Pattern-Based Calculi with Finitary Matching

Sandra Alves[a], Besik Dundua[b], Mário Florido[c], Temur Kutsia[d]

[a] *CRACS/INESCTEC, Faculty of Sciences, University of Porto, Portugal*
[b] *VIAM, Ivane Javakhishvili Tbilisi State University, Georgia*
[c] *Dep. of Computer Science, Faculty of Sciences & LIACC, University of Porto, Portugal*
[d] *RISC, Johannes Kepler University Linz, Austria*

Abstract

Finitary matching problems are those that have finitely many solutions. Pattern calculi generalize the lambda-calculus, replacing the abstraction over variables by an abstraction over terms that are called patterns. Consequently, reduction requires solving a pattern matching problem. The framework described in this paper considers the case when such problems are finitary. It is parametrized by the solving function, which is responsible for computing solutions to the matching problems. A concrete instance of the function gives a concrete version of the pattern calculus. We impose conditions on the solving function, obtaining a generic confluence proof for a class of pattern calculi with finitary matching. Instances of the solving function are presented.

Keywords: Lambda calculus, pattern calculus, finitary matching, confluence, functional programming.

1. Introduction

Pattern matching plays a central role in functional programming languages such as, e.g., Haskell and the ML family. It is intensively used in proof assistants as well (Agda, Coq, HOL, Isabelle, Theorema, etc.). More recently, it has been adopted in multi paradigm languages with a broad use (see the example of Scala, for instance). New pattern matching algorithms have been applied in several situations (XHaskell, Mathematica, XCentric, constraint logic programming languages).

The main formal models for pattern matching-based programming languages are known by the general name of pattern calculi. They extend the λ-calculus with pattern matching capabilities, replacing the abstraction over variables by an abstraction over terms that are called patterns. For example, $\lambda(f\,x\,y).\,(g\,x)$ is a valid term of a pattern calculus, where f and g are function symbols and the pattern is $(f\,x\,y)$. Intuitively, it stands for a function whose argument should have the form of $(f\,x\,y)$. The application $(\lambda(f\,x\,y).\,(g\,x))\,Q$ can be reduced if $(f\,x\,y)$ matches Q: for instance, if $Q = (f\,a\,b)$, then the reduction gives $(g\,a)$. In some pattern calculi, not all variables in the pattern are allowed to be replaced by matching. In such cases, the matchable ones are explicitly indicated. For example, in $\lambda_{\{x\}}(f\,x\,y).\,(g\,x)$, only x can be matched. Then this term can not be applied to $(f\,a\,b)$, but can be applied to, e.g., $(f\,a\,y)$. Variations on the permitted patterns, operations over them, and of the used matching algorithms gave rise to various versions of pattern calculi, see, e.g., the λ-calculus with first-order constructor patterns (Peyton Jones, 1987), the λ-calculus with patterns (van Oostrom, 1990; Klop et al., 2008), ρ-calculus (Cirstea and Kirchner, 2001), or the pure pattern calculus (Jay and Kesner, 2006).

Although pattern calculi are expressive, there is a price to pay for that: confluence of reduction is lost and various restrictions have to be imposed to recover it. Some restrictions concern the form of patterns, some others permit arbitrary patterns but restrict matching, etc. Cirstea and Faure (2007) proposed an elegant generic approach to proving confluence, which has been applied to various calculi, including the λ-calculus with patterns (van Oostrom, 1990), ρ-calculus (Cirstea and Kirchner, 2001), and a simplified version of the pure pattern calculus (Jay and Kesner, 2006).

Jay and Kesner (2009) went further in developing such a general framework. Their idea was to allow each symbol x to appear as either a variable symbol x or a matchable symbol $\hat{x}$. It helps to free the reduction relation from using a context that would be needed to keep track of binding symbols and distinguish them from the non-binding ones, i.e., it helps the reduction relation to become context-free. Variables can be instantiated by substitutions. Matchables are used to match. Patterns are first-class citizens: They can be passed as parameters, evaluated, and returned as results. The framework is parametrized by the matching function, as in (Cirstea and Faure, 2007), and gives a generic confluence proof for various expressive pattern calculi, including the original pure pattern calculus (Jay and Kesner, 2006) and all other pattern calculi in which successful matching is limited to closed patterns (these are patterns that are protected from substitution by their enclosing binder).

The approaches mentioned above consider unitary matching: each solvable pattern matching problem has a unique solution. A different setting is the one of finitary matching, e.g., when matching is performed modulo some equational theory, which leads to finitely many matchers. For instance, if f is commutative, the pattern $(f\, x\, y)$ matches $(f\, a\, b)$ in two different ways with the substitutions $\{x \mapsto a, y \mapsto b\}$ and $\{x \mapsto b, y \mapsto a\}$. Hence, applying a function with a pattern-abstraction may return different results: $(\lambda_{\{x,y\}}(f\, x\, y).(g\, x)).(f\, a\, b)$ can give either $(g\, a)$ or $(g\, b)$.

Some well-known examples of finitary matching theories include matching modulo associativity (A), commutativity (C), associativity with the unit element (AU), associativity and commutativity (AC), AC with the unit element (ACU), second-order matching, string, hedge, context matching, etc. There are programming languages and tools that use finitary matching in various forms. For instance, A, C, and AC matchings are used in Maude (Clavel et al., 2007), Elan (Borovanský et al., 1996), and OBJ (Goguen et al., 2000; Diaconescu and Futatsugi, 1998); A and AU matchings in ASF-SDF (van den Brand et al., 2001) and Tom (Balland et al., 2007); A, C, AC, and hedge matchings in Wolfram (the programming language of the symbolic computation system Mathematica (Wolfram, 2003)); restricted higher-order matching in the program transformation language MAG (de Moor and Sittampalam, 2001); hedge matching in the hedge transformation tool PρLog (Dundua et al., 2009, 2017), in the XML processing language XCentric (Coelho and Florido, 2007), in the CLP language for hedges CLP(H) (Dundua et al., 2016), in various XML query languages, etc.

Computing finitely many matchers makes the resulting calculus non-deterministic, but rather than having a purely non-deterministic evaluation, we define a reduction that introduces a formal sum of terms, representing all possible results of a non-deterministic computation. For instance, we reduce $(\lambda_{\{x,y\}}(f\, x\, y).(g\, x)).(f\, a\, b)$ to $(g\, a) + (g\, b)$, if f is commutative. The idea of transforming a non-deterministic calculus into a deterministic one with such a technique is not new. It has been exploited in a way or another in, e.g., the differential λ-calculus (Ehrhard and Regnier, 2003), the linear-algebraic λ-calculus (Arrighi and Dowek, 2008; Díaz-Caro and Petit, 2012), the resource calculus (Pagani and Ronchi Della Rocca, 2010), etc. In our calculus, sums originate from sets of different matchers. Hence, besides being associative and commutative, $+$ is also idempotent. Moreover, abstraction distributes over sum from the left, and application distributes from the right.

Extending confluence results from unitary to finitary matching is syntactically and semantically a non-trivial problem and opens new challenges, as underlined in (Cirstea and Faure, 2007). This is the problem we address in this paper. Our approach is based on the one from (Jay and Kesner, 2009), permitting arbitrary terms in patters, syntactically distinguishing between variables and matchables, and making the reduction relation context-free.

When trying to reduce a term $(\lambda_\chi P.N)Q$, the following alternatives exist (cf. those from (Jay and Kesner, 2009)): First, the pattern P may match Q in multiple ways, returning finitely many matchers $\sigma_1, \ldots, \sigma_n$. In this case the reduct is the sum of instances of N under these substitutions: $N\sigma_1 + \cdots + N\sigma_n$. Second, the attempt of matching P to Q fails. In the case the reduct is the identity function. Such a reduction can help to model conditionals and pattern patching functions, see (Jay and Kesner, 2009). Third, matching between P and Q is not defined. In this case reduction is not made. One should wait, hoping that some reductions and instantiations may change P or Q so that matching between them succeeds or fails.

We use the matching function as a parameter and formulate conditions on it, which we then prove are sufficient for confluence. They ensure that no fresh free variable appears during reduction, and that reductions are stable both by substitution and by parallel reduction. Hence, in this way, we obtain a generic confluence proof for pattern calculi with finitary matching, which satisfy the formulated conditions. Our results generalize those of (Jay and Kesner, 2009): If our conditions are restricted to unitary matching, they imply the Rigid Matching Condition of Jay and Kesner, which is sufficient for confluence for the unitary case. We also give concrete examples of the matching function that satisfy the sufficient conditions.

Overview. The rest of the paper is organized as follows: In Section 2 we present our finitary pattern calculus and a general notion of reduction parametrized by a pattern matching function. In Section 3 we define conditions on the pattern matching function and prove confluence of the calculus when the matching function satisfies the conditions. In Section 4 we provide specific instances of the pattern matching function that fulfill the confluence conditions. Concluding remarks are given in Section 5.

2. Finitary Pattern Calculus

In this section we define the syntax and the operational semantics of the finitary pattern calculus.

2.1. Syntax

The alphabet $\mathcal{A}$ of the pattern calculus consists of the sets $\mathcal{V}$ of variables and $\mathcal{F}$ of function symbols. They are disjoint and countably infinite. The set $\mathcal{F}$ includes special constants $+$ and $\hat{}$.

Definition 2.1. *Terms* are defined by the following grammar:

$$\begin{array}{lll} M,N ::= & x & \text{(variable)} \\ & \mid \hat{x} & \text{(matchable)} \\ & \mid f & \text{(function symbol from } \mathcal{F}\setminus\{+,\hat{}\}) \\ & \mid M\,N & \text{(application)} \\ & \mid \lambda_\chi M.N & \text{(abstraction)} \\ & \mid M+N & \text{(sum)} \end{array}$$

In the abstraction $\lambda_\chi M.N$, the term M is called a *pattern*, and χ is a finite set of variables, called *binding variables*.

The set of binding variables χ in $\lambda_\chi M.N$ helps to specify which variables and matchables are bound by the abstraction. We define this notion formally a bit later. The variables are used for substitution, while matchables are used for matching.

We will use x, y, z, v for variables, $\hat{x}, \hat{y}, \hat{z}, \hat{v}$ for matchables, f, g, h, a, b, c for function symbols from $\mathcal{F}\setminus\{+,\hat{}\}$, and M, N, P, Q, W for terms, where P is usually reserved for patterns.

As usual, applications associate to the left and abstractions to the right. Application binds stronger than abstraction and $+$. Abstraction binds stronger than $+$. When there is no ambiguity, parentheses are omitted.

Example 2.2. Due to the conventions above, we write

- $((f\,\hat{x})\,a)\,x$ as $f\hat{x}ax$,
- $\lambda_{\{x\}}\,\hat{x}.\,(g\,x)$ as $\lambda_{\{x\}}\,\hat{x}.\,gx$,
- $\lambda_{\{x,y\}}\,x.((g\,x)+y)$ as $\lambda_{\{x,y\}}\,x.\,(gx+y)$,
- $(\lambda_{\{x,z\}}((f\,\hat{x})\,\hat{y}).\,(\hat{x}\,x))\,z$ as $(\lambda_{\{x,z\}}f\hat{x}\hat{y}.\,\hat{x}x)\,z$,
- $\lambda_{\{x,z\}}(((f\,\hat{x})+(g\,\hat{x}))\,\hat{z}).\,((f\,x)\,((g\,y)+(g\,z)+(g\,y)))$ as $\lambda_{\{x,z\}}(f\hat{x}+g\hat{x})\hat{z}.\,fx\,(gy+gz+gy)$.

Notation: Given a set of variable χ, the set $\hat{\chi}$ is defined as $\hat{\chi} := \{\hat{x} \mid x \in \chi\}$.

Definition 2.3. The *sets of free variables, free matchables, bound variables,* and *bound matchables* of a term Q, denoted by $\mathtt{fv}(Q)$, $\mathtt{fm}(Q)$, $\mathtt{bv}(Q)$, and $\mathtt{bm}(Q)$ respectively, are defined inductively as follows:

$$\begin{array}{ll}
\mathtt{fv}(x) = \{x\} \quad \mathtt{fv}(f) = \emptyset & \mathtt{fm}(x) = \emptyset \qquad \mathtt{fm}(f) = \emptyset \\
\mathtt{fv}(\hat{x}) = \emptyset & \mathtt{fm}(\hat{x}) = \{\hat{x}\} \\
\mathtt{fv}(M\,N) = \mathtt{fv}(M) \cup \mathtt{fv}(N) & \mathtt{fm}(M\,N) = \mathtt{fm}(M) \cup \mathtt{fm}(N) \\
\mathtt{fv}(\lambda_\chi P.N) = & \mathtt{fm}(\lambda_\chi P.N) = \\
\qquad (\mathtt{fv}(N) \setminus \chi) \cup \mathtt{fv}(P) & \qquad (\mathtt{fm}(P) \setminus \hat{\chi}) \cup \mathtt{fm}(N) \\
\mathtt{fv}(M+N) = \mathtt{fv}(M) \cup \mathtt{fv}(N) & \mathtt{fm}(M+N) = \mathtt{fm}(M) \cup \mathtt{fm}(N)
\end{array}$$

$$\begin{array}{ll}
\mathtt{bv}(x) = \emptyset \quad \mathtt{bv}(f) = \emptyset & \mathtt{bm}(Q) = \{\hat{x} \mid x \in \mathtt{bv}(Q)\} \\
\mathtt{bv}(\hat{x}) = \emptyset & \\
\mathtt{bv}(M\,N) = \mathtt{bv}(M) \cup \mathtt{bv}(N) & \\
\mathtt{bv}(\lambda_\chi P.N) = \mathtt{bv}(P) \cup \chi \cup \mathtt{bv}(N) & \\
\mathtt{bv}(M+N) = \mathtt{bv}(M) \cup \mathtt{bv}(N) &
\end{array}$$

We say that in a term $\lambda_\chi P.N$, for each binding variable $x \in \chi$ the quantifier λ_χ *binds* all free variable occurrences of x in N and all free matchable occurrences of $\hat{x}$ in P.

Example 2.4. Now we illustrate the introduced notions on examples:

- Let $Q = \lambda_{\{x,y\}}\, x.\, gxy$. Then

  $$\mathtt{fv}(Q) = \{x\}, \quad \mathtt{fm}(Q) = \emptyset, \quad \mathtt{bv}(Q) = \{x, y\}, \quad \mathtt{bm}(Q) = \{\hat{x}, \hat{y}\}.$$

  The x in $\mathtt{fv}(Q)$ is taken from the pattern x, while $x \in \mathtt{bv}(Q)$ because it belongs to the set of binding variables. Note also that $\mathtt{bm}(Q) = \{\hat{x}, \hat{y}\}$, although neither $\hat{x}$ nor $\hat{y}$ appear in the term explicitly.

- Let $Q = (\lambda_{\{x,z\}} f\hat{x}\hat{y}.\, \hat{x}x)\, z$. Then

  $$\mathtt{fv}(Q) = \{z\}, \quad \mathtt{fm}(Q) = \{\hat{x}, \hat{y}\},$$
  $$\mathtt{bv}(Q) = \{x, z\}, \quad \mathtt{bm}(Q) = \{\hat{x}, \hat{z}\}.$$

Here $\hat{x} \in \texttt{fm}(Q)$ because it appears freely in the subterm $(\hat{x}\,x)$, $\hat{y} \in \texttt{fm}(Q)$ because it appears freely in the pattern $f\hat{x}\hat{y}$, and $\texttt{bv}(Q) = \{x, z\}$ and $\texttt{bm}(Q) = \{\hat{x}, \hat{z}\}$ because $\{x, z\}$ is the set of all binding variables that appear in Q. Note that $\hat{z}$ does not appear in the term explicitly, z appears both in $\texttt{fv}(Q)$ and $\texttt{bv}(Q)$, and $\hat{x}$ appears both in $\texttt{fm}(Q)$ and $\texttt{bm}(Q)$.

Definition 2.5. A partial operation of *renaming of a variable* x by a variable y is defined as follows:

$$\begin{array}{ll}
x[x := y] = y & \\
z[x := y] = z & \text{if } z \neq x \\
\hat{z}[x := y] = \hat{z} & \\
(M\,N)[x := y] = M[x := y]N[x := y] & \\
(\lambda_\chi P.N)[x := y] = \lambda_\chi P[x := y].N[x := y] & \text{if } x, y \notin \chi \\
(M + N)[x := y] = M[x := y] + N[x := y] &
\end{array}$$

A partial operation of *renaming of a matchable* $\hat{x}$ by a matchable $\hat{y}$ is defined as follows:

$$\begin{array}{ll}
z[\hat{x} := \hat{y}] = z & \\
\hat{x}[\hat{x} := \hat{y}] = \hat{y} & \\
\hat{z}[\hat{x} := \hat{y}] = \hat{z} & \text{if } z \neq x \\
(M\,N)[\hat{x} := \hat{y}] = M[\hat{x} := \hat{y}]N[\hat{x} := \hat{y}] & \\
(\lambda_\chi P.N)[\hat{x} := \hat{y}] = \lambda_\chi P[\hat{x} := \hat{y}].N[\hat{x} := \hat{y}] & \text{if } x, y \notin \chi \\
(M + N)[\hat{x} := \hat{y}] = M[\hat{x} := \hat{y}] + N[\hat{x} := \hat{y}] &
\end{array}$$

Next, we introduce axioms that define certain properties of terms. The first one is an axiom schema which helps to identify terms that differ only in the names of bound variables and matchables:

$$(\alpha) \qquad (\lambda_\chi P.N) = \lambda_{(\chi \setminus \{x\}) \cup \{y\}} P[\hat{x} := \hat{y}].N[x := y],$$

where $x \in \chi$, $y \notin \chi \cup \texttt{fv}(N)$, and $\hat{y} \notin \texttt{fm}(P)$.

The other axioms characterize the function symbol $+$, stating its associativity, commutativity, and idempotence:

$$\begin{array}{ll}
\text{(A)} & (M_1 + M_2) + M_3 = M_1 + (M_2 + M_3). \\
\text{(C)} & M_1 + M_2 = M_2 + M_1. \\
\text{(I)} & M + M = M.
\end{array}$$

By $\approx$, we denote the least congruence relation on terms generated by the axioms (α), (A), (C), and (I).

Example 2.6. Here we give a couple of examples of $\approx$-equal terms:

- $\lambda_{\{x\}}(\hat{x}+\hat{x}).(a+(x+z)) \approx \lambda_{\{y\}}\hat{y}.((z+y)+(a+z))$.
- $\lambda_{\{x\}}\hat{x}.((x+a)+x)+\lambda_{\{y\}}\hat{y}.(a+y) \approx \lambda_{\{z\}}\hat{z}.(z+a)$.

An equational theory is given by a set of equational axioms. For an equational theory $\mathcal{E}$, we write $M \approx_{\mathcal{E}} N$ if M and N are equal modulo $\approx$ and $\mathcal{E}$. For instance, if $\mathcal{E}$ asserts commutativity of f, we have $\lambda_{\{x,y\}}f\hat{x}a\hat{y}.fxy + fab + \lambda_{\{x,z\}}f\hat{z}\hat{x}a.fxz \approx_{\mathcal{E}} fba + \lambda_{\{y,z\}}fa\hat{y}\hat{z}.fyz$, because

- $fab \approx_{\mathcal{E}} fba$ due to commutativity of f,
- $\lambda_{\{x,y\}}f\hat{x}a\hat{y}.fxy \approx_{\mathcal{E}} \lambda_{\{x,z\}}f\hat{z}\hat{x}a.fxz \approx_{\mathcal{E}} \lambda_{\{y,z\}}fa\hat{y}\hat{z}.fyz$ due to α-renaming and commutativity of f,
- $\lambda_{\{y,z\}}fa\hat{y}\hat{z}.fyz + fba + \lambda_{\{y,z\}}fa\hat{y}\hat{z}.fyz \approx_{\mathcal{E}} fba + \lambda_{\{y,z\}}fa\hat{y}\hat{z}.fyz$ due to associativity, commutativity, and idempotence of $+$.

We adopt Barendregt's *variable name convention* (Barendregt, 1984), i.e., free variables are distinct from bound variables and free matchables differ from bound ones. It can be proved that every term is equivalent modulo α (and, hence, equivalent modulo $\approx_{\mathcal{E}}$) to such a term.

Usually, we do not distinguish between terms that are equal modulo $\approx_{\mathcal{E}}$.

Definition 2.7. A *substitution* is a mapping from variables to terms such that all but finitely many variables are mapped to themselves.

Substitutions will be denoted by Greek letters σ, ϑ, ρ, and φ. The identity substitution is represented by Id. We will use the set notation, writing $\{x \mapsto \sigma(x) \mid x \not\approx_{\mathcal{E}} \sigma(x)\}$ for a substitution σ.

The sets $dom(\sigma) = \{x \mid x \not\approx_{\mathcal{E}} \sigma(x)\}$ and $ran(\sigma) = \{\sigma(x) \mid x \in dom(\sigma)\}$ are called the *domain* and the *range* of σ, respectively. The sets of free variables, free matchables, and variables of a substitution σ are defined respectively as $\mathtt{fv}(\sigma) = \mathtt{fv}(ran(\sigma))$, $\mathtt{fm}(\sigma) = \mathtt{fm}(ran(\sigma))$, and $var(\sigma) = dom(\sigma) \cup \mathtt{fv}(\sigma)$.

The $\approx_{\mathcal{E}}$ relation extends to substitutions in a straightforward way: $\sigma \approx_{\mathcal{E}} \vartheta$ iff $x\sigma \approx_{\mathcal{E}} x\vartheta$ for any x.

Definition 2.8. The *application of a substitution σ to a term M*, denoted $M\sigma$, is defined inductively in the following way:

$$\begin{array}{ll} x\sigma = \sigma(x). & (M\,N)\sigma = M\sigma\,N\sigma. \\ \hat{x}\sigma = \hat{x}. & (M+N)\sigma = M\sigma + N\sigma. \\ f\sigma = f. & (\lambda_\chi P.N)\sigma = \lambda_\chi P\sigma.N\sigma. \end{array}$$

In the substitution application to an abstraction term, it is assumed that $var(\sigma) \cap \chi = \emptyset$ and $\mathtt{fm}(\sigma) \cap \hat{\chi} = \emptyset$. This is achieved by variable convention, which helps to properly rename bound symbols.

The term $M\sigma$ is called the *instance* of M under σ.

The *composition* of substitutions is defined in the standard way, as the composition of two mappings. We use juxtaposition to denote it, writing $\sigma\vartheta$ for the composition of σ and ϑ. For all M, we have $M\sigma\vartheta = (M\sigma)\vartheta$.

