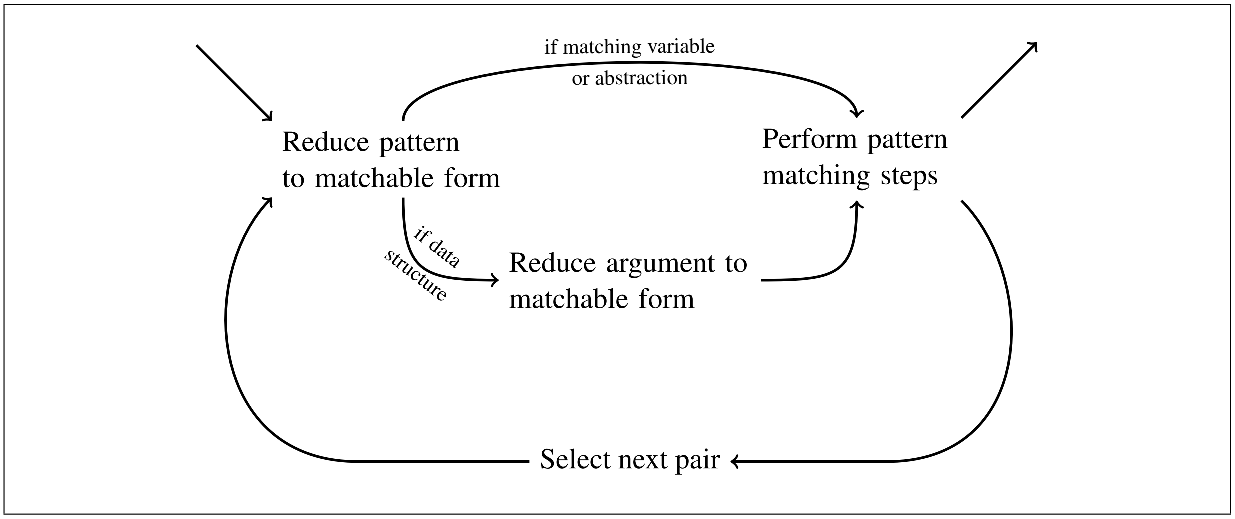

Figure 5: Matching-driven reduction strategy

$$\frac{t_1 \longrightarrow t_1'}{t_1 t_2 \longrightarrow t_1' t_2}$$

$$\frac{p \longrightarrow p'}{b\langle\theta|\mu|(a,p)\Delta\rangle \longrightarrow b\langle\theta|\mu|(a,p')\Delta\rangle}$$

$$\frac{a \longrightarrow a'}{b\langle\theta|\mu|(a,\hat{x})\Delta\rangle \longrightarrow b\langle\theta|\mu|(a',\hat{x})\Delta\rangle}\; x \notin \theta$$

$$\frac{a \longrightarrow a'}{b\langle\theta|\mu|(a,p_1 \bullet p_2)\Delta\rangle \longrightarrow b\langle\theta|\mu|(a',p_1 \bullet p_2)\Delta\rangle}$$

Figure 6: Context rules for matching-driven reduction

$$\begin{array}{rcll}
b\langle\theta|\tau|(a,\hat{x})\Delta\rangle & \longrightarrow_r & b^{\{x\mapsto a\}}\langle\theta|\tau\cup\{x\}|\Delta\rangle & \text{if } x\in\theta,\, x\notin\tau \text{ and } fn(a)\cap\theta=\emptyset \\
b\langle\theta|\theta|\emptyset\rangle & \longrightarrow_r & b^{\sigma} & \\
b\langle\theta|\tau|\emptyset\rangle & \longrightarrow_r & \bot & \text{if } \tau\neq\theta \\
b\langle\theta|\bot|\Delta\rangle & \longrightarrow_r & \bot &
\end{array}$$

Figure 7: Partial substitution rules

3.2 An Extension: Partial Substitution

Relaxing the matching procedure generates new possibilities of evaluation, which may bring more partial evaluation, more sharing or more parallelism. We explore here an extension of PPC_{EM} where the partial result of a matching can be applied to the function body before the matching process is completed.

Example 5. *Consider the following reduction:*

$$
\begin{array}{ll}
 & ([x]\hat{x}z \to (([\emptyset]x \to b)\hat{c}))\,(\hat{c}t) \\
\longrightarrow_B & ([x]\hat{x}z \to (b\,\langle\emptyset|\emptyset|(\hat{c},x)\rangle))\,(\hat{c}t)
\end{array}
$$

The matching $\langle\emptyset|\emptyset|(\hat{c},x)\rangle$ is blocked because of the presence of the variable x in the pattern. Still, the external application can be evaluated:

$$
\begin{array}{ll}
\longrightarrow_B & (b\,\langle\emptyset|\emptyset|(\hat{c},x)\rangle)\,\langle x|\emptyset|(\hat{c}t,\hat{x}z)\rangle \\
\longrightarrow_\bullet^2 & (b\,\langle\emptyset|\emptyset|(\hat{c},x)\rangle)\,\langle x|\emptyset|(\hat{c}\bullet t,\hat{x}\bullet z)\rangle \\
\longrightarrow_m & (b\,\langle\emptyset|\emptyset|(\hat{c},x)\rangle)\,\langle x|\emptyset|(\hat{c},\hat{x})(t,z)\rangle \\
\longrightarrow_m & (b\,\langle\emptyset|\emptyset|(\hat{c},x)\rangle)\,\langle x|\{x\mapsto\hat{c}\}|(t,z)\rangle
\end{array}
$$

