Higher-order matching modulo (super)developements. Applications to second-order matching

Germain Faure

HIGHER-ORDER MATCHING MODULO (SUPER)DEVELOPMENTS APPLICATIONS TO SECOND-ORDER MATCHING

GERMAIN FAURE

Parc Orsay Université, 4 rue Jacques Monod, 91893 Orsay Cedex, France
e-mail address: germain.faure@inria.fr

Abstract. To perform higher-order matching, we need to decide the $\beta\eta$-equivalence on λ-terms. The first way to do it is to use simply typed λ-calculus and this is the usual framework where higher-order matching is performed. Another approach consists in deciding a restricted equivalence. This restricted equivalence can be based on finite developments or more interestingly on finite superdevelopments. We consider higher-order matching modulo (super)developments over untyped λ-terms for which we propose terminating, sound and complete matching algorithms.

This is in particular of interest since all second-order β-matches are matches modulo superdevelopments. We further propose a restriction to second-order matching that gives exactly all second-order matches. We finally apply these results in the context of higher-order rewriting.

Contents

Introduction

Higher-order matching and unification are two operations fundamental in various fields such as higher-order logic programming [Mil90] and logical frameworks [Pfe01], computational linguistics [DSP91], program transformation [HL78, Shi94, Vis05], higher-order rewriting [vOvR93, MN98, NP98], proof theory etc.

Higher-order matching and unification in the λ-calculus cannot be studied directly since this requires to decide the equality between two terms and the equality modulo β of terms of the λ-calculus is undecidable as it was shown by Church. Nevertheless, in practice we do

Received by the editors October 19, 2009.
1998 ACM Subject Classification: F.4.1, F.3.1.
Key words and phrases: Matching, λ-calculus, developments, superdevelopments, higher-order matching, second-order matching, patterns *à la* Miller, higher-order rewriting.

DOI:10.2168/LMCS-???

not need the full power of the pure λ-calculus. For example, in the context of automated deduction we study unification in a typed setting (Curry-Howard-de Bruijn isomorphism). Unification in this context is still undecidable [Hue75] but the terms written in practice often satisfy some properties that make unification decidable [Mil91] and even linear [Qia96].

The work presented in this paper is dealing with higher-order matching. Higher-order matching is usually defined as the following problem: given a set of equations $s_i = t_i$ between *typed* λ-terms where the terms t_i do not contain free variables, is there a substitution σ such that for all i $s_i\sigma$ is equal to t_i modulo the usual $\beta(\eta)$ relation. If we solve the equations modulo the $\beta\eta$ relation, the problem is known to be decidable [Sti09]. But if we solve the equations modulo the β relation, it is undecidable [Loa03].

Even if higher-order matching is a particular case of unification, it requires dedicated works (see *e.g.* [HL78]). But until now, most of the algorithms are specializations of the general unification algorithm introduced in [Hue75]. This makes them unnecessarily difficult to understand and to use in practice.

We propose a new approach to deal with higher-order matching. Instead of deciding the equality modulo the β-equivalence in the typed λ-calculus, we propose to decide the equality modulo a restriction of the β-equivalence in the pure λ-calculus. The standard restriction of the β-equivalence is given by finite developments [Bar84]. Unfortunately, this restriction is too rough for being useful in the context of higher-order matching. We will show that it is neither complete for tackling second-order matching problems nor matching of patterns *à la* Miller .

We thus consider the more general notion of superdevelopments [vR96, vR93]. A superdevelopment is a reduction sequence that may reduce the redexes of the term, its residuals (like in developments) and some created redexes but not those created by the substitution of a variable in functional position by a λ-abstraction.

In this work, we thus consider matching equations built over untyped λ-terms and solve them modulo superdevelopments. The matching problems are of interest particularly because the set of matches modulo superdevelopments contains, but is not restricted to, second-order β-matches.

We propose a sound complete and terminating algorithm for matching modulo superdevelopments. We show that we can decline this algorithm in several ways: to deal with matching modulo developments, to deal with second-order matching etc. We also show that whereas in a typed context the use of η-long normal form is fundamental, in the context of matching modulo superdevelopments the use of the η-equivalence does not impact our algorithms.

The paper also deals with higher-order rewriting [Ter03]. Higher-order rewriting is usually build on a given instance of the λ-calculus, called the *substitution calculus* [Oos94], for which matching is decidable. Typically, we consider simply typed λ-calculus modulo $\beta\eta$ [MN98] or untyped λ-calculus modulo developments [Klo80, KvOvR93]. Higher-order matching is used in the context of higher-order rewriting to decide whether a rewrite rule can be applied. Our contributions in this context are (1) to give an algorithm for matching modulo developments and (2) to show that the λ-calculus modulo superdevelopments is a very good substitution calculus.