The *restriction* of a substitution σ to a set of variables X, denoted $\sigma|_X$, is defined as $\sigma|_X(x) = \sigma(x)$ if $x \in X$, and $\sigma|_X(x) = x$ otherwise.

Definition 2.9. Let M be a term and Θ be a finite set of substitutions. Application of Θ to M, denoted $M[\Theta]$, is defined as follows:

- If $\Theta = \{\sigma_1, \dots, \sigma_n\}$, $n > 0$, then $M[\Theta] = M\sigma_1 + \cdots + M\sigma_n$.
- If $\Theta = \emptyset$, then $M[\Theta] = \lambda_{\{x\}}\hat{x}.x$.

Definition 2.10. The *hatted application of a substitution σ to a term M*, denoted $M\hat{\sigma}$, is defined inductively in the following way:

$$\begin{array}{ll} \hat{x}\hat{\sigma} = \sigma(x). & (M\,N)\hat{\sigma} = M\hat{\sigma}\,N\hat{\sigma}. \\ x\hat{\sigma} = x. & (M+N)\hat{\sigma} = M\hat{\sigma} + N\hat{\sigma}. \\ f\hat{\sigma} = f. & (\lambda_\chi P.N)\hat{\sigma} = \lambda_\chi P\hat{\sigma}.N\hat{\sigma}. \end{array}$$

As in Definition 2.8, here also we assume that $var(\sigma) \cap \chi = \emptyset$ and $\mathtt{fm}(\sigma) \cap \hat{\chi} = \emptyset$ in the hatted application of a substitution to an abstraction term. $M\hat{\sigma}$ is called the *hatted instance* of M under σ.

An *equational matching equation* is a quadruple of a pattern term P, a data term Q, a finite set of variables χ, and an equational theory $\mathcal{E}$, written as $P \ll_\chi^\mathcal{E} Q$. Its *solution* (a *matcher*) is a substitution σ such that $dom(\sigma) = \chi$ and $P\hat{\sigma} \approx_\mathcal{E} Q$.

2.2. Operational semantics

The operational semantics of the finitary pattern calculus is given by a set of reduction rules.

Definition 2.11. We define *reduction* (or *evaluation*) in the pattern calculus by the following binary relations β_{p}, λ_{ld}, and A_{rd} on terms, written in the form of reduction rules:

$$\begin{aligned}
\beta_{\mathsf{p}} :&\quad (\lambda_\chi P.N)\, Q \to N[\mathsf{solve}(P \ll^{\mathcal{E}}_{\chi} Q)].\\
\lambda_{\mathsf{ld}} :&\quad \lambda_\chi P.(N_1 + N_2) \to \lambda_\chi P.N_1 + \lambda_\chi P.N_2.\\
\mathsf{A}_{\mathsf{rd}} :&\quad (M_1 + M_2)N \to M_1 N + M_2 N.
\end{aligned}$$

The relation β_{p} defines the way how pattern-abstractions are applied. It is parametrized by a *pattern matching function* solve, which takes as input a pair of terms, a finite set of variables, and an equational theory $\mathcal{E}$ (the input written above in the form of an $\mathcal{E}$-matching equation $P \ll^{\mathcal{E}}_{\chi} Q$) and returns a finite set of substitutions that are solutions of $P \ll^{\mathcal{E}}_{\chi} Q$. Note that solve is a partial function, i.e., for some P, Q, χ, and $\mathcal{E}$ it might not be defined. In that case β_{p} does not apply. The relation λ_{ld} defines left-distributivity of abstraction over $+$ and the relation A_{rd} defines right-distributivity of application over $+$.

The term *finitary* pattern calculus is related to the fact that solve in the definition above returns a *finite* set of substitutions.

A property of solve that we assume in this paper is that it preserves $\approx_{\mathcal{E}}$-equivalence, producing equivalent results for equivalent inputs. This property can be defined more precisely: For any terms P_0, Q_0, P_1, Q_1, a finite set of variables χ, and an equational theory $\mathcal{E}$, let $(\lambda_\chi P_0.N)Q_0 \approx_{\mathcal{E}} (\lambda_\chi P_1.N)Q_1$ for an arbitrary N. Then either both $\mathsf{solve}(P_0 \ll^{\mathcal{E}}_{\chi} Q_0)$ and $\mathsf{solve}(P_1 \ll^{\mathcal{E}}_{\chi} Q_1)$ are undefined, or solve satisfies $\mathsf{solve}(P_0 \ll^{\mathcal{E}}_{\chi} Q_0) \approx_{\mathcal{E}} \mathsf{solve}(P_1 \ll^{\mathcal{E}}_{\chi} Q_1)$, where the $\approx_{\mathcal{E}}$ relation between two sets S_0 and S_1 is understood as follows: (a) $\emptyset \approx_{\mathcal{E}} \emptyset$; (b) for each $\sigma_0 \in S_0$ there exists $\sigma_1 \in S_1$ such that $\sigma_0 \approx_{\mathcal{E}} \sigma_1$ and vice versa: for each $\sigma_1 \in S_1$ there exists $\sigma_0 \in S_0$ such that $\sigma_0 \approx_{\mathcal{E}} \sigma_1$.

A reducible expression, or *redex,* is any expression to which β_{p}, λ_{ld}, or A_{rd} can be applied.

Definition 2.12. A binary relation $\to_R$ on terms is *compatible* if it is defined by the following inference rules below:

$$\frac{M \to_R M'}{MN \to_R M'N} \qquad \frac{M \to_R M'}{NM \to_R NM'} \qquad \frac{M \to_R M'}{M + N \to_R M' + N}$$

$$\frac{P \rightharpoonup_R P'}{\lambda_\chi P.N \rightharpoonup_R \lambda_\chi P'.N} \qquad \frac{N \rightharpoonup_R N'}{\lambda_\chi P.N \rightharpoonup_R \lambda_\chi P.N'}$$

Because of commutativity of $+$, one rule for the sum is sufficient.

In what follows, $\twoheadrightarrow$ denotes the compatible closure of the union of the relations β_{p}, λ_{ld} and A_{rd}. The relation $\twoheadrightarrow^*$ denotes the reflexive and transitive closure of $\twoheadrightarrow$.

Definition 2.13. We say that $\twoheadrightarrow$ is *confluent*, if for all terms M, N, and Q, $M \twoheadrightarrow^* N$ and $M \twoheadrightarrow^* Q$ imply that there exists a term W such that $N \twoheadrightarrow^* W$ and $Q \twoheadrightarrow^* W$. Confluence of a calculus means confluence of the relation $\twoheadrightarrow$ for that calculus.

Remark 2.14. Confluence above is, actually, confluence modulo $\approx_{\mathcal{E}}$, which means that for all terms M_1, M_2, N, and Q, if $M_1 \approx_{\mathcal{E}} M_2$, $M_1 \twoheadrightarrow^* N$ and $M_2 \twoheadrightarrow^* Q$, then there exist terms W_1 and W_2 such that $N \twoheadrightarrow^* W_1$, $Q \twoheadrightarrow^* W_2$, and $W_1 \approx_{\mathcal{E}} W_2$. But since we identify $\approx_{\mathcal{E}}$-equal terms, we could formulate it in a simpler way as above, with terms actually standing for $\approx_{\mathcal{E}}$-equivalence classes.

Remark 2.15. In Definition 2.1, we did not impose restrictions on the set of binding variables χ in $\lambda_\chi P.N$. This differs from the definition of abstraction terms in (Cirstea and Faure, 2007), where χ is supposed to be a subset of the set of free variables of P. Note that matchables and variables are not distinguished in (Cirstea and Faure, 2007). This restriction has a consequence that the set of terms is not closed under reduction. Had we imposed the analogous condition, which in our case would be $\hat{\chi} \subseteq \mathtt{fm}(P)$, we would also reduce some terms to expressions that are not terms. For instance, consider a term $\lambda_{\{x\}}((\lambda_{\{y\}}\, \hat{y}.\, z)\, \hat{x}).M$. If we assume that the β_{p} rule is applicable by the substitution $\{y \mapsto \hat{x}\}$, we would get $\lambda_{\{x\}} z.M$. However, $\{\hat{x}\} \subseteq \mathtt{fm}(z)$ does not hold, since $\mathtt{fm}(z) = \emptyset$. Hence, $\lambda_{\{x\}} z.M$ would not be a term.

Note that Jay and Kesner (2009) do not consider the restriction $\chi \subseteq \mathtt{fv}(P)$ either.

Our goal is to study confluence of the finitary pattern calculus. It is tempting to leave out χ from $\lambda_\chi P.N$ and assume that the abstraction binds all variables in the pattern. However, it causes a serious problem, since bound variables can be freed and confluence does not hold even for the following simple (unitary) case:

$$\lambda(\lambda\hat{x}.a)\hat{y}.y \twoheadrightarrow \lambda a.y.$$
$$\lambda(\lambda\hat{x}.a)\hat{y}.y \approx \lambda(\lambda\hat{x}.a)\hat{z}.z \twoheadrightarrow \lambda a.z.$$

In comparison, having χ explicit, the term $\lambda_{\{y\}}(\lambda_{\{x\}}\hat{x}.a)\hat{y}.y$ reduces to two $\approx$-equivalent terms:

$$\lambda_{\{y\}}(\lambda_{\{x\}}\hat{x}.a)\hat{y}.y \twoheadrightarrow \lambda_{\{y\}}a.y.$$
$$\lambda_{\{y\}}(\lambda_{\{x\}}\hat{x}.a)\hat{y}.y \approx \lambda_{\{z\}}(\lambda_{\{x\}}\hat{x}.a)\hat{z}.z \twoheadrightarrow \lambda_{\{z\}}a.z \approx \lambda_{\{y\}}a.y.$$

Remark 2.16. Jay and Kesner (2009) proposed explicit distinction between matching success (given by a matching substitution), failure (denoted there by fail), and waiting match (denoted by wait). The reduction in case of successful matching is standard. With fail, every term is reduced to the identity function. We also followed this idea in reduction, via Definition 2.9. As Jay and Kesner (2009) show, such an interpretation of matching failure provides a natural branching mechanism. Waiting matching does not lead to a reduction.

In comparison, we distinguish between successful and failing matches by the result of the solve function: If it is nonempty, the match is successful, otherwise fails. We do not consider waiting matches explicitly. Instead, out solving function is partial. When it is not defined, the β_{p} rule is not fired.

3. Confluence

In this section we show that the reduction relation defined in the previous section is confluent, provided that the function solve satisfies the conditions $\mathbf{C}_1$ and $\mathbf{C}_2$ defined below. These are sufficient conditions: For each solve that satisfies them, the calculus will be confluent. This requires some additional definitions.

The confluence proof will be based on the method due to Tait and Martin-Löf (Barendregt, 1984), using parallel reductions. They were first used by Tait and Martin-Löf to prove the Church-Rosser theorem for β-reduction in the λ-calculus. Intuitively, parallel reductions mean that a term is reduced by simultaneous reduction of some redexes.

After the conditions on solve are formulated, the confluence proof follows as expected: We first show that the transitive closures of both the original and parallel reductions are the same relation (Corollary 3.6 of Lemma 3.5). Then we prove that parallel reduction satisfies the diamond property (Lemma 3.8) and, hence, is confluent, which finally implies that the original reduction is also confluent (Theorem 3.9). Note that proving confluence of parallel reductions is done only by induction. Parallelism avoids to take care of residuals explicitly and to add extra auxiliary syntax to the original calculus.

Now we define the notions and formulate the sufficient conditions.

Definition 3.1. The notion of *parallel reduction*, denoted by $\rightrightarrows$, is inductively defined in the following way:

$$\frac{}{M \rightrightarrows M} \qquad \frac{M \rightrightarrows M' \quad N \rightrightarrows N'}{M\,N \rightrightarrows M'\,N'} \qquad \frac{P \rightrightarrows P' \quad N \rightrightarrows N'}{\lambda_\chi P.N \rightrightarrows \lambda_\chi P'.N'}$$

$$\frac{M \rightrightarrows M' \quad N \rightrightarrows N'}{M + N \rightrightarrows M' + N'} \qquad \frac{P \rightrightarrows P' \quad N_1 \rightrightarrows N_1' \quad N_2 \rightrightarrows N_2'}{\lambda_\chi P.(N_1 + N_2) \rightrightarrows \lambda_\chi P'.N_1' + \lambda_\chi P'.N_2'}$$

$$\frac{M_1 \rightrightarrows M_1' \quad M_2 \rightrightarrows M_2' \quad N \rightrightarrows N'}{(M_1 + M_2)N \rightrightarrows M_1'N' + M_2'N'} \qquad \frac{P \rightrightarrows P' \quad N \rightrightarrows N' \quad Q \rightrightarrows Q'}{(\lambda_\chi P.N)Q \rightrightarrows N'[\mathsf{solve}(P' \ll_\chi^\mathcal{E} Q')]}.$$

The definition is extended to substitutions having the same domain:

Definition 3.2. Let σ, σ' be substitutions such that $dom(\sigma) = dom(\sigma')$. We say that $\sigma \rightrightarrows \sigma'$ if and only if $x\sigma \rightrightarrows x\sigma'$ for all $x \in dom(\sigma)$.

The following proposition easily follows from the definitions of compatible relation and parallel reduction:

Proposition 3.3. $\rightrightarrows$ is a compatible relation.

The notion of parallel reduction on substitutions (having the same domain) can be further extended to the sets of substitutions (having the same domain):

Definition 3.4. Let Θ and Θ' be two sets of substitutions all having the same domain. We say that $\Theta \rightrightarrows \Theta'$ if and only if

(a) for all $\sigma \in \Theta$ there exists $\sigma' \in \Theta'$ such that $\sigma \rightrightarrows \sigma'$ and

(b) for all $\sigma' \in \Theta'$ there exists $\sigma \in \Theta$ such that $\sigma \rightrightarrows \sigma'$.

Note that it is not necessary Θ and Θ' to have the same number of elements. The definition also implies that if $\Theta \rightrightarrows \Theta'$, then $\Theta = \emptyset$ iff $\Theta' = \emptyset$.

Now we define the conditions for solve.

C$_1$**:** If $\mathsf{solve}(P \ll^{\mathcal{E}}_{\chi} Q) = \{\sigma_1, \dots, \sigma_n\}$, $n \geq 0$, then for all ϑ with $\mathtt{fm}(\vartheta) \cap \hat{\chi} = \emptyset$, we have $\mathsf{solve}(P\vartheta \ll^{\mathcal{E}}_{\chi} Q\vartheta) = \{(\sigma_1\vartheta)|_{\chi}, \dots, (\sigma_n\vartheta)|_{\chi}\}$.

C$_2$**:** If $\mathsf{solve}(P \ll^{\mathcal{E}}_{\chi} Q) = \Theta$, $P \rightrightarrows P'$, and $Q \rightrightarrows Q'$, then there exists a finite set of substitutions Θ' such that $\mathsf{solve}(P' \ll^{\mathcal{E}}_{\chi} Q') = \Theta'$ and $\Theta \rightrightarrows \Theta'$.

These conditions correspond stability by substitution and stability by reduction in the finitary matching case. We briefly explain the intuition behind them:

- *Stability by substitution:* When we have an application $(\lambda_{\chi}P.N)Q$, it may happen that $\mathtt{fv}(Q) \cup (\mathtt{fv}(P) \setminus \chi)$ is nonempty. Either we wait until those variables get instantiated and then perform reduction, or we reduce $(\lambda_{\chi}P.N)Q$ and then instantiate the variables that come from the set $\mathtt{fv}(Q) \cup (\mathtt{fv}(P) \setminus \chi)$. The results in both cases should be the same. This is what **C**$_1$ requires.[1] Intuitively, this property can be visualized by the commutative diagram below:

$$\begin{tikzcd}
P \ll^{\mathcal{E}}_{\chi} Q \arrow[r, "\mathsf{solve}"] \arrow[d, "\text{substitute}"'] & \{\sigma_1, \dots, \sigma_n\} \arrow[d, "\text{substitute}"] \\
P\vartheta \ll^{\mathcal{E}}_{\chi} Q\vartheta \arrow[r, "\mathsf{solve}"'] & \{(\sigma_1\vartheta)|_{\chi}, \dots, (\sigma_n\vartheta)|_{\chi}\}
\end{tikzcd}$$

- *Stability by reduction:* When the application term $(\lambda_{\chi}P.N)Q$ is reduced, it is not necessary for P and Q to be in a normal form. One can either reduce the application immediately, or first transform P and Q into P' and Q' and only afterwards try to reduce the application. This subsequent reduction of application should not fail. Even more, each substitution in $\mathsf{solve}(P' \ll^{\mathcal{E}}_{\chi} Q')$ should be derivable from a substitution in $\mathsf{solve}(P \ll^{\mathcal{E}}_{\chi} Q)$, and each substitution in $\mathsf{solve}(P \ll^{\mathcal{E}}_{\chi} Q)$ should be

[1]Note that **C**$_1$ does not require $\mathsf{solve}(P \ll^{\mathcal{E}}_{\chi} Q)$ and $\mathsf{solve}(P\vartheta \ll^{\mathcal{E}}_{\chi} Q\vartheta)$ to have the same number of elements. See, e.g., examples 4.20 and 4.21 below.

reducible to a substitution in $\mathsf{solve}(P' \ll^{\mathcal{E}}_{\chi} Q')$. It is not necessary for these two sets to contain the same number of elements.[2] Stability of reduction is required by $\mathbf{C}_2$. The intuition behind it can be visualized by the commutative diagram, where $n, m \geq 0$:

$$\begin{tikzcd}
P \ll^{\mathcal{E}}_{\chi} Q \arrow[r, "\mathsf{solve}"] \arrow[d, "\text{parallel reduction}"'] & \{\sigma_1, \ldots, \sigma_n\} \arrow[d, "\text{parallel reduction}"] \\
P' \ll^{\mathcal{E}}_{\chi} Q' \arrow[r, "\mathsf{solve}"'] & \{\sigma'_1, \ldots, \sigma'_m\}
\end{tikzcd}$$

The properties P1 and P2 from (Jay and Kesner, 2009)[3] are very similar to $\mathbf{C}_1$ and $\mathbf{C}_2$, respectively. The only difference is that P1 and P2 are formulated for unitary matching, while $\mathbf{C}_1$ and $\mathbf{C}_2$ consider finitary matching. Hence, taking in $\mathbf{C}_1$ and $\mathbf{C}_2$ the number of solutions to be 1, we get restrictions equivalent to P1 and P2, respectively. In a similar way, $\mathbf{C}_1$ and $\mathbf{C}_2$ generalize the hypotheses $\mathbf{H_1}$ and $\mathbf{H_2}$ from (Cirstea and Faure, 2007).

Moreover, like Jay and Kesner (2009), we also require the substitutions that solve computes for $P \ll^{\mathcal{E}}_{\chi} Q$ to have χ as the domain, and match P to Q. In case of Jay and Kesner (2009), matching is done modulo α (which corresponds to the unitary matching case),[4] while in our case it is performed modulo $\approx_{\mathcal{E}}$ (since we have the finitary case).

Now we will show that $\mathbf{C}_1$ and $\mathbf{C}_2$ are sufficient for proving confluence of our calculus. We assume that the relations $\multimap$ and $\rightrightarrows$ in the lemmas and in the theorem below use a solve that satisfies $\mathbf{C}_1$ and $\mathbf{C}_2$. The equational theory is $\mathcal{E}$. The notation $\Sigma_{i=1}^{n} M_i$ abbreviates $M_1 + \cdots + M_n$.

Lemma 3.5. *The following inclusions hold:* $\multimap \subseteq \rightrightarrows \subseteq \multimap^*$.

Proof. First we prove $\multimap \subseteq \rightrightarrows$.

[2]See, e.g., examples 4.20 and 4.21 below.

[3]These two properties imply the Rigid Matching Condition, which is sufficient for confluence of the context-free reduction relation from (Jay and Kesner, 2009) for unary matching.

[4]This matchability requirement of Jay and Kesner (2009) implies the hypothesis $\mathbf{H_0}$ of Cirstea and Faure (2007).

When the reduction occurs at the top position, the inclusion follows from the definition of $\rightrightarrows$. Indeed, assume $M = (\lambda_\chi M_1.M_2)M_3$ reduces by β_{p} to $N = M_2[\mathsf{solve}(M_1 \ll^{\mathcal{E}}_{\chi} M_3)]$. Then $M \rightrightarrows N$ follows from the definition of $\rightrightarrows$, since by reflexivity we have $M_1 \rightrightarrows M_1$, $M_2 \rightrightarrows M_2$, and $M_3 \rightrightarrows M_3$. If $M = \lambda_\chi M_1.(M_2 + M_3)$ reduces to $N = \lambda_\chi M_1.M_2 + \lambda_\chi M_1.M_3$ by λ_{ld} or if $M = (N_1 + N_2)M$ reduces to $N = N_1 M + N_2 M$ by A_{rd}, then again by the same reasoning with reflexivity of $\rightrightarrows$ we get $M \rightrightarrows N$. If the reduction does not occur at the top position, the inclusion follows from compatibility of $\rightrightarrows$ (Proposition 3.3).

Now we prove $\rightrightarrows \subseteq \rightarrowtriangle^*$, that is we show that $W_1 \rightrightarrows W_2$ implies $W_1 \rightarrowtriangle^* W_2$ by induction on the derivation length of $W_1 \rightrightarrows W_2$.

- Let $W_1 = M \rightrightarrows W_2 = M$. Then the result follows from reflexivity of $\rightarrowtriangle^*$.

- Let $W_1 = M_1 M_2 \rightrightarrows W_2 = N_1 N_2$ with $M_1 \rightrightarrows N_1$ and $M_2 \rightrightarrows N_2$. Then by the induction hypothesis (IH) we have $M_1 \rightarrowtriangle^* N_1$ and $M_2 \rightarrowtriangle^* N_2$. By compatibility of $\rightarrowtriangle^*$ we also have $M_1 M_2 \rightarrowtriangle^* N_1 M_2$ and $N_1 M_2 \rightarrowtriangle^* N_1 N_2$. By transitivity of $\rightarrowtriangle^*$ we finally get $M_1 M_2 \rightarrowtriangle^* N_1 N_2$.