Now, the external matching $\langle x|\{x\mapsto\hat{c}\}|(t,z)\rangle$ is also blocked because of the variable z. However, its partial result is a substitution for x which, if applied, may unlock the internal matching. Indeed, allowing this partial substitution could lead to a reduction like:

$$
\begin{array}{ll}
\longrightarrow & (b\,\langle\emptyset|\emptyset|(\hat{c},\hat{c})\rangle)\,\langle x|\{x\mapsto\hat{c}\}|(t,z)\rangle \\
\longrightarrow_m & (b\,\langle\emptyset|\emptyset|\emptyset\rangle)\,\langle x|\{x\mapsto\hat{c}\}|(t,z)\rangle \\
\longrightarrow_r & b\,\langle x|\{x\mapsto\hat{c}\}|(t,z)\rangle
\end{array}
$$

where the internal matching is finally solved!

This kind of power may be of interest in two situations:

- By allowing more reduction in open terms, we gain more partial evaluation capabilities. This may be interesting for greater sharing and efficient evaluation [HG91].
- Suppose now that z is replaced in the example by a possibly big term. In a parallel implementation we could complete the external matching and evaluate the internal one in parallel. As pointed out in [FMS06], this might represent another gain in efficiency.

A light variation on PPC_{EM} gives this new power to our formalism. The principle of this variant is to systematically apply partial results (substitutions) as soon as they are obtained. Hence they do not need to be remembered in the object representing ongoing matching operations. Only a list of used variables is remembered for linearity verification.

The object representing a matching is now $\langle\theta|\tau|\Delta\rangle$ where τ is either $\perp$ or the list of the names of the matching variables that have already been used. Now the test of disjoint union of substitutions is replaced by a simple test against τ, while the final check compares θ and τ.

Initialisation, structural application, and most matching rules are the same in this variant. The only differences are for the first matching rule and the resolution rules, which are now as in Figure 7.

Any PPC_{EM} term can be translated into a term of this new calculus by applying the following transformation: turn any $b\,\langle\theta|\sigma|\Delta\rangle$ into $b^\sigma\,\langle\theta|dom(\sigma)|\Delta\rangle$ (there is nothing to change in a failed matching).

The simulation between PPC_{EM} and this extension is only one way: any reduction of PPC_{EM} is mapped by the previous morphism to a reduction sequence, but the converse is not true. Indeed the calculus with partial substitution allows new reductions, as pictured in Example 5. Confluence for this variant seems to be provable using the same technique as for plain PPC_{EM}.

Conclusion

The *Pure Pattern Calculus* is a compact framework modelling pattern matching with dynamic patterns. However, the conciseness of *PPC* is due to the use of several meta-level notions which deepens the gap between the calculus and implementation-related problems. This contribution defines the *Pure Pattern Calculus with Explicit Matching*, a refinement which is confluent and simulates *PPC*, and allows reasoning on the pattern matching mechanisms.

This enables a very simple definition of new reduction strategies in the spirit of call-by-name, which is new in this kind of framework since the reduction of the argument of a function depends on the pattern of the function, pattern which is itself a dynamic object. In the same direction, it would be interesting to express standardisation in pattern calculi (as presented for example in [KLR10]) using explicit matching.

References

[CK04] S. Cerrito and D. Kesner: Pattern Matching as Cut Elimination. *TCS*, 323:71–127, 2004. doi:10.1016/j.tcs.2004.03.032.

[CFK04] H. Cirstea, G. Faure and C. Kirchner: A Rho-Calculus of Explicit Constraint Application. *5th Workshop on Rewriting Logic and Applications*. *ENTCS*, vol. 117, 51–67, 2005. doi:10.1016/j.entcs.2004.06.029.

[FMS06] M. Fernández, I. Mackie, F.-R. Sinot: Interaction Nets vs the Rho-Calculus: Introducing Bigraphical Nets. *ENTCS*, 154(3):19–32, 2006. doi:10.1016/j.entcs.2006.05.004.

[For02] J. Forest: A Weak Calculus with Explicit Operators for Pattern Matching and Substitution. *13th International Conference on Rewriting Techniques and Applications*. *LNCS*, 2378:174–191, 2002. doi:10.1007/3-540-45610-4_13.

[HG91] C. K. Holst and D. K. Gomard: Partial Evaluation is Fuller Laziness. *PEPM'91*, 223–233, 1991. doi:10.1145/115866.115890.

[Jay09] B. Jay. *Pattern Calculus: Computing with Functions and Data Structures*. Springer, 2009.

[JK09] B. Jay and D. Kesner: First-Class Patterns. *J. Funct. Programming*, 19(2):191–225, 2009. doi:10.1017/S0956796808007144.

[KLR10] D. Kesner, C. Lombardi and A. Ríos: Standardisation for Constructor Based Pattern Calculi. *5th International Workshop on Higher-Order Rewriting: HOR 2010.*

[Klo80] J. W. Klop: *Combinatory Reduction Systems. Ph.D. Thesis, Mathematisch Centrum, Amstermdam, 1980*

[KvOdV08] J. W. Klop, V. van Oostrom, and R. de Vrijer: Lambda Calculus with Patterns. *TCS*, 398:16–31, 2008. doi:10.1016/j.tcs.2008.01.019.

[Ter03] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

[vOo90] V. van Oostrom. Lambda Calculus with Patterns. Technical Report IR228, Vrije Universiteit, Amsterdam, 1990.