- Let $W_1 = (\lambda_\chi M_1.M_2)M_3 \rightrightarrows W_2 = N_2[\mathsf{solve}(N_1 \ll^{\mathcal{E}}_{\chi} N_3)]$ with $M_1 \rightrightarrows N_1$, $M_2 \rightrightarrows N_2$, and $M_3 \rightrightarrows N_3$. By the IH, we have $M_1 \rightarrowtriangle^* N_1$, $M_2 \rightarrowtriangle^* N_2$, $M_3 \rightarrowtriangle^* N_3$. Compatibility and transitivity of $\rightarrowtriangle^*$ give $\lambda_\chi M_1.M_2 \rightarrowtriangle^* \lambda_\chi N_1.N_2$ and, hence, $(\lambda_\chi M_1.M_2)M_3 \rightarrowtriangle^* (\lambda_\chi N_1.N_2)N_3$. By β_{p} we have $(\lambda_\chi N_1.N_2)N_3 \rightarrowtriangle N_2[\mathsf{solve}(N_1 \ll^{\mathcal{E}}_{\chi} N_3)]$. Therefore, $(\lambda_\chi M_1.M_2)M_3 \rightarrowtriangle^* N_2[\mathsf{solve}(N_1 \ll^{\mathcal{E}}_{\chi} N_3)]$.

- Let $W_1 = (M_1 + M_2)M_3 \rightrightarrows W_2 = N_1 N_3 + N_2 N_3$ with $M_1 \rightrightarrows N_1$, $M_2 \rightrightarrows N_2$ and $M_3 \rightrightarrows N_3$. Then by IH we have $M_1 \rightarrowtriangle^* N_1$, $M_2 \rightarrowtriangle^* N_2$ and $M_3 \rightarrowtriangle^* N_3$. By compatibility of $\rightarrowtriangle^*$ we have $(M_1 + M_2)M_3 \rightarrowtriangle^* (N_1 + N_2)N_3$. By A_{rd}, $(N_1 + N_2)N_3$ reduces to $N_1 N_3 + N_2 N_3$ and, hence, we conclude that $(M_1 + M_2)M_3 \rightarrowtriangle^* N_1 N_3 + N_2 N_3$.

- The case $W_1 = \lambda_\chi M_1.M_2 \rightrightarrows W_2 = \lambda_\chi N_1.N_2$ with $M_1 \rightrightarrows N_1$ and $M_2 \rightrightarrows N_2$ is similar to the case $W_1 = M_1 M_2$ and $W_2 = N_1 N_2$.

- Let $W_1 = \lambda_\chi M_1.(M_2 + M_3) \rightrightarrows W_2 = \lambda_\chi N_1.N_2 + \lambda_\chi N_1.N_3$ with $M_1 \rightrightarrows N_1$, $M_2 \rightrightarrows N_2$ and $M_3 \rightrightarrows N_3$. Then by the IH we have $M_1 \rightarrowtriangle^* N_1$, $M_2 \rightarrowtriangle^* N_2$ and $M_3 \rightarrowtriangle^* N_3$. By compatibility of $\rightarrowtriangle^*$ we get $\lambda_\chi M_1.(M_2 + M_3) \rightarrowtriangle^* \lambda_\chi N_1.(N_2 + N_3)$. By λ_{ld} we have that $\lambda_\chi N_1.(N_2 + N_3)$ reduces to $\lambda_\chi N_1.N_2 + \lambda_\chi N_1.N_3$ and, hence, we conclude $\lambda_\chi M_1.(M_2 + M_3) \rightarrowtriangle^* \lambda_\chi N_1.N_2 + \lambda_\chi N_1.N_3$.

- The case $W_1 = M_1+M_2 \rightrightarrows W_2 = N_1+N_2$ with $M_1 \rightrightarrows N_1$ and $M_2 \rightrightarrows N_2$ is also similar to the case $W_1 = M_1M_2$ and $W_2 = N_1N_2$ above.

□

From this lemma, we immediately get that the relations $\rightarrowtriangle^*$ and $\rightrightarrows^*$ are the same:

Corollary 3.6. $\rightarrowtriangle^* = \rightrightarrows^*$.

Lemma 3.7 (Fundamental Lemma)**.** *For all terms M, M' and substitutions ϑ, ϑ' with $dom(\vartheta) = dom(\vartheta')$, if $M \rightrightarrows M'$ and $\vartheta \rightrightarrows \vartheta'$, then $M\vartheta \rightrightarrows M'\vartheta'$.*

Proof. By induction on the length of the derivation of $M \rightrightarrows M'$.

Case $M = M'$. We can proceed by structural induction on M, using the definitions of substitution application (Definition 2.8) and parallel reduction (Definitions 3.1 and 3.2).

- Let $M = x$ and $x \in dom(\vartheta)$. By Definition 3.2 we have $x\vartheta \rightrightarrows x\vartheta'$.
- Let $M = y$ and $y \notin dom(\vartheta)$. By the assumption, $y \notin dom(\vartheta')$. By Definition 2.8 and reflexivity of $\rightrightarrows$ we get $y\vartheta' = y \rightrightarrows y = y\vartheta'$.
- The cases when $M = \hat{x}$ or $M = f$ are analogous to the previous one.
- Let $M = M_1M_2$ with $M_1 \rightrightarrows M_1$ and $M_2 \rightrightarrows M_2$. By the IH, we have $M_1\vartheta \rightrightarrows M_1\vartheta'$ and $M_2\vartheta \rightrightarrows M_2\vartheta'$. By Definition 3.1, we get $M_1\vartheta M_2\vartheta \rightrightarrows M_1\vartheta' M_2\vartheta'$. By Definition 2.8, we conclude $(M_1M_2)\vartheta \rightrightarrows (M_1M_2)\vartheta'$.
- The cases $M = \lambda_\chi M_1.M_2$ or $M = M_1 + M_2$ with $M_1 \rightrightarrows M_1$ and $M_2 \rightrightarrows M_2$ are analogous to the previous one.

Case $M \neq M'$. We consider cases depending on the rules of parallel reduction.

- Let $M = (\lambda_\chi M_1.M_2)M_3 \rightrightarrows M' = M'_2[\mathsf{solve}(M'_1 \ll^{\mathcal{E}}_{\chi} M'_3)]$ with $M_1 \rightrightarrows M'_1$, $M_2 \rightrightarrows M'_2$, $M_3 \rightrightarrows M'_3$. By the IH, we have $M_1\vartheta \rightrightarrows M'_1\vartheta'$, $M_2\vartheta \rightrightarrows M'_2\vartheta'$, $M_3\vartheta \rightrightarrows M'_3\vartheta'$. By Definition 3.1, we have $(\lambda M_1\vartheta.M_2\vartheta)M_3\vartheta \rightrightarrows (M'_2\vartheta')[\mathsf{solve}(M'_1\vartheta' \ll^{\mathcal{E}}_{\chi} M'_3\vartheta')]$.

  First, assume that $\mathsf{solve}(M'_1 \ll^{\mathcal{E}}_{\chi} M'_3) = \emptyset$. Then by $\mathbf{C}_1$ we also have $\mathsf{solve}(M'_1\vartheta' \ll^{\mathcal{E}}_{\chi} M'_3\vartheta') = \emptyset$. Hence, we get $M'_2[\mathsf{solve}(M'_1 \ll^{\mathcal{E}}_{\chi} M'_3)]\vartheta' \approx_{\mathcal{E}} \lambda_{\{x\}}\hat{x}.x \approx_{\mathcal{E}} (M'_2\vartheta')[\mathsf{solve}(M'_1\vartheta' \ll^{\mathcal{E}}_{\chi} M'_3\vartheta')]$.

  Now assume $\mathsf{solve}(M'_1 \ll^{\mathcal{E}}_{\chi} M'_3) = \{\sigma_1,\dots,\sigma_n\}$, $n > 0$, $\mathsf{solve}(M'_1\vartheta' \ll^{\mathcal{E}}_{\chi} M'_3\vartheta') = \{\rho_1,\dots,\rho_k\}$, $k > 0$, and show $(\Sigma^n_{i=1}M'_2\sigma_i)\vartheta' \approx_{\mathcal{E}} \Sigma^k_{i=1}(M'_2\vartheta')\rho_i$. Since $\mathsf{solve}(M'_1 \ll^{\mathcal{E}}_{\chi} M'_3) = \{\sigma_1,\dots,\sigma_n\}$, by $\mathbf{C}_1$ we have $\mathsf{solve}(M'_1\vartheta' \ll^{\mathcal{E}}_{\chi} M'_3\vartheta') = \{(\sigma_1\vartheta')_{|\chi},\dots, (\sigma_n\vartheta')_{|\chi}\}$ (one can always guarantee $\mathtt{fm}(\vartheta') \cap \hat{\chi} = \emptyset$, renaming bound variables). Since $\mathsf{solve}(M'_1\vartheta' \ll^{\mathcal{E}}_{\chi} M'_3\vartheta') = \{\rho_1,\dots,\rho_k\}$, we have $\{(\sigma_1\vartheta')_{|\chi},\dots,(\sigma_n\vartheta')_{|\chi}\} \approx_{\mathcal{E}} \{\rho_1,\dots,\rho_k\}$ (set equality modulo $\approx_{\mathcal{E}}$) and, hence, $n \geq k$. So, if we manage to prove that $\sigma_i\vartheta' \approx_{\mathcal{E}} \vartheta'(\sigma_i\vartheta')_{|\chi}$ holds for all $1 \leq i \leq n$, we will get $\Sigma^n_{i=1}M'_2\sigma_i\vartheta' \approx_{\mathcal{E}} \Sigma^n_{i=1}M'_2\vartheta'(\sigma_i\vartheta')_{|\chi} \approx_{\mathcal{E}} \Sigma^k_{i=1}M'_2\vartheta'\rho_i$, which, by Definition 2.8, would imply $(\Sigma^n_{i=1}M'_2\sigma_i)\vartheta' \approx_{\mathcal{E}} \Sigma^k_{i=1}(M'_2\vartheta')\rho_i$.

  Proving $\sigma_i\vartheta' \approx_{\mathcal{E}} \vartheta'(\sigma_i\vartheta')_{|\chi}$ means to prove $x\sigma_i\vartheta' \approx_{\mathcal{E}} x\vartheta'(\sigma_i\vartheta')_{|\chi}$ for all x. There are two cases:

  - $x \in \chi$. Then $x \in dom(\sigma_i)$, since $dom(\sigma_i) = \chi$. We show $x\vartheta'(\sigma_i\vartheta')_{|\chi} \approx_{\mathcal{E}} x\sigma_i\vartheta'$. Since we can assume $dom(\vartheta') \cap \chi = \emptyset$, by the substitution application we have $x\vartheta'(\sigma_i\vartheta')_{|\chi} \approx_{\mathcal{E}} x(\sigma_i\vartheta')_{|\chi}$. Since $dom(\vartheta') \cap dom(\sigma_i) = \emptyset$ (it can always be guaranteed through renaming of bound matchables and variables), and since $x \in dom(\sigma_i)$, by the substitution composition we get $x(\sigma_i\vartheta')_{|\chi} \approx_{\mathcal{E}} x\sigma_i\vartheta'$.
  - $x \notin \chi$. Then $x \notin dom(\sigma_i)$, since $dom(\sigma_i) = \chi$. We show $x\vartheta'(\sigma_i\vartheta')_{|\chi} \approx_{\mathcal{E}} x\sigma_i\vartheta'$. By the substitution application, $x\sigma_i\vartheta' \approx_{\mathcal{E}} x\vartheta'$. Since $dom(\sigma_i\vartheta')_{|\chi} = \chi$, we can rename bound variables so that $dom(\sigma_i\vartheta')_{|\chi} \cap \mathtt{fv}(\vartheta') = \emptyset$. By substitution application and composition, we get $x\vartheta'(\sigma_i\vartheta')_{|\chi} \approx_{\mathcal{E}} x\vartheta'$.

- Let $M = M_1M_2 \rightrightarrows M' = M'_1M'_2$ with $M_1 \rightrightarrows M'_1$ and $M_2 \rightrightarrows M'_2$. By the IH, we have $M_1\vartheta \rightrightarrows M_1'\vartheta'$ and $M_2\vartheta \rightrightarrows M_2'\vartheta'$. By Definitions 3.1 and 2.8, we conclude $(M_1M_2)\vartheta = M_1\vartheta M_2\vartheta \rightrightarrows M_1'\vartheta' M_2'\vartheta' = (M_1'M_2')\vartheta'$.

- The cases when $M = \lambda_\chi M_1.M_2 \rightrightarrows M' = \lambda_\chi M_1'.M_2'$ or $M = M_1 + M_2 \rightrightarrows M' = M_1' + M_2'$ with $M_1 \rightrightarrows M_1'$ and $M_2 \rightrightarrows M_2'$ are analogous to the previous one.

- Let $M = \lambda_\chi M_1.(M_2 + M_3) \rightrightarrows M' = \lambda_\chi M_1'.M_2' + \lambda_\chi M_1'.M_3'$ with $M_1 \rightrightarrows M_1', M_2 \rightrightarrows M_2'$ and $M_3 \rightrightarrows M_3'$. By the IH, we have $M_1\vartheta \rightrightarrows M_1'\vartheta'$, $M_2\vartheta \rightrightarrows M_2'\vartheta'$, $M_3\vartheta \rightrightarrows M_3'\vartheta'$. By Definitions 3.1 and 2.8, we get $(\lambda_\chi M_1.(M_2 + M_3))\vartheta = \lambda_\chi M_1\vartheta.(M_2\vartheta + M_3\vartheta) \rightrightarrows \lambda_\chi M_1'\vartheta'.M_2'\vartheta' + \lambda_\chi M_1'\vartheta'.M_3'\vartheta' = (\lambda_\chi M_1'.M_2' + \lambda_\chi M_1'.M_3')\vartheta'$.

- When $M = (M_1 + M_2)M_3 \rightrightarrows M' = M_1'M_3' + M_2'M_3'$ with $M_1 \rightrightarrows M_1', M_2 \rightrightarrows M_2'$ and $M_3 \rightrightarrows M_3'$, the reasoning is similar to the previous case.

□

Lemma 3.8 (Diamond Property). *For all M, N, and Q, if $M \rightrightarrows N$ and $M \rightrightarrows Q$, then there exists W such that $N \rightrightarrows W$ and $Q \rightrightarrows W$.*

Proof. We prove the lemma by induction on the structure of M.

Case 1. M is a variable, matchable, or a function symbol. In this case $M = N = Q$, and the lemma holds trivially.

Case 2. M is an abstraction.

2.1. First, we consider the case, when

$$M = \lambda_\chi M_1.M_2, \quad N = \lambda_\chi N_1.N_2, \quad Q = \lambda_\chi Q_1.Q_2$$

with $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$, and $M_2 \rightrightarrows Q_2$. Applying the IH to M_1 and to M_2, we get that there exist two terms W_1 and W_2 such that $N_1 \rightrightarrows W_1$, $Q_1 \rightrightarrows W_1$, $N_2 \rightrightarrows W_2$, and $Q_2 \rightrightarrows W_2$. Taking $W = \lambda_\chi W_1.W_2$, by Definition 3.1, we can conclude $N \rightrightarrows W$ and $Q \rightrightarrows W$.

2.2. Now assume $M = \lambda_\chi M_1.(M_2 + M_3)$. We have four cases depending on the forms of N and Q:

2.2.1. Let first

$$N = \lambda_\chi N_1.N_2 + \lambda_\chi N_1.N_3, \qquad Q = \lambda_\chi Q_1.Q_2 + \lambda_\chi Q_1.Q_3$$

with $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$, $M_2 \rightrightarrows Q_2$, $M_3 \rightrightarrows N_3$, and $M_3 \rightrightarrows Q_3$. By the IH, there exist terms W_1, W_2, and W_3, such that $N_1 \rightrightarrows W_1$, $Q_1 \rightrightarrows W_1$, $N_2 \rightrightarrows W_2$, $Q_2 \rightrightarrows W_2$, $N_3 \rightrightarrows W_3$, and $Q_3 \rightrightarrows W_3$. Taking $W = \lambda_\chi W_1.W_2 + \lambda_\chi W_1.W_3$, by Definition 3.1, we can conclude $N \rightrightarrows W$ and $Q \rightrightarrows W$.

2.2.2. Let now

$$N = \lambda_\chi N_1.(N_2 + N_3), \qquad Q = \lambda_\chi Q_1.Q_2 + \lambda_\chi Q_1.Q_3$$

again with $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$, $M_2 \rightrightarrows Q_2$, $M_3 \rightrightarrows N_3$, and $M_3 \rightrightarrows Q_3$. The reasoning goes exactly like in the previous case, with the only difference that we need another $\rightrightarrows$-rule from Definition 3.1 to conclude $N = \lambda_\chi N_1.(N_2 + N_3) \rightrightarrows \lambda_\chi W_1.W_2 + \lambda_\chi W_1.W_3 = W$.

2.2.3. With a reasoning similar to above, we can prove the lemma, when

$$N = \lambda_\chi N_1.N_2 + \lambda_\chi N_1.N_3, \qquad Q = \lambda_\chi Q_1.(Q_2 + Q_3).$$

2.2.4. Finally, the case with

$$N = \lambda_\chi N_1.(N_2 + N_3), \qquad Q = \lambda_\chi Q_1.(Q_2 + Q_3)$$

is included in case 2.1.

Case 3. M is an application.

3.1. First, we consider the case, when

$$M = M_1 M_2, \quad N = N_1 N_2, \quad Q = Q_1 Q_2$$

with $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$, and $M_2 \rightrightarrows Q_2$. Applying the IH to M_1 and M_2, we get that there exist terms W_1 and W_2 such that $N_1 \rightrightarrows W_1$, $Q_1 \rightrightarrows W_1$, $N_2 \rightrightarrows W_2$, and $Q_2 \rightrightarrows W_2$. Taking $W = W_1 W_2$, by Definition 3.1, we conclude that $N \rightrightarrows W$ and $Q \rightrightarrows W$.

3.2. Now, we assume that $M = (M_1 + M_2)M_3$. We have the following four cases depending on the forms of N and Q.

3.2.1. Let first

$$N = N_1N_3 + N_2N_3, \qquad Q = Q_1Q_3 + Q_2Q_3$$

with $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$, $M_2 \rightrightarrows Q_2$, $M_3 \rightrightarrows N_3$, and $M_3 \rightrightarrows Q_3$. By the IH, there exist terms W_1, W_2, and W_3, such that $N_1 \rightrightarrows W_1$, $Q_1 \rightrightarrows W_1$, $N_2 \rightrightarrows W_2$, $Q_2 \rightrightarrows W_2$, $N_3 \rightrightarrows W_3$, and $Q_3 \rightrightarrows W_3$. Taking $W = W_1W_2 + W_1W_3$, by Definition 3.1, we can conclude $N \rightrightarrows W$ and $Q \rightrightarrows W$.

3.2.2. Let now

$$N = (N_1 + N_2)N_3, \qquad Q = Q_1Q_3 + Q_2Q_3$$

with $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$, $M_2 \rightrightarrows Q_2$, $M_3 \rightrightarrows N_3$, and $M_3 \rightrightarrows Q_3$. The reasoning goes exactly like in the previous case, with the only difference that we need another $\rightrightarrows$-rule from Definition 3.1 to conclude $N = (N_1 + N_2)N_3 \rightrightarrows W_1W_2 + W_1W_3 = W$.

3.2.3. With a reasoning similar to the previous case, we can prove the lemma, when

$$N = N_1N_3 + N_2N_3, \qquad Q = (Q_1 + Q_2)Q_3.$$

3.2.4. The case, when

$$N = (N_1 + N_2)N_3, \qquad Q = (Q_1 + Q_2)Q_3$$

is included in the case 3.1.

3.3. Now assume $M = (\lambda_\chi M_1.M_2)M_3$. We have the following four cases depending on the forms of N and Q.

3.3.1. First, let N be obtained from $(\lambda_\chi N_1.N_2)N_3$ by $\mathsf{solve}(N_1 \ll_\chi^{\mathcal{E}} N_3) = \{\sigma_1, \ldots, \sigma_n\}$, $n \geq 0$, and Q be obtained from the $(\lambda_\chi Q_1.Q_2)Q_3$ by $\mathsf{solve}(Q_1 \ll_\chi^{\mathcal{E}} Q_3) = \{\vartheta_1, \ldots, \vartheta_k\}$, $k \geq 0$:

$$N = N_2[\{\sigma_1, \ldots, \sigma_n\}], \qquad Q = Q_2[\{\vartheta_1, \ldots, \vartheta_k\}]$$

where $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$, $M_2 \rightrightarrows Q_2$, $M_3 \rightrightarrows N_3$, and $M_3 \rightrightarrows Q_3$. Applying the IH to M_1, M_2, and M_3 we get that there exist such W_1, W_2, and W_3 that $N_1 \rightrightarrows W_1$, $Q_1 \rightrightarrows W_1$, $N_2 \rightrightarrows W_2$, $Q_2 \rightrightarrows W_2$, $N_3 \rightrightarrows W_3$, and $Q_3 \rightrightarrows W_3$.

Now, we apply $\mathbf{C}_2$ with $N_1 \rightrightarrows W_1$ and $N_3 \rightrightarrows W_3$, getting (by Definition 3.4) that there exist $\rho_1, \ldots \rho_m$ such that $\mathsf{solve}(W_1 \ll^{\mathcal{E}}_{\chi} W_3) = \{\rho_1, \ldots, \rho_m\}$, $m \geq 0$, where each σ_i $\rightrightarrows$-reduces to some ρ_j, and each ρ_j is a parallel reduction of some σ_i. If $n = 0$ then $m = 0$ and we get $N \approx_{\mathcal{E}} \lambda_{\{x\}}\hat{x}.x \approx_{\mathcal{E}} W_2[\{\rho_1, \ldots, \rho_m\}]$. If $n > 0$, then also $m > 0$. By Lemma 3.7 we get that each $N_2\sigma_i$ $\rightrightarrows$-reduces to some $W_2\rho_j$, and each $W_2\rho_j$ is a parallel reduction of some $N_2\sigma_i$ ($1 \leq i \leq n$, $1 \leq j \leq m$). From this, Definition 3.1 gives $\Sigma^n_{i=1} N_2\sigma_i \rightrightarrows \Sigma^m_{i=1} W_2\rho_i$. But $n > 0$ and $m > 0$, by Definition 2.9, imply $N = \Sigma^n_{i=1} N_2\sigma_i$ and $W_2[\{\rho_1, \ldots, \rho_m\}] = \Sigma^m_{i=1} W_2\rho_i$. Hence, we got $N \rightrightarrows W_2[\{\rho_1, \ldots, \rho_m\}]$.

We can reason in the same way, applying $\mathbf{C}_2$ to $Q_1 \rightrightarrows W_1$ and $Q_3 \rightrightarrows W_3$ to obtain $Q \rightrightarrows W_2[\{\rho_1, \ldots, \rho_m\}]$.

Taking $W = W_2[\{\rho_1, \ldots, \rho_m\}]$, we get $N \rightrightarrows W$ and $Q \rightrightarrows W$, which proves this case.

3.3.2. Let now N be obtained from $(\lambda_\chi N_1.N_2)N_3$ by $\mathsf{solve}(N_1 \ll^{\mathcal{E}}_{\chi} N_3) = \{\sigma_1, \ldots, \sigma_n\}$, $n \geq 0$:

$$N = N_2[\{\sigma_1, \ldots, \sigma_n\}], \qquad Q = (\lambda_\chi Q_1.Q_2)Q_3$$

with $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$, $M_2 \rightrightarrows Q_2$, $M_3 \rightrightarrows N_3$ and $M_3 \rightrightarrows Q_3$. Applying the IH to M_1, M_2, and M_3 we get that there exist such W_1, W_2, and W_3 that $N_1 \rightrightarrows W_1$, $Q_1 \rightrightarrows W_1$, $N_2 \rightrightarrows W_2$, $Q_2 \rightrightarrows W_2$, $N_3 \rightrightarrows W_3$, and $Q_3 \rightrightarrows W_3$.

Reasoning in the same way as in the previous case, we can show that $N \rightrightarrows W_2[\{\rho_1, \ldots, \rho_m\}]$, where $\{\rho_1, \ldots, \rho_m\} = \mathsf{solve}(W_1 \ll^{\mathcal{E}}_{\chi} W_3)$.

On the other hand, by Definition 3.1, from $Q_1 \rightrightarrows W_1$, $Q_2 \rightrightarrows W_2$, $Q_3 \rightrightarrows W_3$, and $\mathsf{solve}(W_1 \ll^{\mathcal{E}}_{\chi} W_3) = \{\rho_1, \ldots \rho_m\}$ we get $Q \rightrightarrows W_2[\{\rho_1, \ldots, \rho_m\}]$.

Taking $W = W_2[\{\rho_1, \ldots, \rho_m\}]$, we get $N \rightrightarrows W$ and $Q \rightrightarrows W$, which proves this case.

3.3.3. With a reasoning similar to the previous case, we can prove the lemma, when

$$N = (\lambda_\chi N_1.N_2)N_3 \qquad Q = Q_2[\{\vartheta_1, \ldots, \vartheta_k\}].$$

3.3.4. The case when

$$N = (\lambda_\chi N_1.N_2)N_3 \qquad Q = (\lambda_\chi Q_1.Q_2)Q_3$$

is included in the case 3.1.

Case 4. M is a sum. Let

$$M = M_1 + M_2, \quad N = N_1 + N_2, \quad Q = Q_1 + Q_2$$

with $M_1 \rightrightarrows N_1$, $M_1 \rightrightarrows Q_1$, $M_2 \rightrightarrows N_2$ and, $M_2 \rightrightarrows Q_2$. Applying the IH to M_1 and M_2, we get that there exist two terms W_1 and W_2 such that $N_1 \rightrightarrows W_1$, $Q_1 \rightrightarrows W_1$, $N_2 \rightrightarrows W_2$, and $Q_2 \rightrightarrows W_2$. Taking $W = W_1 + W_2$, by Definition 3.1 we conclude that $N \rightrightarrows W$ and $Q \rightrightarrows W$. □

Theorem 3.9. *The pattern calculus with finitary matching is confluent if* solve *satisfies* $\mathbf{C}_1$ *and* $\mathbf{C}_2$*.*

Proof. We need to show that the relation $\twoheadrightarrow^*$ is confluent. From Corollary 3.6 of Lemma 3.5 we have $\twoheadrightarrow^* = \rightrightarrows^*$. By Lemma 3.8, the relation $\rightrightarrows$ has the diamond property and, therefore, $\rightrightarrows^*$ is confluent. Hence, $\twoheadrightarrow^*$ is confluent. □

4. Instantiations of the Finitary Pattern Calculus

In this section we define instances of the solve function that satisfy the conditions $\mathbf{C}_1$ and $\mathbf{C}_2$.

First, a generic definition of solve is given, which uses (as a parameter) an algorithm that solves matching problems modulo some equational theory. Then we consider two concrete equational theories, for commutative and flat symbols, and give the corresponding rule-based matching algorithms. Both are examples of finitary matching, described in a rule-based way. In this way, we obtain concrete instances of solve for commutative and flat theories. For the commutative theory, we also consider a polymorphic version. We then prove that each of these instances satisfy $\mathbf{C}_1$ and $\mathbf{C}_2$.

Matching equations that are passed to solve arise from terms during reduction. Since our terms follow the variable name convention, we can be sure that in $P \ll^{\mathcal{E}}_{\chi} Q$ we always have $\mathtt{fm}(Q) \cap \hat{\chi} = \emptyset$ and $\mathtt{fv}(Q) \cap \chi = \emptyset$. We call this assumption the *binding variable separation convention.* Due to this convention, solution of matching problems considered in this section will be idempotent. Composition of two domain-disjoint idempotent substitutions can be obtained by just taking the union of their set representations. This observation justifies the use of substitution union instead of their composition in the definitions below.

A *match* μ is either a substitution, a special constant **fail** or a special constant **wait**. A match is *positive* if it is a substitution; it is *decided* if it is either positive or **fail**.

The disjoint union of two matches is an operation that plays very important role in pattern calculi. In the literature, there are two ways of defining it, see, e.g., (van Oostrom and van Raamsdonk, 2015): the *strict approach,* where **wait** is dominant over **fail**, and the *non-strict approach*, where **fail** dominates over **wait**. Strict approach is taken, for instance, in (Jay and Kesner, 2006; Jay, 2009), while the non-strict approach has been followed by Jay and Kesner (2009) and Bonelli et al. (2012).

Definition 4.1 (Strict and non-strict disjoint union of matches)**.** Given two matches μ_1 and μ_2, their *disjoint union* $\mu_1 \uplus \mu_2$ is a match defined as follows:

In the strict approach:

- If μ_1 and μ_2 are substitutions with $dom(\mu_1) \cap dom(\mu_2) = \emptyset$, then $\mu_1 \uplus \mu_2 = \mu_1 \cup \mu_2$.
- If μ_1 or μ_2 is **wait**, then $\mu_1 \uplus \mu_2 =$ **wait**.
- Otherwise, $\mu_1 \uplus \mu_2 =$ **fail**.

In the non-strict approach:

- If μ_1 and μ_2 are substitutions with $dom(\mu_1) \cap dom(\mu_2) = \emptyset$, then $\mu_1 \uplus \mu_2 = \mu_1 \cup \mu_2$.
- If μ_1 or μ_2 is **wait** and none of them is **fail**, then $\mu_1 \uplus \mu_2 =$ **wait**.
- Otherwise, $\mu_1 \uplus \mu_2 =$ **fail**.

Hence, in the strict approach, the following equality holds:

$$\mathbf{fail} \uplus \mathbf{wait} = \mathbf{wait} \uplus \mathbf{fail} = \mathbf{wait},$$

while in the non-strict approach we have

$$\mathbf{fail} \uplus \mathbf{wait} = \mathbf{wait} \uplus \mathbf{fail} = \mathbf{fail}.$$

We assume that $\uplus$ associates left and omit parentheses: $\mu_1 \uplus \mu_2 \uplus \mu_3$ will stand for $((\mu_1 \uplus \mu_2) \uplus \mu_3)$. Note that non-strict disjoint union is not associative: $\mathbf{wait} \uplus \{x \mapsto M\} \uplus \{x \mapsto N\} = \mathbf{wait}$, but $\mathbf{wait} \uplus (\{x \mapsto M\} \uplus \{x \mapsto N\}) = \mathbf{fail}$.

The operation of disjoint union extends to sets of matches $\mathcal{M}_1$ and $\mathcal{M}_2$ as follows: $\mathcal{M}_1 \uplus \mathcal{M}_2 = \{\mu_1 \uplus \mu_2 \mid \mu_1 \in \mathcal{M}_1 \text{ and } \mu_2 \in \mathcal{M}_2\}$. This definition implies that $\emptyset \uplus \mathcal{M} = \mathcal{M} \uplus \emptyset = \emptyset$.

An $\mathcal{E}$*-matching problem* is defined by the following grammar:

$$\Gamma ::= \{\!\!\{ P \ll^{\mathcal{E}}_{\chi} Q \}\!\!\} \mid \mu \mid \Gamma_1 \uplus \Gamma_2.$$

An *equational matching rule* (briefly, $\mathcal{E}$*-matching rule*) transforms an $\mathcal{E}$-matching problem of the form of an equation $\{\!\!\{ P \ll^{\mathcal{E}}_{\chi} Q \}\!\!\}$ into an $\mathcal{E}$-matching problem. We use the notation $\{\!\!\{ P \ll^{\mathcal{E}}_{\chi} Q \}\!\!\} \rightsquigarrow \Gamma$ for matching rules.

It can happen that a matching equation is transformed by matching rules in multiple ways. We assume that we have finitely many matching rules, they are all terminating, and for any equation there is at least one rule that applies to it.

Definition 4.2. The matching function solve is a partial function from the quadruple $P, Q, \chi, \mathcal{E}$ to a finite set of substitutions. With $\mathcal{E}$, the corresponding $\mathcal{E}$-matching rules are assumed to be given. Then the matching function is written as $\mathsf{solve}_{\mathcal{E}}(P \ll^{\mathcal{E}}_{\chi} Q)$ and is defined as follows:

- The initial problem $\{\!\!\{ P \ll^{\mathcal{E}}_{\chi} Q \}\!\!\}$ is created and the matching rules are applied as long as possible. If the same equation can be transformed in multiple ways, it is done concurrently. The process ends with a finite set $\mathcal{M}$ of matches.

- If $\mathbf{wait} \in \mathcal{M}$, then $\mathsf{solve}_{\mathcal{E}}(P \ll^{\mathcal{E}}_{\chi} Q)$ is *not defined.*

- Otherwise, $\mathsf{solve}_{\mathcal{E}}(P \ll^{\mathcal{E}}_{\chi} Q)$ is defined as

$$\mathsf{solve}_{\mathcal{E}}(P \ll^{\mathcal{E}}_{\chi} Q) = \mathcal{M} \setminus \Big(\{\mathbf{fail}\} \cup \{\sigma \mid \mathit{dom}(\sigma) \neq \chi \text{ or } P\hat{\sigma} \not\approx_{\mathcal{E}} Q\}\Big).$$

The last item of this definition needs an explanation: When $\mathcal{M}$ does not contain **wait**, we will collect successful matches from it, i.e., we ignore **fail**. Moreover, from those successful matches the interesting ones are only those substitutions whose domain coincides to χ and which match P to Q modulo $\mathcal{E}$, as the definition of solve requires.

Sequence of rule applications in Definition 4.2 form a *derivation*. Note that while computing solve, it is not always necessary to develop derivations till the end, until all the ingredients are matches. For instance, one may stop the derivation with μ, if the matching problem $\mu \uplus \Gamma$ or $\Gamma \uplus \mu$ is reached, where $\mu = \mathbf{wait}$ for the strict $\uplus$ and $\mu = \mathbf{fail}$ for the non-strict $\uplus$.

One can see that $\mathsf{solve}_{\mathcal{E}}$ that uses non-strict disjoint union is defined (i.e., gives a definite answer) more often than the one with strict disjoint union.

A matching equation $P \ll^{\mathcal{E}}_{\chi} Q$ is called *linear*, if no matchable from $\hat{\chi}$ appears in P more than once. Otherwise it is *nonlinear*.

In the next sections we will need terms in a special form, called *matchable-sum form*. Following the similar idea from (Jay and Kesner, 2009), the motivation is to have terms sufficiently reduced for matching to be defined. *Data structures* $\mathbf{D}$ and *matchable forms* $\mathbf{M}$ are defined via the grammar

$$\begin{aligned}\mathbf{D} &::= \hat{x} \mid f \mid \mathbf{D}N\\ \mathbf{M} &::= \mathbf{D} \mid \lambda_{\chi}P.N\end{aligned}$$

Matchable-sum forms $\mathbf{M}_+$ are matchable forms or their sums:

$$\mathbf{M}_+ ::= \mathbf{M} \mid \mathbf{M}_+ + \mathbf{M}_+$$

An equational theory $\mathcal{E}$ is over $\mathbf{D}$ (resp. over $\mathbf{M}$, over $\mathbf{M}_+$) iff for each axiom $N_1 = N_2$ of $\mathcal{E}$, both N_1 and N_2 are data structures (resp. matchable forms, matchable-sum forms).

It is interesting to see that these special kinds of terms have the following closure properties:

Lemma 4.3. *The set of data structures is closed under substitution application and reduction for equational theories over* $\mathbf{D}$.

The set of matchable-sum forms is closed under substitution application and reduction for equational theories over $\mathbf{M}_+$.

The set of matchable forms is closed under substitution application for equational theories over $\mathbf{M}$.

Proof. Since matchables and function symbols are not affected by substitution application and reduction and the equational theory equates data structures to data structures, it is obvious that the set data structures is closed under these operations. As for matchable forms (and for their sums), closure under substitution application follows from the fact that besides data structures, abstractions are also closed under this operation and the equational theory equates terms from the same class (matchable forms to matchable forms, matchable-sum forms to matchable-sum forms). Matchable forms are not closed under reduction: $\lambda_\chi P.(N_1 + N_2)$ is a matchable form, but its reduct by λ_{Id}, the term $\lambda_\chi P.N_1 + \lambda_\chi P.N_2$ is not. In general, reduction reduces matchable terms to matchable-sums, when the equational theories keeps equal terms within $\mathbf{M}$ or $\mathbf{M}_+$. Hence, we have closure of matchable-sums under reduction for equational theories over $\mathbf{M}_+$. □

Note that if we remove the condition about equational theories, the lemma does not hold anymore. For instance, if $\mathcal{E}$ is defined by the axiom $f(gx)x = \lambda_{\{y\}}y.y$ (i.e., it is defined over $\mathbf{M}$ but not over $\mathbf{D}$), then $f(ga)((\lambda_{\{y\}}.y)a)$ is a data structure, but $f(ga)((\lambda_{\{y\}}.y)a) \rightarrow f(ga)a \approx_\mathcal{E} \lambda_{\{y\}}y.y$ and the reduct is not a data structure anymore.

From this lemma and Lemma 3.5 we have the following corollary:

Corollary 4.4. *The set of data structures is closed under parallel reduction for equational theories over* $\mathbf{D}$*. The set of matchable-sum forms is closed under parallel reduction for equational theories over* $\mathbf{M}_+$*.*

Below we will need a restricted form of terms as well, which we call *values*. They contain no free variables, no subterms of the form $(\lambda_\chi P.\, N)Q$, no subterms of the form $\lambda_\chi P.\,(N_1 + N_2)$, and no subterms of the form $(M_1 + M_2)N$. All values are matchable-sum forms.

Definition 4.5. The *head* of a term M, denoted by *head*(M), is the head symbol of M, defined as follows:

- If M is a variable, matchable, or a function symbol, then $head(M) = M$.
- If M is an abstraction, then $head(M) = \lambda$.
- If M is a sum, $head(M) = +$.
- If M is an application M_1M_2, then $head(M) = head(M_1)$.

From the definition of data structures it is clear that each data structure has either a matchable or a function symbol as its head. The head of a matchable form can be either a matchable, a function symbol, or λ.

In the coming sections, we will consider equational theories over $\mathbf{D}$.

4.1. An Instance of solve *with Commutative Matching*

The equational theory for a commutative function symbol f is specified by the axiom schema $fx_1 \cdots x_n = fx_{\pi(1)} \cdots x_{\pi(n)}$, $n \geq 2$, for each non-identity permutation π of $(1, \ldots, n)$. We denote this theory by C.

Definition 4.6. Let $\mathbf{M}_+ = \mathbf{M}_1 + \cdots + \mathbf{M}_m$ be a matchable-sum such that no two $\mathbf{M}_i$ and $\mathbf{M}_j$, $i \neq j$, are $\approx_{\mathsf{C}}$-equivalent. We say that $\mathbf{M}_+$ is $\approx_{\mathsf{C}}$-*rigid*, if at least one of the following conditions are fulfilled:

- $m = 1$, or
- $\mathbf{M}_i$ is an abstraction for some $1 \leq i \leq m$, or
- $\mathbf{M}_i$ and $\mathbf{M}_j$ have different heads for some $1 \leq i, j \leq m$, or
- $\mathbf{M}_i$ and $\mathbf{M}_j$ are values for some $1 \leq i, j \leq m$, $i \neq j$.

To make use of this definition, below we assume that in sums, nested occurrences of $+$ are flattened out and all $\approx_{\mathsf{C}}$-equivalent summands are identified. That means, whenever a matchable sum occurs is matching problems, no two summands are $\approx_{\mathsf{C}}$-equivalent.

Example 4.7. Let f be a commutative function symbol. Then the following matchable sums are $\approx_{\mathsf{C}}$-rigid:

$$fab + gx, \quad fab + fac, \quad f(a+b) + f(a+c), \quad \lambda_{\emptyset} y.b + \lambda_{\emptyset} a.b,$$
$$fab + fax + \lambda_{\chi} P.N, \quad f(\lambda_{\{x,y\}}(\hat{x} + \hat{y}).gxy) + f(\lambda_{\{x,y\}}(\hat{x} + \hat{y}).gyx).$$

The following matchable-sums are not $\approx_{\mathsf{C}}$-rigid:

$$fab + fax, \quad fa + f((\lambda_{\{x\}}\hat{x}.x)a), \quad f((g+h)a) + f(ga + ha),$$
$$faa + fbx, \quad f(\lambda_{\emptyset} y.b) + f(\lambda_{\emptyset} a.b), \quad f(\lambda_{\emptyset}(a+b)c.b) + f(\lambda_{\emptyset}(ac+bc).b),$$
$$fa(\lambda_{\{x\}}\hat{x}.(x+b)) + f(\lambda_{\{x\}}\hat{x}.x + \lambda_{\{x\}}\hat{x}.b)a$$

Lemma 4.8. *$\approx_{\mathsf{C}}$-rigid matchable-sum forms are closed under substitution application and reduction.*

Proof. Let $\mathbf{M} = \mathbf{M}_1 + \cdots + \mathbf{M}_m$ be a $\approx_{\mathsf{C}}$-rigid matchable-sum and consider all the cases of the definition to show that both $\mathbf{M}\vartheta$ (for some ϑ) and $\mathbf{M}'$ with $\mathbf{M} \twoheadrightarrow \mathbf{M}'$ are $\approx_{\mathsf{C}}$-rigid matchable-sums:

- $m = 1$: Then $\mathbf{M}\vartheta$ has also exactly one summand. As for $\mathbf{M}'$, it has one or more, where more is possible only if $\mathbf{M}$ has the form $\lambda_\chi P.(N_1 + N_2)$, which gives $\mathbf{M}' = \lambda_\chi P.N_1 + \lambda_\chi P.N_2$ by the λ_{Id} rule. But then, by Definition 4.6, $\mathbf{M}'$ is also $\approx_{\mathsf{C}}$-rigid.

- $\mathbf{M}_i$ is an abstraction form some $1 \leq i \leq m$: $\mathbf{M}_i\vartheta$ is again an abstraction, therefore, $\mathbf{M}\vartheta$ is $\approx_{\mathsf{C}}$-rigid. The reduct of $\mathbf{M}_i$ can be either an abstraction or a sum of abstractions. In both cases $\mathbf{M}'$ is $\approx_{\mathsf{C}}$-rigid.

- $\mathbf{M}_i$ and $\mathbf{M}_j$ have different heads for some $1 \leq i, j \leq m$: Substitution application does not affect heads: $\mathbf{M}_i\vartheta$ and $\mathbf{M}_j\vartheta$ will have also different ones. For reduction, it is sufficient to consider the case when $\mathbf{M}_i$ and $\mathbf{M}_j$ are data structures. By Lemma 4.3, their reducts are also data structure with different heads, because reduction does not affect heads of data structures. Hence, $\mathbf{M}'$ is again $\approx_{\mathsf{C}}$-rigid.

- $\mathbf{M}_i$ and $\mathbf{M}_j$ are values for some $1 \leq i, j \leq m$, $i \neq j$: Neither substitution application nor reduction affects values.

□

From this lemma and Lemma 3.5, we have the following corollary:

Corollary 4.9. *$\approx_{\mathsf{C}}$-rigid matchable-sum forms are closed under parallel reduction.*

The commutative matching rules are the following ones: function symbol deletion (Del-F), matchable deletion (Del-M), matchable elimination (ME), matchable elimination under a commutative symbol (ME-C), application decomposition (Dec-App), decomposition under a commutative symbol (Dec-C), sum decomposition (Dec-Sum), fail (Fail), and the rule for wait (Wait). Rules are applied modulo $\approx$-equivalence.

Del-F: $\{\!\{ f \ll_\chi^{\mathsf{C}} f \}\!\} \rightsquigarrow Id$.

Del-M: $\{\!\{ \hat{x} \ll_\chi^{\mathsf{C}} \hat{x} \}\!\} \rightsquigarrow Id$, where $x \notin \chi$.

ME: $\{\!\{ \hat{x} \ll_\chi^{\mathsf{C}} Q \}\!\} \rightsquigarrow \{x \mapsto Q\}$, where $x \in \chi$.

ME-C: $\{\!\{ \hat{x}\, P_1 \cdots P_m \ll_\chi^{\mathsf{C}} f Q_1 \cdots Q_n \}\!\} \rightsquigarrow \vartheta \uplus \biguplus_{i=1}^{m} \{\!\{ P_i \ll_\chi^{\mathsf{C}} Q_{\pi(n-m+i)} \}\!\}$, where f is a commutative function symbol, $n \geq m > 0$, π is a permutation of $(1, \ldots, n)$, $x \in \chi$, and $\vartheta = \{x \mapsto f Q_{\pi(1)} \cdots Q_{\pi(n-m)}\}$.

Dec-App: $\{\!\{P_1P_2 \ll^{\mathsf{C}}_{\chi} Q_1Q_2\}\!\} \rightsquigarrow \{\!\{P_1 \ll^{\mathsf{C}}_{\chi} Q_1\}\!\} \uplus \{\!\{P_2 \ll^{\mathsf{C}}_{\chi} Q_2\}\!\}$, where $P_1\,P_2$ and $Q_1\,Q_2$ are matchable forms and $head(Q_1Q_2)$ is not a commutative function symbol.

Dec-C: $\{\!\{fP_1\cdots P_n \ll^{\mathsf{C}}_{\chi} fQ_1\cdots Q_n\}\!\} \rightsquigarrow \biguplus_{i=1}^{n}\{\!\{P_i \ll^{\mathsf{C}}_{\chi} Q_{\pi(i)}\}\!\}$, where f is a commutative function symbol, $n > 0$, and π is a permutation of $(1,\dots,n)$.

Dec-Sum: $\{\!\{P_1 + \cdots + P_m \ll^{\mathsf{C}}_{\chi} Q_1 + \cdots + Q_n\}\!\} \rightsquigarrow \biguplus_{i=1}^{m}\{\!\{P_i \ll^{\mathsf{C}}_{\chi} \sum_{Q\in\mathcal{Q}_i} Q\}\!\}$, where $m > 1$, $n \geq 1$, $P_1,\dots,P_m$ are matchable forms, $Q_1,\dots,Q_n$ are data structures, and each $\mathcal{Q}_i$ is a nonempty subset of $\{Q_1,\dots,Q_n\}$ such that $\mathcal{Q}_1 \cup\cdots\cup \mathcal{Q}_m = \{Q_1,\dots,Q_n\}$.

Fail: $\{\!\{P \ll^{\mathsf{C}}_{\chi} Q\}\!\} \rightsquigarrow$ **fail** if P and Q are matchable-sum forms such that if P is a data structure, then Q is a $\approx_{\mathsf{C}}$-rigid matchable-sum form, and no rule above is applicable.

Wait: $\{\!\{P \ll^{\mathsf{C}}_{\chi} Q\}\!\} \rightsquigarrow$ **wait**, if no other rule applies.

One can notice that the rules ME-C, Dec-C, and Dec-Sum can transform the same equation in multiple ways. On the other hand, no two rules apply to the same equation.

Definition 4.10. The instance of solve for a commutative theory, denoted $\mathsf{solve}_{\mathsf{C}}$, is defined by the construction from Definition 4.2, based on the rules for commutative matching as described above. Before $\mathsf{solve}_{\mathsf{C}}$ applies to P and Q, it is assumed that in P and Q all nested occurrences of $+$ are flattened out and $\approx_{\mathsf{C}}$-equivalent summands are identified, and the results of rule applications are simplified analogously.

Unless explicitly mentioned the opposite, the results below remain true for both strict and non-strict disjoint unions.

Lemma 4.11. *Let $P \ll^{\mathsf{C}}_{\chi} Q$ be a nonlinear commutative matching equation. Then $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q) = \emptyset$ or $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q)$ is undefined.*

Proof. For simplicity, assume that P contains two occurrences of $\hat{x}$ with $x \in \chi$. By analyzing the rules, one can see that unless Fail or Wait rules apply, at every branch of the derivation tree eventually a problem Γ will be generated, where Γ may contain at least one of the following forms of equation pairs:

- $\{\!\{\hat{x} \ll^{\mathsf{C}}_{\chi} Q\}\!\}$ and $\{\!\{\hat{x} \ll^{\mathsf{C}}_{\chi} N\}\!\}$,
- $\{\!\{\hat{x}P_1 \cdots P_p \ll^{\mathsf{C}}_{\chi} fQ_1 \cdots Q_q\}\!\}$ and $\{\!\{\hat{x} \ll^{\mathsf{C}}_{\chi} N\}\!\}$ for a commutative f,
- $\{\!\{\hat{x}P_1 \cdots P_p \ll^{\mathsf{C}}_{\chi} fQ_1 \cdots Q_q\}\!\}$ and $\{\!\{\hat{x}M_1 \cdots M_m \ll^{\mathsf{C}}_{\chi} fN_1 \cdots N_n\}\!\}$ for a commutative f.

Then, in the first case, applying the ME rule twice will give **fail**. In the second case it will happen after applying ME-C and ME rules, and in the third case after applying ME-C twice. Then, if the Wait rule does not interfere, it is guaranteed that in the computed set of matches consists of **fail** only. Otherwise it can be either **fail** or **wait**. In any case, there will be no positive match. □

This lemma says that matching with nonlinear patterns does not succeed. This is a pretty common requirement when proving confluence.

Theorem 4.12. $\mathsf{solve}_{\mathsf{C}}$ *is terminating and sound.*

Proof. Termination of $\mathsf{solve}_{\mathsf{C}}$ follows from the fact that every rule strictly decreases the multiset of sizes (number of symbols) of the left hand sides of matching equations.

Soundness is a straightforward consequence of Definition 4.10, since we retain only those substitutions that are solution of the matching problem. □

The function $\mathsf{solve}_{\mathsf{C}}$, obviously, is not complete. For instance, it can not compute the solution $\{x \mapsto a\}$ of the equation $\lambda_{\emptyset}\hat{x}.\hat{x} \ll^{\mathsf{C}}_{\{x\}} \lambda_{\emptyset}a.a$.

It is also obvious that some matching problems may have finitely many solutions. $f\hat{x}\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} fab$, with commutative f, is an example where $\mathsf{solve}_{\mathsf{C}}$ returns two substitutions $\{x \mapsto a, y \mapsto b\}$ and $\{x \mapsto b, y \mapsto a\}$. Due to the termination shown above, we can not have an infinitary case.

Since all nested occurrences of $+$ are flattened out and $\approx_{\mathsf{C}}$-equivalent summands are identified in the input, and the order of summands does not matter in the matching rules, $\mathsf{solve}_{\mathsf{C}}$ preserves $\approx_{\mathsf{C}}$-equivalence: Equivalent problems have equivalent solutions.

We note that for terms that do not contain function symbols and $+$, a pattern calculus with $\mathsf{solve}_{\mathsf{C}}$ as the matching function coincides to the *pure pattern calculus with matchable symbols* as defined by Jay and Kesner (2009). Indeed, for such terms the only applicable rules are Del-M, ME, Dec-App, Fail, and Wait, which correspond exactly to the rules of *compound matching* from (Jay and Kesner, 2009). The non-strict version of $\uplus$ should be used. The condition of Fail in such a case degenerates into a simple check whether P and Q are matchable forms.

Below we give several examples that illustrate $\mathsf{solve}_{\mathsf{C}}$. They show specific features of not only commutative symbols, but also of $+$.

Example 4.13. $\mathsf{solve}_{\mathsf{C}}(\hat{x}\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} (\lambda_{\{z,u\}} f\hat{z}\hat{u}.fzu)(fab))$ is undefined, because the right hand side is not a matchable form and can not be decomposed. However, if we reduce it, then the obtained problem is solvable. If f is commutative, there are even two solutions: $\mathsf{solve}_{\mathsf{C}}(\hat{x}\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} fab) = \{\{x \mapsto fa, y \mapsto b\}, \{x \mapsto fb, y \mapsto a\}\}$.

Example 4.14. $\mathsf{solve}_{\mathsf{C}}(fx + fa \ll^{\mathsf{C}}_{\emptyset} fa)$ is undefined, because after applying the Dec-Sum and Dec-App rules we will get an equation $\{\!\{x \ll^{\mathsf{C}}_{\emptyset} a\}\!\}$, to which only the Wait applies. If we replace x by a in the original equation, then there is a single solution, the identity substitution.

If we reverse the sides of the problem, $\mathsf{solve}_{\mathsf{C}}(fa \ll^{\mathsf{C}}_{\emptyset} fx + fa)$ is still undefined, because the Fail rule does not apply: Q is not a rigid matchable-sum. If we did not put this requirement in the condition of Fail, then we would have $\mathsf{solve}_{\mathsf{C}}(fa \ll^{\mathsf{C}}_{\emptyset} fx+fa) = \emptyset$, $\mathsf{solve}_{\mathsf{C}}(fa\{x \mapsto a\} \ll^{\mathsf{C}}_{\emptyset} (fx+fa)\{x \mapsto a\}) = \mathsf{solve}_{\mathsf{C}}(fa \ll^{\mathsf{C}}_{\emptyset} fa) = \{Id\}$, and $\mathbf{C}_1$ would be violated.

Example 4.15. $\mathsf{solve}_{\mathsf{C}}(f\hat{x} \ll^{\mathsf{C}}_{\{x\}} Q)$ is undefined in the following cases:

- $Q = f((\lambda_{\emptyset} b.a)b) + fa$,
- $Q = f(\lambda_{\emptyset} y.b) + f(\lambda_{\emptyset} a.b)$,
- $Q = f(\lambda_{\emptyset} b.(a+b)) + f(\lambda_{\emptyset} b.a + \lambda_{\emptyset} b.b)$,
- $Q = f((g+h)a) + f(ga + ha)$,
- $Q = f(\lambda_{\emptyset}(a+b)c.b) + f(\lambda_{\emptyset}(ac+bc).b)$.

A common feature of these Q's is that none of them is a $\approx_{\mathsf{C}}$-rigid matchable-sum. If we did not require $\approx_{\mathsf{C}}$-rigid matchable-sum right hand sides in the Fail rule, $\mathsf{solve}_{\mathsf{C}}$ would be the empty set for all these problems and it would violate $\mathbf{C}_1$ or $\mathbf{C}_2$. For instance, instantiating y by a in $f(\lambda_{\emptyset} y.b) + f(\lambda_{\emptyset} a.b)$ gives a solvable problem $\mathsf{solve}_{\mathsf{C}}(f\hat{x} \ll^{\mathsf{C}}_{\{x\}} f(\lambda_{\emptyset} a.b)) = \{\{x \mapsto \lambda_{\emptyset} a.b\}\}$ and $\mathbf{C}_1$ is violated. Reducing $f(\lambda_{\emptyset} b.(a+b)) + f(\lambda_{\emptyset} b.a + \lambda_{\emptyset} b.b)$ gives a solvable problem $\mathsf{solve}_{\mathsf{C}}(f\hat{x} \ll^{\mathsf{C}}_{\{x\}} f(\lambda_{\emptyset}b.a + \lambda_{\emptyset}b.b)) = \{\{x \mapsto \lambda_{\emptyset}b.a + \lambda_{\emptyset}b.b\}\}$ and $\mathbf{C}_2$ is violated.

Example 4.16. $\mathsf{solve}_{\mathsf{C}}(\hat{x} + \hat{y} \ll^{\mathsf{C}}_{\{x,y\}} \lambda_{\{z\}}\hat{z}.(a + b)) = \emptyset$. Dec-Sum does not apply, because $\lambda_{\{z\}}\hat{z}.(a+b)$ is not a data structure. No other rule, except Fail, matches the form of the equation. Fail applies because $\hat{x} + \hat{y}$ is a matchable-sum form and $\lambda_{\{z\}}\hat{z}.(a + b)$ is a (rigid) matchable-sum form. That means, there is only one branch in its derivation tree, which ends with **fail** and, hence, no substitutions are collected.

If we reduce the right hand side of the equation, we again get a matching problem on which $\mathsf{solve}_{\mathsf{C}}$ is defined and returns the empty set of substitutions: $\mathsf{solve}_{\mathsf{C}}(\hat{x} + \hat{y} \ll^{\mathsf{C}}_{\{x,y\}} \lambda_{\{z\}}\hat{z}.a + \lambda_{\{z\}}\hat{z}.b) = \emptyset$. Note that Dec-Sum rule does not apply.

Example 4.17. $\mathsf{solve}_{\mathsf{C}}(\hat{x} + f\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} (h + g)a)$ is undefined: Neither Dec-Sum nor Fail rules apply, since $(h + g)a$ is not even a matchable-sum form. If we reduce the right hand side of the equation, we get a matching problem on which $\mathsf{solve}_{\mathsf{C}}$ returns the empty set of substitutions: $\mathsf{solve}_{\mathsf{C}}(\hat{x} + f\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} ha + ga) = \emptyset$. All five derivations end with **fail**.

Example 4.18. $\mathsf{solve}_{\mathsf{C}}(\hat{x} + f\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} z)$ is undefined: Neither Dec-Sum nor Fail rules apply, because z is not a matchable form. The other rules do not match the form of the equation.

If we instantiate z, say, with $a+fv$, then the obtained problem is solvable: $\mathsf{solve}_{\mathsf{C}}(\hat{x} + f\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} a + fv) = \{\{x \mapsto a,\, y \mapsto v\}, \{x \mapsto a + fv,\, y \mapsto v\}\}$.

If we instantiate z with a, then the obtained problem has the empty set of solutions: $\mathsf{solve}_{\mathsf{C}}(\hat{x} + f\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} a) = \emptyset$, because there is only one branch in the derivation, eventually the problem $\{x \mapsto a\} \uplus \{\!\{f\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} a\}\!\}$ is obtained, which is transformed by the Fail rule, giving the only final match **fail**. The same result will be returned if z is instantiated by $\lambda_{\{v\}}\hat{v}.(a + b))$ or by $ga + gb + hv$ (note that both terms are rigid matchable-sums).

$\mathsf{solve}_{\mathsf{C}}$ remains undefined if we instantiate z by a matchable-sum form that is not rigid (e.g., $fa + fv$), or by a term that is not a matchable-sum form (e.g., $(f + g)a$).

Example 4.19. $\mathsf{solve}_{\mathsf{C}}(f\hat{x}\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} fz) = \emptyset$. Note that in whatever way we instantiate the free variable z in this example, $\mathsf{solve}_{\mathsf{C}}$ will always return the empty set.

Example 4.20. Let P and Q be respectively the terms $f\hat{x}\hat{y}$ and faW for some W, where f is commutative. Then $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\{x,y\}} Q) = \{\sigma_1, \sigma_2\}$, where

$$\begin{aligned}
&\sigma_1 = \{x \mapsto a,\, y \mapsto W\},\\
&\sigma_2 = \{x \mapsto W,\, y \mapsto a\}.
\end{aligned}$$

Let first W be a variable z and let $\vartheta = \{z \mapsto a\}$. Then $\mathsf{solve}_{\mathsf{C}}((P\vartheta \ll^{\mathsf{C}}_{\{x,y\}} Q\vartheta) = \mathsf{solve}_{\mathsf{C}}(f\hat{x}\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} faa) = \{\{x \mapsto a,\, y \mapsto a\}\}$. Hence, the number of solutions decreases, but we do have

$$\{(\sigma_1\vartheta)|_{\{x,y\}}, (\sigma_2\vartheta)|_{\{x,y\}}\} = \{\{x \mapsto a,\, y \mapsto a\}\}.$$

Now let W be $(\lambda_{\{z\}}\hat{z}\cdot z)a$. Then $P \rightrightarrows P' = P$, $Q \rightrightarrows Q' = faa$ and we get $\mathsf{solve}_{\mathsf{C}}((P' \ll^{\mathsf{C}}_{\{x,y\}} Q') = \mathsf{solve}_{\mathsf{C}}(f\hat{x}\hat{y} \ll^{\mathsf{C}}_{\{x,y\}} faa) = \{\{x \mapsto a,\, y \mapsto a\}\}$. Hence, the number of solutions decreases, but we do have

$$\{\sigma_1, \sigma_2\} \rightrightarrows \{\{x \mapsto a,\, y \mapsto a\}\}.$$

Example 4.21. Let P and Q be respectively the terms $\hat{x} + \hat{y}$ and $fa + fW$ for some W. Then $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\{x,y\}} Q)$ consists of the following substitutions:

$$\begin{aligned}
&\sigma_1 = \{\{x \mapsto fa, && y \mapsto fW\}\},\\
&\sigma_2 = \{\{x \mapsto fa, && y \mapsto fa + fW\}\},\\
&\sigma_3 = \{\{x \mapsto fW, && y \mapsto fa\}\},\\
&\sigma_4 = \{\{x \mapsto fW, && y \mapsto fa + fW\}\},\\
&\sigma_5 = \{\{x \mapsto fa + fW, && y \mapsto fa\}\},\\
&\sigma_6 = \{\{x \mapsto fa + fW, && y \mapsto fW\}\},\\
&\sigma_7 = \{\{x \mapsto fa + fW, && y \mapsto fa + fW\}\}.
\end{aligned}$$

Let first W be a variable z and let $\vartheta = \{z \mapsto a\}$. Then $\mathsf{solve}_{\mathsf{C}}((P\vartheta \ll^{\mathsf{C}}_{\{x,y\}} Q\vartheta) = \mathsf{solve}_{\mathsf{C}}(\hat{x} + \hat{y} \ll^{\mathsf{C}}_{\{x,y\}} fa) = \{\{x \mapsto fa,\, y \mapsto fa\}\}$. Hence, the number of solutions decreases, but we do have

$$\{(\sigma_i\vartheta)|_{\{x,y\}} \mid 1 \leq i \leq 7\} = \{\{x \mapsto fa,\, y \mapsto fa\}\}.$$

Let now W be $(\lambda_{\{z\}}\hat{z}\cdot z)a$. Then $P \rightrightarrows P' = P$, $Q \rightrightarrows Q' = fa$ and we get $\mathsf{solve}_{\mathsf{C}}((P' \ll^{\mathsf{C}}_{\{x,y\}} Q') = \mathsf{solve}_{\mathsf{C}}(\hat{x} + \hat{y} \ll^{\mathsf{C}}_{\{x,y\}} fa) = \{\{x \mapsto fa,\, y \mapsto fa\}\}$. Hence, the number of solutions decreases, but we do have

$$\{\sigma_1, \ldots, \sigma_7\} \rightrightarrows \{\{x \mapsto fa,\, y \mapsto fa\}\}.$$

Example 4.22. Let $P = fab$ and $Q = fcz$ for a commutative f. If $\uplus$ is strict, then $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\{x\}} Q)$ is undefined, because we have a derivation $\{\!\{fab \ll^{\mathsf{C}}_{\{x\}} fcz\}\!\} \rightsquigarrow \{\!\{fa \ll^{\mathsf{C}}_{\{x\}} fc\}\!\} \uplus \{\!\{b \ll^{\mathsf{C}}_{\{x\}} z\}\!\} \rightsquigarrow^{+} \mathbf{fail} \uplus \mathbf{wait} = \mathbf{wait}$. If $\uplus$ is non-strict, then the above derivation ends with **fail** and so does the other one: $\{\!\{fab \ll^{\mathsf{C}}_{\{x\}} fcz\}\!\} \rightsquigarrow \{\!\{fa \ll^{\mathsf{C}}_{\{x\}} fz\}\!\} \uplus \{\!\{b \ll^{\mathsf{C}}_{\{x\}} c\}\!\} \rightsquigarrow^{+} \mathbf{wait} \uplus \mathbf{fail} = \mathbf{fail}$. Hence, in this case $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\{x\}} Q) = \emptyset$.

If z is instantiated by a term that is not a matchable-sum, then $\mathsf{solve_C}$ remains undefined for the strict $\uplus$ and empty for the non-strict one.

If z is instantiated by a matchable-sum, them $\mathsf{solve_C}$ is the empty set for both strict and non-strict $\uplus$.

Example 4.23. If $P = f\hat{x}(ga)$ and $Q = f(ga)(hz)$ for a commutative f, then with strict $\uplus$, $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\{x\}} Q)$ is undefined due to $\{\!\{ga \ll^{\mathsf{C}}_{\{x\}} hz\}\!\}$, which will lead to **wait**. With non-strict $\uplus$, the same $\{\!\{ga \ll^{\mathsf{C}}_{\{x\}} hz\}\!\}$ will go to **fail** and we get $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\{x\}} Q) = \{\{x \mapsto hz\}\}$. Replacing z with b gives $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\{x\}} Q) = \{\{x \mapsto hb\}\}$ for both strict and non-strict $\uplus$.

Example 4.24. Let $P = f\hat{x}\hat{x}W$ and $Q = faab$ for a commutative f, where W is not a matchable-sum. Assume $\uplus$ is strict. Then $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\{x\}} Q)$ is undefined, because we have a derivation, where $W \ll^{\mathsf{C}}_{\{x\}} a$ appears and it ends with **wait**. If $\uplus$ is non-strict, then there are six derivations altogether and substitutions with non-disjoint domains appear in each of them: two contain $\{\!\{x \mapsto a\}\!\} \uplus \{\!\{x \mapsto a\}\!\}$, two other $\{\!\{x \mapsto a\}\!\} \uplus \{\!\{x \mapsto b\}\!\}$, and two more $\{\!\{x \mapsto b\}\!\} \uplus \{\!\{x \mapsto a\}\!\}$. Due to non-strictness, all these derivations end with **fail** and, therefore, $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\{x\}} Q) = \emptyset$.

If W reduces (or instantiates) into a matchable-sum form, then $\mathsf{solve_C}$ is the empty set both for strict and non-strict disjoint union.

The following lemma plays the crucial role in proving that $\mathsf{solve_C}$ makes the calculus confluent.

Lemma 4.25. *Let P and Q be two terms and χ be a finite set of variables such that $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\chi} Q)$ is defined. Then:*

1. *If ϑ is a substitution such that $\mathtt{fm}(\vartheta) \cap \hat{\chi} = \emptyset$, then $\mathsf{solve_C}(P\vartheta \ll^{\mathsf{C}}_{\chi} Q\vartheta)$ is defined and $\sigma \in \mathsf{solve_C}(P \ll^{\mathsf{C}}_{\chi} Q)$ iff $(\sigma\vartheta)|_{\chi} \in \mathsf{solve_C}(P\vartheta \ll^{\mathsf{C}}_{\chi} Q\vartheta)$.*
2. *If P' and Q' are terms with $P \rightrightarrows P'$ and $Q \rightrightarrows Q'$, then $\mathsf{solve_C}(P' \ll^{\mathsf{C}}_{\chi} Q')$ is defined and $\mathsf{solve_C}(P \ll^{\mathsf{C}}_{\chi} Q) \rightrightarrows \mathsf{solve_C}(P' \ll^{\mathsf{C}}_{\chi} Q')$.*

Proof. Proofs of both statements are very similar to each other. The first one is a bit simpler than the second one. Therefore, we present the latter below.

We proceed by induction on the term structure for P. We do not consider the case when P is a variable, because it would make $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q)$ undefined, which contradicts our assumption.

P is a function symbol f. If $Q \approx f$, then $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q) = \{Id\}$, $P \rightrightarrows f = P'$, $Q \rightrightarrows f = Q'$, and $\mathsf{solve}_{\mathsf{C}}(P' \ll^{\mathsf{C}}_{\chi} Q') = \{Id\}$. If $Q \not\approx f$, then Q must be an $\approx_{\mathsf{C}}$-rigid matchable-sum form, since $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q)$ is defined. In this case $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q) = \emptyset$. By Corollary 4.9, Q' is an $\approx_{\mathsf{C}}$-rigid matchable-sum form. Besides, from reduction of matchable-sum forms we can observe that when $Q \not\approx f$ and $Q \rightrightarrows Q'$, then $Q' \not\approx f$. Hence, Q' is an $\approx_{\mathsf{C}}$-rigid matchable-sum, different from f. Therefore, only the Fail rule applies and $\mathsf{solve}_{\mathsf{C}}(P' \ll^{\mathsf{C}}_{\chi} Q') = \emptyset$.

P is a matchable $\hat{x}$. If $x \in \chi$, then $\mathsf{solve}_{\mathsf{C}}(\hat{x} \ll^{\mathsf{C}}_{\chi} Q) = \{\{x \mapsto Q\}\}$ for any Q. By definition of parallel reduction, $P = \hat{x} \rightrightarrows P' = \hat{x}$. Therefore $\mathsf{solve}_{\mathsf{C}}(P' \ll^{\mathsf{C}}_{\chi} Q') = \{\{x \mapsto Q'\}\}$ for any $Q \rightrightarrows Q'$. The case when $x \notin \chi$ can be proved similarly to the case $P = f$ above.

P is an abstraction. Since $\mathsf{solve}_{\mathsf{C}}$ is defined, Q is a matchable-sum form and $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q) = \emptyset$. By definition of parallel reduction, an abstraction can be reduced to a finite sum of abstractions $P' = P'_1 + \cdots + P'_m$, $m \geq 1$, where each P' is an abstraction. Q' is a matchable-sum form. When $m = 1$, the matching problem $\{\!\{P' \ll^{\mathsf{C}}_{\chi} Q'\}\!\}$ can be transformed by Fail rule (hence, defined) and therefore $\mathsf{solve}_{\mathsf{C}}(P' \ll^{\mathsf{C}}_{\chi} Q') = \emptyset$. When $m > 1$, the matching problem $\{\!\{P' \ll^{\mathsf{C}}_{\chi} Q'\}\!\}$ is transformed by $\mathsf{Dec\text{-}Sum}$ rule to a set of matching problems, where the left hand side of each matching equation is an abstraction. Therefore, Fail rule applies to each problem, which implies that $\mathsf{solve}_{\mathsf{C}}(P' \ll^{\mathsf{C}}_{\chi} Q')$ is defined and, moreover, $\mathsf{solve}_{\mathsf{C}}(P' \ll^{\mathsf{C}}_{\chi} Q') = \emptyset$.

P is an application. Since $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q)$ is defined, both P and Q are matchable forms.

First, assume Q is not an application. In this case, $\{\!\{P \ll^{\mathsf{C}}_{\chi} Q\}\!\}$ can be transformed by the Fail rule only (since $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q)$ is defined), and $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q) = \emptyset$. When P is a matchable form and application, P is actually a data structure. Therefore, Q must be an $\approx_{\mathsf{C}}$-rigid matchable-sum form in order the Fail rule to apply. Let $P \rightrightarrows P'$ and $Q \rightrightarrows Q'$. By

Corollary 4.4, P' is again a data structure. By Corollary 4.9, Q' is an $\approx_{\mathsf{C}}$-rigid matchable-sum form. Therefore, only the Fail rule applies and $\mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = \emptyset$.

Now assume Q is an application. Hence, P and Q have forms $P = P_1P_2$ and $Q = Q_1Q_2$. Let $\chi_1 = \mathtt{fv}(P_1)\cup(\chi\setminus(\mathtt{fv}(P_2))$ and $\chi_2 = \mathtt{fm}(P_2)\cup(\chi\setminus(\mathtt{fv}(P_1))$ be two sets of matchables. Obviously, $\chi_1 \cup \chi_2 = \chi$. We distinguish between the cases whether $head(Q)$ is a commutative symbol or not.

Case 1. $head(Q)$ is not a commutative symbol. Then Dec-App applies to $P \ll_{\chi}^{\mathsf{C}} Q$ and by definition of $\mathsf{solve}_{\mathsf{C}}$, we can conclude $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) = (\mathsf{solve}_{\mathsf{C}}(P_1 \ll_{\chi_1}^{\mathsf{C}} Q_1)\uplus\mathsf{solve}_{\mathsf{C}}(P_2 \ll_{\chi_2}^{\mathsf{C}} Q_2))\setminus\{\mathbf{fail}\}$. Besides, $\mathsf{solve}_{\mathsf{C}}(P_1 \ll_{\chi_1}^{\mathsf{C}} Q_1)$ and $\mathsf{solve}_{\mathsf{C}}(P_2 \ll_{\chi_2}^{\mathsf{C}} Q_2))$ are defined. Let $P \rightrightarrows P'$ and $Q \rightrightarrows Q'$. By definition of matchable forms and parallel reduction, we have $P' = P_1'P_2'$ and $Q' = Q_1'Q_2'$ with $P_1 \rightrightarrows P_1'$, $P_2 \rightrightarrows P_2'$, $Q_1 \rightrightarrows Q_1'$ and $Q_2 \rightrightarrows Q_2'$. Moreover, since reduction does not affect the heads of matchable forms, $head(Q')$ is not a commutative symbol and again Dec-App applies to $P' \ll_{\chi}^{\mathsf{C}} Q'$. Therefore, by definition of $\mathsf{solve}_{\mathsf{C}}$, we have $\mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = (\mathsf{solve}_{\mathsf{C}}(P_1' \ll_{\chi_1}^{\mathsf{C}} Q_1') \uplus \mathsf{solve}_{\mathsf{C}}(P_2' \ll_{\chi_2}^{\mathsf{C}} Q_2')) \setminus \{\mathbf{fail}\}$. The induction hypothesis (IH) gives that $\mathsf{solve}_{\mathsf{C}}(P_i' \ll_{\chi_i}^{\mathsf{C}} Q_i')$, $i = 1, 2$, is defined and $\mathsf{solve}_{\mathsf{C}}(P_i \ll_{\chi_i}^{\mathsf{C}} Q_i) \rightrightarrows \mathsf{solve}_{\mathsf{C}}(P_i' \ll_{\chi_i}^{\mathsf{C}} Q_i')$. Then definedness of $\mathsf{solve}_{\mathsf{C}}(P_i' \ll_{\chi_i}^{\mathsf{C}} Q_i')$, $i = 1, 2$, implies that $\mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$ is also defined.

If $\chi_1 \cap \chi_2 \neq \emptyset$, then both $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) = \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = \emptyset$. Assume $\chi_1 \cap \chi_2 = \emptyset$. If $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) = \emptyset$, then $\mathsf{solve}_{\mathsf{C}}(P_1 \ll_{\chi}^{\mathsf{C}} Q_1) = \emptyset$ or $\mathsf{solve}_{\mathsf{C}}(P_2 \ll_{\chi}^{\mathsf{C}} Q_2) = \emptyset$. By the IH, we get $\mathsf{solve}_{\mathsf{C}}(P_1' \ll_{\chi}^{\mathsf{C}} Q_1') = \emptyset$ or $\mathsf{solve}_{\mathsf{C}}(P_2' \ll_{\chi}^{\mathsf{C}} Q_2') = \emptyset$, implying $\mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = \emptyset$. If $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) \neq \emptyset$, then we take $\sigma \in \mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q)$. It has a form of union $\sigma_1 \cup \sigma_2$, where $\sigma_1 \in \mathsf{solve}_{\mathsf{C}}(P_1 \ll_{\chi_1}^{\mathsf{C}} Q_1)$ and $\sigma_2 \in \mathsf{solve}_{\mathsf{C}}(P_2 \ll_{\chi_2}^{\mathsf{C}} Q_2)$. By Definition 3.4, there exist $\sigma_1' \in \mathsf{solve}_{\mathsf{C}}(P_1' \ll_{\chi_1}^{\mathsf{C}} Q_1')$ and $\sigma_2' \in \mathsf{solve}_{\mathsf{C}}(P_2' \ll_{\chi_2}^{\mathsf{C}} Q_2')$ such that $\sigma_1 \rightrightarrows \sigma_1'$ and $\sigma_2 \rightrightarrows \sigma_2'$. Because of the binding variable separation convention, all these substitutions are idempotent. We also have $dom(\sigma_1) = dom(\sigma_1') = \chi_1$, $dom(\sigma_2) \cup dom(\sigma_2') = \chi_2$. Since $\chi_1 \cap \chi_2 = \emptyset$, the disjoint union of these substitutions is just the union of their set representations. Therefore, if we take $\sigma' = \sigma_1' \cup \sigma_2' \in \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$, then $\sigma \rightrightarrows \sigma'$. In the same way we can prove that for each $\sigma' \in \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$ there exists $\sigma \in \mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q)$ such that $\sigma \rightrightarrows \sigma'$. By Definition 3.4, it implies $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) \rightrightarrows \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$.

Case 2. $head(Q)$ is a commutative symbol f, i.e., Q ha s a form $Q = fQ_1 \cdots Q_n$. If $head(P) \notin \chi \cup \{f\}$, then $head(P') \notin \chi \cup \{f\}$ and $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) = \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = \emptyset$.

Assume $head(P) = \hat{x}$ for some $x \in \chi$, i.e., P has a form $P = \hat{x}P_1 \cdots P_m$. Assume also that $n \geq m > 0$. (Otherwise $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) = \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = \emptyset$.) Then the $\mathsf{ME\text{-}C}$ rule applies to $P \ll_{\chi}^{\mathsf{C}} Q$. Let χ_i, $1 \leq i \leq m$, be the set of matchables defined as

$$\chi_i := \mathtt{fm}(P_i) \cup \left(\chi \setminus \bigcup_{1 \leq j \leq m,\, i \neq j} \mathtt{fm}(P_j)\right).$$

Clearly $\{\hat{x}\} \cup \bigcup_{i=1}^{m} \chi_i = \chi$. For each permutation π of $(1, \ldots, n)$, the solution set of equations obtained by the $\mathsf{ME\text{-}C}$ rule is

$$\left(\sigma_\pi \uplus \biguplus_{i=1}^{m} \mathsf{solve}_{\mathsf{C}}(P_i \ll_{\chi_i}^{\mathsf{C}} Q_{\pi(n-m+i)})\right) \setminus \{\mathbf{fail}\},$$

where $\sigma_\pi = \{x \mapsto fQ_{\pi(1)} \cdots Q_{\pi(n-m)}\}$. Since $\mathsf{ME\text{-}C}$ is the only rule that applies to $P \ll_{\chi}^{\mathsf{C}} Q$, $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q)$ is the union of these sets, i.e.,

$$\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) = \mathsf{solve}_{\mathsf{C}}(\hat{x}P_1 \cdots P_m \ll_{\chi}^{\mathsf{C}} fQ_1 \cdots Q_n) =$$
$$\bigcup_{\pi \in \Pi(1,\ldots,n)} \left(\left(\sigma_\pi \uplus \biguplus_{i=1}^{m} \mathsf{solve}_{\mathsf{C}}(P_i \ll_{\chi_i}^{\mathsf{C}} Q_{\pi(n-m+i)})\right) \setminus \{\mathbf{fail}\}\right),$$

where $\Pi(1, \ldots, n)$ is the set of all permutations of $(1, \ldots, n)$. From definedness of $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q)$ it follows that each $\mathsf{solve}_{\mathsf{C}}(P_i \ll_{\chi_i}^{\mathsf{C}} Q_{\pi(n-m+i)})$ is defined for all $1 \leq i \leq m$.

Since P and Q are matchable forms, by definition of parallel reduction we have $P = \hat{x}P_1 \cdots P_m \rightrightarrows \hat{x}P'_1 \cdots P'_m = P'$, where $P_i \rightrightarrows P'_i$ for $1 \leq i \leq m$, and $Q = fQ_1 \cdots Q_n \rightrightarrows fQ'_1 \cdots Q'_n = Q'$, where $Q_i \rightrightarrows Q'_i$ for $1 \leq i \leq n$. Then for $\mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$ we get

$$\mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = \mathsf{solve}_{\mathsf{C}}(\hat{x}P'_1 \cdots P'_m \ll_{\chi}^{\mathsf{C}} fQ'_1 \cdots Q'_n) =$$
$$\bigcup_{\pi \in \Pi(1,\ldots,n)} \left(\left(\sigma'_\pi \uplus \biguplus_{i=1}^{m} \mathsf{solve}_{\mathsf{C}}(P'_i \ll_{\chi_i}^{\mathsf{C}} Q'_{\pi(n-m+i)})\right) \setminus \{\mathbf{fail}\}\right),$$

where $\sigma'_\pi = \{x \mapsto fQ'_{\pi(1)} \cdots Q'_{\pi(n-m)}\}$ and Π is the same as above. Obviously, we have $\sigma_\pi \rightrightarrows \sigma'_\pi$. Besides, by the IH, $\mathsf{solve}_{\mathsf{C}}(P'_i \ll_{\chi_i}^{\mathsf{C}} Q'_{\pi(n-m+i)})$ is defined for all $1 \leq i \leq m$, which implies definedness of $\mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$. The IH also gives $\mathsf{solve}_{\mathsf{C}}(P_i \ll_{\chi_i}^{\mathsf{C}} Q_{\pi(n-m+i)}) \rightrightarrows \mathsf{solve}_{\mathsf{C}}(P'_i \ll_{\chi_i}^{\mathsf{C}} Q'_{\pi(n-m+i)})$.

Assume all χ_i's are pairwise disjoint and do not contain $\hat{x}$. (Otherwise, by Lemma 4.11 and definedness of $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q)$ and $\mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$ we have $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) = \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = \emptyset$ and the lemma holds.) Take $\vartheta \in \mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q)$. It must have a form $\sigma_\pi \cup \sigma_1 \cup \cdots \cup \sigma_m$, where $\sigma_i \in \mathsf{solve}_{\mathsf{C}}(P_i \ll_{\chi_i}^{\mathsf{C}} Q_{\pi(n-m+i)})$. By the IH, we have $\sigma_i' \in \mathsf{solve}_{\mathsf{C}}(P_i' \ll_{\chi_i}^{\mathsf{C}} Q'_{\pi(n-m+i)})$ such that $\sigma_i \rightrightarrows \sigma_i'$. Take $\vartheta' = \sigma_\pi' \cup \sigma_1' \cup \cdots \cup \sigma_m' \in \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$. Then $\sigma_\pi \rightrightarrows \sigma_\pi'$ and $\sigma_i \rightrightarrows \sigma_i'$ by Definition 3.2 imply $\vartheta \rightrightarrows \vartheta'$. With the same reasoning we can prove that for each $\vartheta' \in \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$ there exists $\vartheta \in \mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q)$ such that $\vartheta \rightrightarrows \vartheta'$. Hence, $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) \rightrightarrows \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q')$

The case when $P = fP_1 \cdots P_m$ and Q is $fQ_1 \cdots Q_n$ for a commutative f can be proved similarly.

P is a sum. Assume P has a form $P_1+\cdots+P_m$, $m > 1$. Since $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q)$ is defined, both P and Q are matchable-sums.

If at least one P_i is an abstraction, then it is easy to see that in this case $\mathsf{solve}_{\mathsf{C}}(P \ll_{\chi}^{\mathsf{C}} Q) = \mathsf{solve}_{\mathsf{C}}(P' \ll_{\chi}^{\mathsf{C}} Q') = \emptyset$. Below we assume no P_i is an abstraction, i.e., all P_i's are data structures. The proof proceeds by distinguishing two cases depending on the form of Q.

<u>Case 1.</u> $Q = Q_1+\cdots+Q_n$, $n \geq 1$, and all Q_i's are data structures. Let Θ be the set of m-tuples of non-empty subsets of $Q_1, \ldots, Q_n$, defined as follows:

$$\Theta := \{\langle \mathcal{Q}_1, \ldots, \mathcal{Q}_m \rangle \mid \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_m = \{Q_1, \ldots, Q_n\} \text{ and for all } 1 \leq i \leq m,\ \mathcal{Q}_i \neq \emptyset\}.$$

Then $P \ll_{\chi}^{\mathsf{C}} Q$ is rewritten by $\mathsf{Dec\text{-}Sum}$ rule, giving m new equations $P_i \ll_{\chi}^{\mathsf{C}} \sum_{N \in \mathcal{Q}_i} N$, $1 \leq i \leq m$. Such equations are obtained for each possible selection $\langle \mathcal{Q}_1, \ldots, \mathcal{Q}_m \rangle \in \Theta$.

Let χ_i, $1 \leq i \leq m$, be the set of matchables defined as

$$\chi_i := \mathtt{fm}(P_i) \cup \left(\chi \setminus \bigcup_{1 \leq j \leq m,\, i \neq j} \mathtt{fm}(P_j)\right).$$

Since $\mathsf{Dec\text{-}Sum}$ is the only matching rule that can be used for $\{\!\{P \ll_{\chi}^{\mathsf{C}} Q\}\!\}$,

$\mathsf{solve}_\mathsf{C}(P \ll^\mathsf{C}_\chi Q)$ is

$$\mathsf{solve}_\mathsf{C}(P \ll^\mathsf{C}_\chi Q) = \mathsf{solve}_\mathsf{C}\Big(\sum_{i=1}^m P_i \ll^\mathsf{C}_\chi \sum_{i=1}^n Q_i\Big) =$$

$$\bigcup_{\langle \mathcal{Q}_1,\dots,\mathcal{Q}_m\rangle\in\Theta} \Big(\Big(\biguplus_{i=1}^m \mathsf{solve}_\mathsf{C}(P_i \ll^\mathsf{C}_{\chi_i} \sum_{N\in\mathcal{Q}_i} N)\Big) \setminus \{\mathbf{fail}\}\Big).$$

Definedness of $\mathsf{solve}_\mathsf{C}(P \ll^\mathsf{C}_\chi Q)$ implies that $\mathsf{solve}_\mathsf{C}(P_i \ll^\mathsf{C}_{\chi_i} \sum_{N\in\mathcal{Q}_i} N)$ is defined for all $1 \le i \le m$ and $\langle \mathcal{Q}_1,\dots,\mathcal{Q}_m\rangle \in \Theta$.

Now let $P_i \rightrightarrows P'_i$, $Q_1 \rightrightarrows Q'_1,\dots,Q_n \rightrightarrows Q'_n$, and Θ' be the set

$$\Theta' := \{\langle \mathcal{Q}'_1,\dots,\mathcal{Q}'_m\rangle \mid \mathcal{Q}'_1 \cup\cdots\cup \mathcal{Q}'_m = \{Q'_1,\dots,Q'_n\}$$
$$\text{and for all } 1 \le i \le m,\ \mathcal{Q}'_i \neq \emptyset\}.$$

By Corollary 4.4, $P' = P'_1 + \cdots + P'_m$ is a matchable-sum form and $Q' = Q'_1 + \cdots + Q'_n$ is a sum of data structures.[5] Then Dec-Sum is the only rule that can be used for $\{\!\{P' \ll^\mathsf{C}_\chi Q'\}\!\}$ and $\mathsf{solve}_\mathsf{C}(P' \ll^\mathsf{C}_\chi Q')$ is

$$\mathsf{solve}_\mathsf{C}(P' \ll^\mathsf{C}_\chi Q') = \mathsf{solve}_\mathsf{C}\Big(\sum_{i=1}^m P'_i \ll^\mathsf{C}_\chi \sum_{i=1}^n Q'_i\Big) =$$

$$\bigcup_{\langle \mathcal{Q}'_1,\dots,\mathcal{Q}'_m\rangle\in\Theta'} \Big(\Big(\biguplus_{i=1}^m \mathsf{solve}_\mathsf{C}(P'_i \ll^\mathsf{C}_{\chi_i} \sum_{N'\in\mathcal{Q}'_i} N')\Big) \setminus \{\mathbf{fail}\}\Big).$$

By the IH, $\mathsf{solve}_\mathsf{C}(P'_i \ll^\mathsf{C}_{\chi_i} \sum_{N'\in\mathcal{Q}'_i} N')$ is defined for all $1 \le i \le m$ and $\langle \mathcal{Q}'_1,\dots,\mathcal{Q}'_m\rangle \in \Theta'$, and $\mathsf{solve}_\mathsf{C}(P_i \ll^\mathsf{C}_{\chi_i} \sum_{N\in\mathcal{Q}_i} N) \rightrightarrows \langle \mathcal{Q}'_1,\dots,\mathcal{Q}'_m\rangle \in \Theta'$ holds. With the same reasoning as in the previous case (when P was an application) we can conclude that $\mathsf{solve}_\mathsf{C}(P' \ll^\mathsf{C}_\chi Q')$ is defined and $\mathsf{solve}_\mathsf{C}(P \ll^\mathsf{C}_\chi Q) \rightrightarrows \mathsf{solve}_\mathsf{C}(P' \ll^\mathsf{C}_\chi Q')$.

Case 2. $Q = Q_1 + \cdots + Q_n$, $n \ge 1$, and at least one Q_i is not a data structure, i.e., it is an abstraction, since Q is a matchable-sum form. Hence, Q is actually an $\approx_\mathsf{C}$-rigid matchable-sum form and $\mathsf{solve}_\mathsf{C}(P \ll^\mathsf{C}_\chi Q) = \emptyset$. By Corollary 4.4, P' is a matchable-sum. By Corollary 4.9, Q' is also an $\approx_\mathsf{C}$-rigid matchable-sum form, containing at least one abstraction as a summand. It implies that $\mathsf{solve}_\mathsf{C}(P' \ll^\mathsf{C}_\chi Q')$ is defined and $\mathsf{solve}_\mathsf{C}(P' \ll^\mathsf{C}_\chi Q') = \emptyset$. □

[5] Because of idempotence of $+$, the number of distinct summands in the sums of data structures $P'_1 + \cdots + P'_m$ and $Q'_1 + \cdots + Q'_n$ can be smaller than m and n, respectively, but it does not affect the reasoning.

The lemma we just proved is valid for both strict and non-strict $\uplus$. Its proof did not depend on these properties.

Now we formulate the main result of this section, stating that this instance of solve satisfies $\mathbf{C}_1$ and $\mathbf{C}_2$ (and, thus, makes the calculus confluent):

Theorem 4.26. *The function* $\mathsf{solve}_\mathsf{C}$ *satisfies* $\mathbf{C}_1$ *and* $\mathbf{C}_2$*.*

Proof. When $\mathsf{solve}_\mathsf{C}(P \ll^{\mathsf{C}}_{\chi} Q)$ is undefined, the conditions $\mathbf{C}_1$ and $\mathbf{C}_2$ trivially hold. Therefore, we assume $\mathsf{solve}_\mathsf{C}(P \ll^{\mathsf{C}}_{\chi} Q)$ is defined.

Proving $\mathbf{C}_1$*.* Let ϑ be a substitution such that $\mathtt{fm}(\vartheta) \cap \hat{\chi} = \emptyset$. Then $\mathbf{C}_1$ follows from the fact that $\sigma \in \mathsf{solve}_\mathsf{C}(P \ll^{\mathsf{C}}_{\chi} Q)$ iff $(\sigma\vartheta)|_{\chi} \in \mathsf{solve}_\mathsf{C}(P\vartheta \ll^{\mathsf{C}}_{\chi} Q\vartheta)$, which is statement 1 of Lemma 4.25.

Proving $\mathbf{C}_2$*.* Let $P \rightrightarrows P'$ and $Q \rightrightarrows Q'$. Then $\mathbf{C}_2$ follows from the fact that Let $\mathsf{solve}_\mathsf{C}(P \ll^{\mathsf{C}}_{\chi} Q) \rightrightarrows \mathsf{solve}_\mathsf{C}(P' \ll^{\mathsf{C}}_{\chi} Q')$, which is statement 2 of Lemma 4.25. $\square$

This theorem is valid for both strict and non-strict $\uplus$. The following corollary immediately follows from theorems 4.26 and 3.9:

Corollary 4.27. *The finitary pattern calculus that uses* $\mathsf{solve}_\mathsf{C}$ *as the pattern matching function is confluent.*

4.2. An Instance of solve *with Polymorphic Commutative Matching*

The solving function considered in the previous section does not permit to do reduction by a *generic* commutative symbol: One needs to know explicitly which function symbol is commutative. However, sometimes it is desirable to abstract from such concrete function symbols and do matching/reduction in the abstract version, bringing in the concrete function symbol only when necessary. In this section, we illustrate how this can be done on the example of commutative matching, obtaining yet another extension of the *pure pattern calculus with matchable symbols* to finitary matching.

First, we start with identifying a subset of variables (and, consequently, matchables) that are restricted to be instantiated (respectively, bound) to single symbols only, instead of arbitrary terms. Hence, if x is such a variable, then we would like it to be instantiated by (respectively, we want $\hat{x}$ to match) a function symbol, a variable, or a matchable, but not by a more complex term such as, e.g., fx, $\lambda_{\{v\}}\hat{v}.v$, or $x + y$. We call this set the set of *single symbol variables*, denoted by $\mathcal{S}$ (resp. *single symbol matchables*, denoted by

$\hat{\mathcal{S}}$) and assume them to be infinite, to have "enough" such variables and matchables for α-renaming.

Note that once we have identified $\mathcal{S}$, there is no need in function symbols anymore. The elements of $\hat{\mathcal{S}}$ can play their role, whenever needed. Therefore, we do not consider function symbols (except $+$) in this section.

The set $\mathcal{S}$ can be split further into disjoint infinite subsets for each equational theory we would like to consider, plus an infinite subset for syntactic symbols (syntactic in the sense of unification theory, with no equational axiom associated to them). In this section we consider only two subsets: $\mathcal{S}_C$ for the commutative theory and $\mathcal{S}_S$ for syntactic symbols. Respectively, we will have two sets of matchables: $\hat{\mathcal{S}_C}$ and $\hat{\mathcal{S}_S}$. The meta-symbols c and $\hat{c}$ will be used for the elements of $\mathcal{S}_C$ and $\hat{\mathcal{S}_C}$, respectively. We will write g and $\hat{g}$ for the elements of $\mathcal{S}_S$ and $\hat{\mathcal{S}_S}$, respectively. The letters x, y, z, v and their hatted counterparts are still reserved for arbitrary variables and matchables.

This distinction between unrestricted variables and single symbol variables should be reflected in substitutions as well. We assume that substitutions map variables from $\mathcal{S}_C$ to symbols from $\mathcal{S}_C \cup \hat{\mathcal{S}_C}$, and variables from $\mathcal{S}_S$ to symbols from $\mathcal{S}_S \cup \hat{\mathcal{S}_S}$, while unrestricted variables, as before, are mapped to arbitrary terms (including commutative and syntactic symbols, since they are terms too).

Now, the equational theory for polymorphic commutativity is specified by the axiom schemata $\hat{c}x_1 \cdots x_n = \hat{c}x_{\pi(1)} \cdots x_{\pi(n)}$, $n \geq 2$, for each $\hat{c} \in \hat{\mathcal{S}_C}$ and non-identity permutation π of $(1, \dots, n)$. We denote this theory by PC.

We can easily modify the rules for $\mathsf{solve}_{\mathsf{C}}$ to obtain the ones that define the solving function $\mathsf{solve}_{\mathsf{PC}}$ for polymorphic commutativity. From the commutative matching algorithm in Section 4.1, we take Del-M, ME, Dec-App, Dec-Sum, Fail and Wait unchanged, and, in addition, introduce the following new rules for commutative symbols:

ME-CS: $\{\!\{\hat{x}P_1 \cdots P_m \ll_{\chi}^{\mathsf{PC}} MQ_1 \cdots Q_n\}\!\} \rightsquigarrow \vartheta \uplus \biguplus_{i=1}^{m} \{\!\{P_1 \ll_{\chi}^{\mathsf{PC}} Q_{\pi(n-m+i)}\}\!\}$, where $x \in \chi$, $M \in \mathcal{S}_C \cup \hat{\mathcal{S}_C}$, $n \geq m > 0$, π is a permutation of $(1, \dots, n)$, and $\vartheta = \{x \mapsto MQ_{\pi(1)} \cdots Q_{\pi(n-m)}\}$.

Dec-CS: $\{\!\{\hat{c}\, P_1 \cdots P_n \ll_{\chi}^{\mathsf{PC}} \hat{c}\, Q_1 \cdots Q_n\}\!\} \rightsquigarrow \biguplus_{i=1}^{n} \{\!\{P_i \ll_{\chi}^{\mathsf{PC}} Q_{\pi(i)}\}\!\}$, where $c \notin \chi$, $n > 0$, and π is a permutation of $(1, \dots, n)$.

These rules have quite intuitive explanations: ME-CS is almost the same as the ME-C rule from $\mathsf{solve}_{\mathsf{C}}$, with the only difference that the head of the right side is a commutative variable or matchable symbol, instead of a commutative function symbol. Dec-CS is the counterpart of the Dec-C rule from $\mathsf{solve}_{\mathsf{C}}$, but instead of commutative function symbol, here we have a commutative matchable symbol which is not bound in χ. However, one should take into account that now $\hat{x}$ can also be from $\hat{\mathcal{S}_C}$ or from $\hat{\mathcal{S}_S}$. If $\hat{x} \in \hat{\mathcal{S}_C}$, the rule succeeds only when $m = n$. If $\hat{x} \in \hat{\mathcal{S}_S}$, then the rule does not apply even if $m = n$, since x can not be mapped to $M \in \mathcal{S}_C \cup \hat{\mathcal{S}_C}$. All these follow from the assumption that the substitution ϑ in ME-CS is defined.

It should be also noted that although the ME rule is taken unchanged, the new definition of substitution makes sure that when in the left hand side of the rule we have $\hat{x} \in \hat{\mathcal{S}_C}$ (resp. $\hat{x} \in \hat{\mathcal{S}_S}$), the rule succeeds only if $M \in \mathcal{S}_C \cup \hat{\mathcal{S}_C}$ (resp. $M \in \mathcal{S}_S \cup \hat{\mathcal{S}_S}$).

Example 4.28. $\mathsf{solve}_{\mathsf{PC}}(\hat{c}_1\hat{x}\hat{y} \ll^{\mathsf{PC}}_{\{c_2,x,y\}} c_2ab) = \{\{c_1 \mapsto c_2, x \mapsto a, y \mapsto b\}, \{c_1 \mapsto c_2, x \mapsto b, y \mapsto a\}\}$. The free variable $c_2 \in \mathcal{S}_C$ can be instantiated only by a symbol from $\mathcal{S}_C \cup \hat{\mathcal{S}_C}$. If ϑ is such a substitution, $\mathsf{solve}_{\mathsf{PC}}(\hat{c}_1\hat{x}\hat{y} \ll^{\mathsf{PC}}_{\{c_2,x,y\}} c_2\vartheta ab) = \{\{c_1 \mapsto c_2\vartheta, x \mapsto a, y \mapsto b\}, \{c_1 \mapsto c_2\vartheta, x \mapsto b, y \mapsto a\}\}$.

Example 4.29. $\mathsf{solve}_{\mathsf{PC}}(\hat{x}\hat{y} \ll^{\mathsf{PC}}_{\{x,y\}} cab) = \{\{x \mapsto ca, y \mapsto b\}, \{x \mapsto cb, y \mapsto a\}\}$. The free variable $c \in \mathcal{S}_C$ can be instantiated only by a symbol from $\mathcal{S}_C \cup \hat{\mathcal{S}_C}$. If ϑ is such a substitution, $\mathsf{solve}_{\mathsf{PC}}(\hat{x}\hat{y} \ll^{\mathsf{PC}}_{\{x,y\}} c\vartheta ab) = \{\{x \mapsto c\vartheta a, y \mapsto b\}, \{x \mapsto c\vartheta b, y \mapsto a\}\}$.

Example 4.30. $\mathsf{solve}_{\mathsf{PC}}(\hat{g}\hat{x}\hat{y} \ll^{\mathsf{PC}}_{\{g,x,y\}} cab) = \emptyset$, because the sides are matchable forms, but the syntactic matchable $\hat{g} \in \hat{\mathcal{S}_S}$ can not match the commutative variable c. The answer will remain the same for any instantiation of the right hand side, since c can be only instantiated by a symbol from $\mathcal{S}_C \cup \hat{\mathcal{S}_C}$.

Theorem 4.31. *The function* $\mathsf{solve}_{\mathsf{PC}}$ *satisfies* $\mathbf{C}_1$ *and* $\mathbf{C}_2$*.*

Proof. The theorem can be proved similarly to Theorem 4.26. We take into account that although free commutative or syntactic variables may appear in the active position in the right hand side of matching equations, it does not cause a problem because any instantiation replaces them only by single symbols. □

Corollary 4.32. *The finitary pattern calculus that uses* $\mathsf{solve}_{\mathsf{PC}}$ *as the pattern matching function is confluent.*

4.3. An Instance of solve *with Flat Patterns*

The equational theory for a flat function symbol f is specified by the axiom schema $f x_1 \cdots x_n\,(f y_1 \cdots y_m)\,z_1 \cdots z_k = f x_1 \cdots x_n\, y_1 \cdots y_m\, z_1 \cdots z_k$, $n, m, k \geq 0$. Essentially, it says that nested occurrences of the symbol f can be flattened out. The theory is quite similar to associativity. We denote it by F. We have, for instance, $\lambda_{\{x,y\}} f(g\hat{x})(f\hat{y})f.\, xy + faf(fc) + fafc \approx_{\mathsf{F}} fac + \lambda_{\{z,y\}} f(g\hat{z})\hat{y}.\, zy$, provided that f is flat.

Similar to the definition of $\approx_{\mathsf{C}}$-rigid matchable-sum form (Definition 4.6), we define $\approx_{\mathsf{F}}$-rigid matchable-sum form, which slightly differs from its commutative counterpart:

Definition 4.33. Let $\mathbf{M}_+ = \mathbf{M}_1 + \cdots + \mathbf{M}_m$ be a matchable-sum such that no two $\mathbf{M}_i$ and $\mathbf{M}_j$, $i \neq j$, are $\approx_{\mathsf{F}}$-equivalent. We say that $\mathbf{M}_+$ is $\approx_{\mathsf{F}}$-*rigid*, if at least one of the following conditions are fulfilled:

- $m = 1$ and if $\mathbf{M}_1 = f N_1 \cdots N_n$ for a flat f (in a flattened form), then each N_i, $1 \leq i \leq n$, is a data structure, or
- $\mathbf{M}_i$ is an abstraction for some $1 \leq i \leq m$, or
- $\mathbf{M}_i$ and $\mathbf{M}_j$ have different heads for some $1 \leq i, j \leq m$, or
- $\mathbf{M}_i$ and $\mathbf{M}_j$ are values for some $1 \leq i, j \leq m$, $i \neq j$.

Lemma 4.34. *$\approx_{\mathsf{F}}$-rigid matchable-sum forms are closed under substitution application and reduction.*

Proof. Let $\mathbf{M} = \mathbf{M}_1 + \cdots + \mathbf{M}_m$ be a $\approx_{\mathsf{F}}$-rigid matchable-sum and show that both $\mathbf{M}\vartheta$ (for some ϑ) and $\mathbf{M}'$ with $\mathbf{M} \multimap \mathbf{M}'$ are $\approx_{\mathsf{F}}$-rigid matchable-sums. The interesting case is $m = 1$. For the other cases of Definition 4.33 the proof is similar to Lemma 4.8.

Let $m = 1$. Assume $\mathbf{M}_1$ has a (flattened) form $f N_1 \cdots N_n$ for a flat f and data structures $N_1, \ldots, N_n$. (Otherwise the proof proceeds as in Lemma 4.8.) Then $\mathbf{M}\vartheta$ has again only one summand. By Lemma 4.3, $N_i\vartheta$ is again a data structure and $\mathit{head}(N_i\vartheta) = \mathit{head}(N_i) \neq f$ for all $1 \leq i \leq n$. Hence, by Definition 4.33, $\mathbf{M}\vartheta$ is $\approx_{\mathsf{F}}$-rigid. For $\mathbf{M}'$ the proof is similar. $\square$

From this lemma and Lemma 3.5, we have the following corollary:

Corollary 4.35. *$\approx_{\mathsf{F}}$-rigid matchable-sum forms are closed under parallel reduction.*

The rule-based flat matching algorithm we described in this section is inspired by the flat matching procedure from (Kutsia, 2008). The rules are given below. Note that Del-F, Del-M, ME, and Dec-Sum are the same as in commutative matching, Dec-App just differs from its commutative counterpart by replacing "commutative" with "flat" in the condition, and Wait is again the default rule. Rules are applied modulo $\approx$-equivalence. An important assumption is that all nested occurrences of flat symbols in matching equations are flattened.

Del-F: $\{\!\!\{ f \ll^{\mathsf{F}}_{\chi} f \}\!\!\} \rightsquigarrow \mathit{Id}$.

Del-M: $\{\!\!\{ \hat{x} \ll^{\mathsf{F}}_{\chi} \hat{x} \}\!\!\} \rightsquigarrow \mathit{Id}$, where $x \notin \chi$.

ME: $\{\!\!\{ \hat{x} \ll^{\mathsf{F}}_{\chi} Q \}\!\!\} \rightsquigarrow \{x \mapsto Q\}$, where $x \in \chi$.

ME-Flat-1: $\{\!\!\{ \hat{x} P_1 \cdots P_m \ll^{\mathsf{F}}_{\chi} f Q_1 \cdots Q_n \}\!\!\} \rightsquigarrow \vartheta \uplus \{\!\!\{ f P_1 \cdots P_m \ll^{\mathsf{F}}_{\chi} f Q_{i+1} \cdots Q_n \}\!\!\}$, where f is a flat function symbol, $Q_1, \ldots, Q_n$ are matchable-sum forms, $m > 0$, $n \geq i \geq 0$, $x \in \chi$, and $\vartheta = \{x \mapsto f Q_1 \cdots Q_i\}$.

ME-Flat-2: $\{\!\!\{ f\, P_1 \cdots P_m\, \hat{x} \ll^{\mathsf{F}}_{\chi} f\, Q_1 \cdots Q_n \}\!\!\} \rightsquigarrow \vartheta \uplus \{\!\!\{ f\, P_1 \cdots P_m \ll^{\mathsf{F}}_{\chi} f\, Q_1 \cdots Q_i \}\!\!\}$ where f is a flat function symbol, each of $Q_1, \ldots, Q_n$ is a matchable-sum form, $m \geq 0$, $n \geq i \geq 0$, $x \in \chi$, and $\vartheta = \{x \mapsto f Q_{i+1} \cdots Q_n\}$,

Dec-App: $\{\!\!\{ P_1 P_2 \ll^{\mathsf{F}}_{\chi} Q_1 Q_2 \}\!\!\} \rightsquigarrow \{\!\!\{ P_1 \ll^{\mathsf{F}}_{\chi} Q_1 \}\!\!\} \uplus \{\!\!\{ P_2 \ll^{\mathsf{F}}_{\chi} Q_2 \}\!\!\}$, where $P_1\, P_2$ and $Q_1\, Q_2$ are matchable forms and $\mathit{head}(Q_1 Q_2)$ is not a flat function symbol.

Dec-Flat: $\{\!\!\{ f\, P_1 \cdots P_m \ll^{\mathsf{F}}_{\chi} f\, Q_1 \cdots Q_n \}\!\!\} \rightsquigarrow \{\!\!\{ f\, P_1 \cdots P_{m-1} \ll^{\mathsf{F}}_{\chi} f\, Q_1 \cdots Q_{n-1} \}\!\!\} \uplus \{\!\!\{ P_m \ll^{\mathsf{F}}_{\chi} Q_n \}\!\!\}$, where f is a flat function symbol, $m, n \geq 1$, $P_m \notin \hat{\chi}$, and Q_n is a matchable-sum form.

Dec-Sum: $\{\!\!\{ P_1 + \cdots + P_m \ll^{\mathsf{F}}_{\chi} Q_1 + \cdots + Q_n \}\!\!\} \rightsquigarrow \biguplus_{i=1}^{m} \{\!\!\{ P_i \ll^{\mathsf{F}}_{\chi} \sum_{Q \in \mathcal{Q}_i} Q \}\!\!\}$, where $m > 1$, $n \geq 1$, $P_1, \ldots, P_m$ are matchable forms, $Q_1, \ldots, Q_n$ are data structures, and each $\mathcal{Q}_i$ is a nonempty subset of $\{Q_1, \ldots, Q_n\}$ such that $\mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_m = \{Q_1, \ldots, Q_n\}$.

Fail: $\{\!\!\{ P \ll^{\mathsf{F}}_{\chi} Q \}\!\!\} \rightsquigarrow$ **fail**, if P and Q are matchable-sum forms such that if P is a data structure, then both P and Q are $\approx_{\mathsf{F}}$-rigid matchable-sum forms, and no rule above is applicable.

Wait: $\{\!\!\{ P \ll^{\mathsf{F}}_{\chi} Q \}\!\!\} \rightsquigarrow$ **wait**, if no other rule applies.

The rules $\mathsf{ME\text{-}Flat\text{-}1}$, $\mathsf{ME\text{-}Flat\text{-}2}$, and $\mathsf{Dec\text{-}Sum}$ can transform the selected equation in multiple ways. No two rules apply to the same equation.

Definition 4.36. The instance of solve for a flat theory, denoted $\mathsf{solve}_\mathsf{F}$, is defined by the construction from Definition 4.2, based on the rules for flat matching as described above. Before $\mathsf{solve}_\mathsf{F}$ applies to P and Q, it is assumed that in P and Q all nested occurrences of $+$ and flat symbols are flattened out and $\approx_\mathsf{F}$-equivalent summands are identified.

Similarly to Lemma 4.11, we can prove that $\mathsf{solve}_\mathsf{F}(P \ll^\mathsf{F}_\chi Q) = \emptyset$ or $\mathsf{solve}_\mathsf{F}(P \ll^\mathsf{F}_\chi Q)$ is undefined for a nonlinear P.

Theorem 4.37. $\mathsf{solve}_\mathsf{F}$ *is terminating and sound.*

Proof. Termination follows from the fact that every rule except $\mathsf{ME\text{-}Flat\text{-}1}$ strictly decreases the multiset of sizes (number of symbols) of the left hand sides of matching equations. The rule $\mathsf{ME\text{-}Flat\text{-}1}$ does not increase that measure, but strictly decreases the number of occurrences of matchable symbols in the right hand sides of matching equations.

Soundness directly follows from Definition 4.2. □

The function $\mathsf{solve}_\mathsf{F}$ preserves $\approx_\mathsf{F}$-equivalence in the sense that $\approx_\mathsf{F}$ problems have $\approx_\mathsf{F}$-equivalent solutions. It follows from the fact that all nested occurrences of $+$ and flat symbols are flattened out and $\approx_\mathsf{F}$-equivalent summands are identified in the input, and the order of summands does not matter in the matching rules.

For terms without function symbols and $+$, $\mathsf{solve}_\mathsf{F}$ with non-strict $\uplus$ coincides to the compound matching algorithm of Jay and Kesner (2009). Hence, a pattern calculus with $\mathsf{solve}_\mathsf{F}$ as the matching function is yet another extension of the *pure pattern calculus with matchable symbols* to finitary matching.

Below we give several examples that illustrate $\mathsf{solve}_\mathsf{F}$.

Example 4.38. Due to the assumption that there are no nested occurrences of flat symbols, the algorithm gives $\mathsf{solve}_\mathsf{F}(\hat{x} f \ll^\mathsf{F}_{\{x\}} f) = \{\{x \mapsto f\}\}$ for a flat f. This is computed by the following rule applications:

$$\begin{aligned}
&\{\!\{\hat{x} f \ll^\mathsf{F}_{\{x\}} f\}\!\} \rightsquigarrow && \text{(by } \mathsf{ME\text{-}Flat\text{-}1} \text{ and flattening)}\\
&\{x \mapsto f\} \uplus \{\!\{f \ll^\mathsf{F}_{\{x\}} f\}\!\} \rightsquigarrow && \text{(by } \mathsf{Del\text{-}F}\text{)}\\
&\{x \mapsto f\} \uplus \mathit{Id} = \{x \mapsto f\}.
\end{aligned}$$

Example 4.39. $\mathsf{solve}_{\mathsf{F}}(f\hat{x}\hat{y} \ll^{\mathsf{F}}_{\{x,y\}} f((\lambda_{\{z\}}\hat{z}.faz)b)$, where f is flat, is undefined, because $(\lambda_{\{z\}}\hat{z}.faz)b$ is not a matchable form and $\mathsf{ME\text{-}Flat\text{-}2}$ does not apply. However, if we reduce it, then the obtained problem is solvable and there are six solutions: $\mathsf{solve}_{\mathsf{F}}(f\hat{x}\hat{y} \ll^{\mathsf{F}}_{\{x,y\}} fab) = \{\{x \mapsto f, y \mapsto fab\}, \{x \mapsto fa, y \mapsto fb\}, \{x \mapsto fab, y \mapsto f\}, \{x \mapsto a, y \mapsto b\}, \{x \mapsto fa, y \mapsto b\}, \{x \mapsto a, y \mapsto fb\}\}$.

Example 4.40. $\mathsf{solve}_{\mathsf{F}}(f\hat{x}\hat{y} \ll^{\mathsf{F}}_{\{x,y\}} fz)$, where f is flat, is undefined, because z is not a matchable form and $\mathsf{ME\text{-}Flat\text{-}2}$ rule does not apply. However, if we instantiate z by a, then the obtained problem is solvable and there are four solutions: $\mathsf{solve}_{\mathsf{F}}(f\hat{x}\hat{y} \ll^{\mathsf{F}}_{\{x,y\}} fa) = \{\{x \mapsto f, y \mapsto a\}, \{x \mapsto f, y \mapsto fa\}, \{x \mapsto a, y \mapsto f\}, \{x \mapsto fa, y \mapsto f\}\}$.

Example 4.41. $\mathsf{solve}_{\mathsf{F}}(f(\hat{x} + \hat{y}) \ll^{\mathsf{F}}_{\{x,y\}} f(a + b) = \{\{x \mapsto a, y \mapsto b\}, \{x \mapsto b, y \mapsto a\}, \{x \mapsto a + b, y \mapsto b\}, \{x \mapsto a + b, y \mapsto a\}, \{x \mapsto a, y \mapsto a + b\}, \{x \mapsto b, y \mapsto a+b\}, \{x \mapsto a+b, y \mapsto a+b\}\}$. However, if f is flat, there is one more substitution that would solve the problem, but the flat matching algorithm does not compute it: $\{x \mapsto f(a + b),\ y \mapsto f(a + b)\}$.

Example 4.42. $\mathsf{solve}_{\mathsf{F}}(f \ll^{\mathsf{F}}_{\emptyset} f((\lambda_{\{x\}}\hat{x}.x)f)$ is undefined. Let f be flat. After reducing the right hand side, we get $\mathsf{solve}_{\mathsf{F}}(f \ll^{\mathsf{F}}_{\emptyset} f) = \{Id\}$. Had we formulated the first item of $\approx_{\mathsf{F}}$-rigidity definition (Definition 4.33) with just $n = 1$ as we did for $\approx_{\mathsf{F}}$-rigidity in Definition 4.6, we would have $\mathsf{solve}_{\mathsf{F}}(f \ll^{\mathsf{F}}_{\emptyset} f((\lambda_{\{x\}}\hat{x}.x)f) = \emptyset$ and $\mathbf{C}_2$ would get violated. Similarly, $\mathsf{solve}_{\mathsf{F}}(f \ll^{\mathsf{F}}_{\emptyset} fz)$ would violate $\mathbf{C}_1$.

Example 4.43. Reverting the sides of the equation in the previous example, $\mathsf{solve}_{\mathsf{F}}(f((\lambda_{\{x\}}\hat{x}.x)f) \ll^{\mathsf{F}}_{\emptyset} f)$ is again undefined. Assume f is flat. If we did not require P to be $\approx_{\mathsf{F}}$-rigid in the condition of the Fail rule, we would have $\mathsf{solve}_{\mathsf{F}}(f((\lambda_{\{x\}}\hat{x}.x)f) \ll^{\mathsf{F}}_{\emptyset} f) = \emptyset$, $\mathsf{solve}_{\mathsf{F}}(f \ll^{\mathsf{F}}_{\emptyset} f) = \{Id\}$ and $\mathbf{C}_2$ would be violated. Similarly, $\mathsf{solve}_{\mathsf{F}}(fz \ll^{\mathsf{F}}_{\emptyset} f)$ would violate $\mathbf{C}_1$.

The following lemma plays the crucial role in proving that $\mathsf{solve}_{\mathsf{F}}$ makes the calculus confluent.

Lemma 4.44. *Let P and Q be two terms and χ be a finite set of variables such that $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q)$ is defined. Then:*

1. *If ϑ is a substitution such that $\mathtt{fm}(\vartheta) \cap \hat{\chi} = \emptyset$, then $\mathsf{solve}_{\mathsf{F}}(P\vartheta \ll^{\mathsf{F}}_{\chi} Q\vartheta)$ is defined and $\sigma \in \mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q)$ iff $(\sigma\vartheta)|_{\chi} \in \mathsf{solve}_{\mathsf{F}}(P\vartheta \ll^{\mathsf{F}}_{\chi} Q\vartheta)$.*

2. *If P' and Q' are terms with $P \rightrightarrows P'$ and $Q \rightrightarrows Q'$, then* $\mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q')$ *is defined and* $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q) \rightrightarrows \mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q')$.

Proof. We only prove the second statement. The proof of the first one is similar.

Let $size(M)$ be the size of a term M, defined as follows: $size(x) = size(\hat{x}) = size(f) = 1$, $size(M_1 M_2) = size(\lambda_{\chi} M_1.M_2) = size(M_1 + M_2) = size(M_1) + size(M_2) + 1$. With each term M, we associate its complexity measure $cm(M)$ as a pair of natural numbers $\langle size(M), \#matchables(M)\rangle$, where $\#matchables(M)$ denotes the number of occurrences of free matchables in M. Let $>_{lex}$ be the lexicographic extension of the standard ordering on natural numbers to pairs of natural numbers. Then we define the ordering on terms $M_1 \succeq M_2$ iff $cm(M) >_{lex} cm(M)$. Obviously, $\succeq$ is a well-founded ordering and we prove the lemma by induction for P with respect to this ordering.

We do not consider the case when P is a variable, because it would make $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q)$ undefined, which contradicts our assumption.

Below we only highlight those points which make this proof different from the proof on the analogous lemma for commutative matching (Lemma 4.8).

P is a function symbol f. Let $Q \not\approx f$, then Q must be an $\approx_{\mathsf{F}}$-rigid matchable-sum form, since $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q)$ is defined. In this case $\mathsf{solve}_{\mathsf{C}}(P \ll^{\mathsf{C}}_{\chi} Q) = \emptyset$. By Corollary 4.35, Q' is an $\approx_{\mathsf{F}}$-rigid matchable-sum form. Besides, from reduction of $\approx_{\mathsf{F}}$-rigid matchable-sum forms we can observe that when $Q \not\approx f$ and $Q \rightrightarrows Q'$, then $Q' \not\approx f$. Hence, Q' is an $\approx_{\mathsf{F}}$-rigid matchable-sum, different from f. Therefore, only the Fail rule applies and $\mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q') = \emptyset$.

P is a matchable $\hat{x}$ or an abstraction. Similar to the analogous cases in Lemma 4.25.

P is an application. Since $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q)$ is defined, both P and Q are matchable forms.

First, assume Q is not an application. In this case, $P \ll^{\mathsf{F}}_{\chi} Q$ can be transformed by the Fail rule only (since $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q)$ is defined), and $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{C}}_{\chi} Q) = \emptyset$. When P is a matchable form and application, P is actually a data structure. Therefore, both P and Q must be $\approx_{\mathsf{F}}$-rigid matchable-sum forms in order the Fail rule to apply. Let $P \rightrightarrows P'$ and $Q \rightrightarrows Q'$. By Corollary 4.4, P' is again a data structure. By Corollary 4.9, P' and Q'

are $\approx_{\mathsf{F}}$-rigid matchable-sum forms. Therefore, only the Fail rule applies and $\mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q') = \emptyset$.

Now assume Q is an application. We distinguish between the cases whether $\mathit{head}(Q)$ is a flat symbol or not. When it is not flat, the proof is the same as for the non-commutative head in Lemma 4.25. Therefore, assume the head of Q is a flat symbol f. Let it have a form $Q = fQ_1 \cdots Q_n$, $n \geq 0$. If $\mathit{head}(P) \notin \chi \cup \{f\}$, then $\mathit{head}(P') \notin \chi \cup \{f\}$. Since $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q)$ is defined, both P and Q are $\approx_{\mathsf{F}}$-rigid matchable-sum forms. Then by Corollary 4.35, so are P' and Q', $\mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q')$ is also defined, and $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q) = \mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q') = \emptyset$.

Assume $\mathit{head}(P) = \hat{x}$ for some $x \in \chi$, i.e., P has a form $P = \hat{x}P_1 \cdots P_m$ for $m > 0$. Since $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q)$ is defined, each Q_i, $1 \leq i \leq n$, is a matchable-sum form. Then the ME-Flat-1 rule applies to $P \ll^{\mathsf{F}}_{\chi} Q$ and the solution set can be represented as

$$\begin{aligned}&\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q) = \mathsf{solve}_{\mathsf{F}}(\hat{x}P_1 \cdots P_m \ll^{\mathsf{F}}_{\chi} fQ_1 \cdots Q_n) = \\ &\left(\bigcup_{i=0}^{n} \sigma_i \uplus \mathsf{solve}_{\mathsf{F}}(fP_1 \cdots P_m \ll^{\mathsf{F}}_{\chi\setminus\{x\}} fQ_{i+1} \cdots Q_n)\right) \setminus \{\mathbf{fail}\},\end{aligned} \tag{1}$$

where $\sigma_i = \{x \mapsto fQ_1 \cdots Q_i\}$. From definedness of $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{C}}_{\chi} Q)$ we get definedness of $\mathsf{solve}_{\mathsf{F}}(fP_1 \cdots P_m \ll^{\mathsf{F}}_{\chi\setminus\{x\}} fQ_{i+1} \cdots Q_n)$ for all $0 \leq i \leq n$.

By definition of parallel reduction, $P = \hat{x}P_1 \cdots P_m \rightrightarrows \hat{x}P'_1 \cdots P'_m = P'$, where $P_i \rightrightarrows P'_i$ for $1 \leq i \leq m$, and $Q = fQ_1 \cdots Q_n \rightrightarrows fQ'_1 \cdots Q'_n = Q'$, where $Q_i \rightrightarrows Q'_i$ for $1 \leq i \leq n$. By Corollary 4.4, each Q'_i is a matchable-sum form. Then we can use ME-Flat-1, and the set $\mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q')$ can be represented as

$$\begin{aligned}&\mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q') = \mathsf{solve}_{\mathsf{F}}(\hat{x}P'_1 \cdots P'_m \ll^{\mathsf{F}}_{\chi} fQ'_1 \cdots Q'_n) = \\ &\left(\bigcup_{i=0}^{n} \sigma'_i \uplus \mathsf{solve}_{\mathsf{F}}(fP'_1 \cdots P'_m \ll^{\mathsf{F}}_{\chi\setminus\{x\}} fQ'_{i+1} \cdots Q'_n)\right) \setminus \{\mathbf{fail}\},\end{aligned} \tag{2}$$

where $\sigma'_i = \{x \mapsto fQ'_1 \cdots Q'_i\}$.

We have $\hat{x}P_1 \cdots P_n \succeq fP_1 \cdots P_n$. Therefore, by the induction hypothesis (IH) we can conclude that $\mathsf{solve}_{\mathsf{F}}(fP'_1 \cdots P'_m \ll^{\mathsf{F}}_{\chi\setminus\{x\}} fQ'_{i+1} \cdots Q'_n)$ is defined. Since ME-Flat-1 is the only rule that applies to $P' \ll^{\mathsf{F}}_{\chi} Q'$, it implies that $\mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q')$ is defined. By the IH, for each $0 \leq i \leq n$ we have $\mathsf{solve}_{\mathsf{F}}(fP_1 \cdots P_m \ll^{\mathsf{F}}_{\chi\setminus\{x\}} fQ_{i+1} \cdots Q_n) \rightrightarrows \mathsf{solve}_{\mathsf{F}}(fP'_1 \cdots P'_m \ll^{\mathsf{F}}_{\chi\setminus\{x\}}$

$fQ'_{i+1}\cdots Q'_n)$. Besides, obviously, $\sigma_i \rightrightarrows \sigma'_i$. From these facts and the equalities (1) and (2), by definition of $\circ$ we get $\mathsf{solve}_{\mathsf{F}}(P \ll^{\mathsf{F}}_{\chi} Q) \rightrightarrows \mathsf{solve}_{\mathsf{F}}(P' \ll^{\mathsf{F}}_{\chi} Q')$.

Assume $head(P) = f$. If P has a form $P = \hat{x}P_1 \cdots P_m, \hat{x}$, $m \geq 0$, $x \in \chi$, the proof proceeds in the same way as above, where $head(P) \in \chi$. If P has a form $P = \hat{x}P_1 \cdots P_m$, $m > 1$, $P_m \notin \chi$, then the proof is similar to the proof of Lemma 4.25 when Dec-App applies.

P is a sum. Proof is the same as for the sum case in Lemma 4.25. □

By the reasoning similar to the case of Theorem 4.26, we get the main theorem of this section:

Theorem 4.45. *The function* $\mathsf{solve}_{\mathsf{F}}$ *satisfies* $\mathbf{C}_1$ *and* $\mathbf{C}_2$.

From this theorem and Theorem 3.9 we immediately get the following corollary:

Corollary 4.46. *The finitary pattern calculus that uses* $\mathsf{solve}_{\mathsf{F}}$ *as the pattern matching function, is confluent.*

All the results of this section are valid for both strict and non-strict disjoint unions.

5. Concluding Remarks

We have presented a general framework for pattern calculi with finitary matching, where patterns are first-class citizens. They can be used as parameters, instantiated, reduced, and returned as a result. Unrestricted reductions for pattern calculi lead to non-confluence even in the unitary case. Nondeterminism introduced by finitary matching makes the situation even more complicated.

Our goal was to study conditions under which confluence would be guaranteed for calculi with finitary matching. The approach we considered in this paper builds on the general approaches to proving confluence for pattern calculi with unitary matching proposed by Cirstea and Faure (2007) and Jay and Kesner (2009). Like those works, we also consider the matching function to be a parameter of the general framework. This function is denoted by solve in this paper. Instantiating it by concrete algorithms, one obtains concrete versions of pattern calculi. We were interested to find conditions for solve, which would guarantee a general confluence result for the framework and, thus, would ensure confluence of its concrete instantiations, provided that the conditions are satisfied.

We formulated two such conditions in this paper, denoted by $\mathbf{C}_1$ and $\mathbf{C}_2$. They ensure that the function solve is stable by substitution and by reduction. These conditions generalize their counterparts from the unitary matching (Cirstea and Faure, 2007; Jay and Kesner, 2009). The solve function is defined in such a way that it computes solutions to matching problems and does not free bound variables during reduction. We illustrated two concrete matching functions that satisfy $\mathbf{C}_1$ and $\mathbf{C}_2$, one for commutative theories and the other one for flat patterns. We also discussed a version of commutative matching, which is polymorphic for commutative symbols and permits abstracting with respect to them.

In another development, an interesting approach to show metatheoretical results (confluence, standardization) of the pure pattern calculus from (Jay and Kesner, 2009) has been proposed in (van Oostrom and van Raamsdonk, 2014). There, the calculus has been represented as a higher-order pattern rewriting system as defined by Nipkow (1991) and Mayr and Nipkow (1998). It was shown that by applying general higher-order rewriting results to this rewriting system, one can obtain properties of the pattern calculus. It is worth to investigate how this approach can be extended to calculi with finitary matching, which would require reasoning with equational higher-order rewriting, but that is beyond the scope of this paper.

Yet another direction for future work is to study normalization strategies for some pattern calculi with finitary matching. Recently, Bonelli et al. (2012) addressed this problem for pure pattern calculus of Jay and Kesner (2009). Matching failure gives this calculus non-sequential behavior. The question is, when matching $P \ll^?_\chi Q$ is undecided in the term $(\lambda_\chi P.N)Q$, which subterm (P, N, or Q) should be reduced first in order to get a normal form of $(\lambda_\chi P.N)Q$. Non-sequentiality means that for a given term M containing redexes under a certain redex-free context, there is no redex in M that, first, is needed (i.e., every reduction sequence from M reduces it or its residual) and, second, its choice is independent from the other redexes. Bonelli et al. (2012) extended the notion of needed redexes to so called necessary set of redexes and proposed a normalizing strategy for pure pattern calculus. By this strategy, a set of redexes reduces simultaneously at each step. It generalizes the leftmost-outermost strategy for λ-calculus and is strictly finer than parallel-outermost. It would be interesting to study how this approach can be extended to pattern calculi with finitary matching.

Acknowledgments

We would like to thank Horatiu Cirstea, Delia Kesner, and the referees for interesting comments and constructive suggestions.

This research has been partially supported by the the ERDF through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, by National Funds through the FCT as part of project UID/EEA/50014/2013, by the Rustaveli National Science Foundation under the grants FR/508/4-120/14 and YS15_2.1.2_70, and by the Austrian Science Fund (FWF) under the projects P 24087-N18 and P 28789-N32.

References

Arrighi, P., Dowek, G., 2008. Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. In: Voronkov, A. (Ed.), Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings. Vol. 5117 of Lecture Notes in Computer Science. Springer, pp. 17–31.

Baader, F. (Ed.), 2007. Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings. Vol. 4533 of Lecture Notes in Computer Science. Springer.

Balland, E., Brauner, P., Kopetz, R., Moreau, P., Reilles, A., 2007. Tom: Piggybacking rewriting on Java. In: Baader (2007), pp. 36–47.

Barendregt, H., 1984. The Lambda Calculus: Its Syntax and Semantics. North-Holland, revised edition.

Bonelli, E., Kesner, D., Lombardi, C., Ríos, A., 2012. Normalisation for dynamic pattern calculi. In: Tiwari, A. (Ed.), 23rd International Conference on Rewriting Techniques and Applications (RTA'12), RTA 2012, May 28 - June 2, 2012, Nagoya, Japan. Vol. 15 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 117–132.

Borovanský, P., Kirchner, C., Kirchner, H., Moreau, P., Vittek, M., 1996. ELAN: A logical framework based on computational systems. Electr. Notes Theor. Comput. Sci. 4, 35–50.

Cirstea, H., Faure, G., 2007. Confluence of pattern-based calculi. In: Baader (2007), pp. 78–92.

Cirstea, H., Kirchner, C., 2001. The rewriting calculus - parts I and II. Logic Journal of the IGPL 9 (3), 339–410.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C. L. (Eds.), 2007. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. Vol. 4350 of Lecture Notes in Computer Science. Springer.

Coelho, J., Florido, M., 2007. XCentric: logic programming for XML processing. In: Fundulaki, I., Polyzotis, N. (Eds.), 9th ACM International Workshop on Web Information and Data Management (WIDM 2007), Lisbon, Portugal, November 9, 2007. ACM, pp. 1–8.

de Moor, O., Sittampalam, G., 2001. Higher-order matching for program transformation. Theor. Comput. Sci. 269 (1-2), 135–162.

Diaconescu, R., Futatsugi, K., 1998. An overview of CafeOBJ. Electr. Notes Theor. Comput. Sci. 15, 285–298.

Díaz-Caro, A., Petit, B., 2012. Linearity in the non-deterministic call-by-value setting. In: Ong, C. L., de Queiroz, R. J. G. B. (Eds.), Logic, Language, Information and Computation - 19th International Workshop, WoLLIC 2012, Buenos Aires, Argentina, September 3-6, 2012. Proceedings. Vol. 7456 of Lecture Notes in Computer Science. Springer, pp. 216–231.

Dundua, B., Florido, M., Kutsia, T., Marin, M., 2016. *CLP(H):* constraint logic programming for hedges. Theory and Practice of Logic Programming 16 (2), 141–162.

Dundua, B., Kutsia, T., Marin, M., 2009. Strategies in PρLog. In: Fernández, M. (Ed.), Proceedings Ninth International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2009, Brasilia, Brazil, 28th June 2009. Vol. 15 of EPTCS. pp. 32–43.

Dundua, B., Kutsia, T., Reisenberger-Hagmayr, K., 2017. An overview of PρLog. In: Lierler, Y., Taha, W. (Eds.), Proceedings of the 19th International Symposium on Practical Aspects of Declarative Languages (PADL

2017). Vol. 10137 of Lecture Notes in Computer Science. Springer, pp. 34–49.

Ehrhard, T., Regnier, L., 2003. The differential lambda-calculus. Theor. Comput. Sci. 309 (1-3), 1–41.

Goguen, J. A., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.-P., 2000. Introducing OBJ. In: Goguen, J., Malcolm, G. (Eds.), Software Engineering with OBJ: Algebraic Specification in Action. Springer, Boston, MA, pp. 3–167.

Jay, B., 2009. Pattern Calculus. Springer.

Jay, C. B., Kesner, D., 2006. Pure pattern calculus. In: Sestoft, P. (Ed.), Programming Languages and Systems, 15th European Symposium on Programming, ESOP 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006, Proceedings. Vol. 3924 of Lecture Notes in Computer Science. Springer, pp. 100–114.

Jay, C. B., Kesner, D., 2009. First-class patterns. J. Funct. Program. 19 (2), 191–225.

Klop, J. W., van Oostrom, V., de Vrijer, R. C., 2008. Lambda calculus with patterns. Theor. Comput. Sci. 398 (1-3), 16–31.

Kutsia, T., 2008. Flat matching. J. Symb. Comput. 43 (12), 858–873.

Mayr, R., Nipkow, T., 1998. Higher-order rewrite systems and their confluence. Theor. Comput. Sci. 192 (1), 3–29.

Nipkow, T., 1991. Higher-order critical pairs. In: Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991. IEEE Computer Society, pp. 342–349.

Pagani, M., Ronchi Della Rocca, S., 2010. Solvability in resource lambda-calculus. In: Ong, C. L. (Ed.), Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings. Vol. 6014 of Lecture Notes in Computer Science. Springer, pp. 358–373.

Peyton Jones, S. L., 1987. The Implementation of Functional Programming Languages. Prentice-Hall.

van den Brand, M., van Deursen, A., Heering, J., de Jong, H., de Jonge, M., Kuipers, T., Klint, P., Moonen, L., Olivier, P. A., Scheerder, J., Vinju, J. J., Visser, E., Visser, J., 2001. The ASF+SDF meta-environment: a component-based language development environment. Electr. Notes Theor. Comput. Sci. 44 (2), 3–8.

van Oostrom, V., 1990. Lambda calculus with patterns. Tech. Rep. IR-228, Vrije Universiteit, Amsterdam.

van Oostrom, V., van Raamsdonk, F., 2014. The dynamic pattern calculus as a higher-order pattern rewriting system. In: Rose, K. (Ed.), Proceedings of the 7th International Workshop on Higher-Order Rewriting (HOR 2014).

van Oostrom, V., van Raamsdonk, F., 2015. Matching in the dynamic pattern calculus. In: Escobar, S., Villaret, M. (Eds.), Proceedings of the 29th International Workshop on Unification (UNIF 2015). pp. 51–55.

Wolfram, S., 2003. The Mathematica book (5. ed.). Wolfram-Media.