Higher-order matching in an untyped setting was already been studied in [Sit01, dMS01] where matching equations are solved modulo a one-step reduction that generalizes the Tait and Martin-Löf parallel reduction. But this one-step reduction is, like the authors of the original paper said, difficult to understand. We show in this paper that it is nothing but a parallel reduction that corresponds [vR96, vR93] to superdevelopments.

The theory of superdevelopments plays a central role in the work presented here. It makes clear the comparison of higher-order match modulo superdevelopments and other approaches, it gives nice intuitions for the properties as well as their proofs. These proofs are *per se* simpler than the one given in the original paper [Sit01, dMS01].

This paper is structured as follows. Section 1 deals with normalization in the λ-calculus. It sets the notations and presents the material used through the paper. In particular, it gives a detailed presentation of the notion of superdevelopments. Section 2 recalls the basic definitions for higher-order matching modulo β in the simply typed λ-calculus. Section 3 defines higher-order matching modulo superdevelopments and studies its expressiveness *w.r.t.* other approaches. Both sections consider the case of η. Section 4 presents an algorithm for matching modulo superdevelopments. We study its properties mainly termination, soundness and completeness. Section 5 presents an algorithm for matching modulo superdevelopments and η. We prove the minimality for Miller patterns. Section 6 and 7 present two declinations of the algorithm for matching modulo superdevelopments: one for second-order matching and one for matching modulo developments. Finally Section 8 applies the results in the context of higher-order rewriting.

An abstract of this work was presented in [Fau06]. The thesis of the author [Fau07] also contains part of the work presented here.

1. Normalization in the lambda-calculus

In this section, we first recall some basic definitions and set some notations related to the λ-calculus. We refer the reader to [Bar84, Dow01] for the fundamental definitions and results on the λ-calculus.

We then define developments, *resp.* superdevelopments, in two different ways: using the underlined, *resp.* the labelled, λ-calculus and using appropriate parallel reductions.

This section is freely inspired by the second chapter of [vR96].

1.1. Typed λ-calculus and β-reduction.

Given a set of base types $\mathfrak{T}_0$, we define the set of types $\mathfrak{T}$ inductively as the smallest set containing $\mathfrak{T}_0$ and such that if α and $\beta \in \mathfrak{T}$ then $(\alpha \to \beta) \in \mathfrak{T}$. The order of a type α denoted $\mathfrak{o}(\alpha)$ is equal to 1, if $\alpha \in \mathfrak{T}_0$. The order of a type $\alpha \to \beta$ is equal to $max(\mathfrak{o}(\alpha) + 1, \mathfrak{o}(\beta))$.

Definition 1.1 (Typed λ-terms)**.** Let $\mathcal{K}$ be a set of constants, having a unique type. For each type $\alpha \in \mathfrak{T}$, we assume given two countably infinite sets of variables of that type, denoted $\mathcal{X}_\alpha$ and $\mathcal{V}_\alpha$. Let $\mathcal{X} = \cup_{\alpha\in\mathfrak{T}}\mathcal{X}_\alpha$ be the set of variables and let $\mathcal{V} = \cup_{\alpha\in\mathfrak{T}}\mathcal{V}_\alpha$ be the set of matching variables. The set $\mathcal{T}_t$ of typed λ-terms is inductively defined as the smallest set containing all variables, all matching variables and all constants, and closed under the following rules:

- If $A, B \in \mathcal{T}_t$ with type resp. $\alpha \to \beta$ and α then $(A\ B) \in \mathcal{T}_t$ with type β.
- If $A \in \mathcal{T}_t$ with type β, and $x \in \mathcal{X}_\alpha$ then $\lambda x. A \in \mathcal{T}_t$ with type $\alpha \to \beta$.

There are two different sets of "variables": the variables belonging to $\mathcal{X}$ on which we abstract and the matching variables belonging to $\mathcal{V}$.

The symbols $A, B, C, \ldots$ range over the set $\mathcal{T}_t$ of terms, the symbols $x, y, z, \ldots$ range over the set $\mathcal{X}$ of variables ($\mathcal{X} \subseteq \mathcal{T}_t$), the symbols $a, b, c, \ldots, f, g, h$ range over a set $\mathcal{K}$ of term constants ($\mathcal{K} \subseteq \mathcal{T}_t$). The symbols $X, Y, \ldots$ range over the set $\mathcal{V}$ of matching variables. Finally, the symbol ε ranges over the set of atoms, which consists of variables, matching variables and constants. To increase the readability, we often write $\varepsilon(A_1, A_2, \ldots, A_n)$ for $(\ldots((\varepsilon\ A_1)\ A_2)\ldots)\ A_n)$ where ε is an atom and $A_1, \ldots, A_n$ are arbitrary terms. All symbols can be indexed. Positions in λ-terms are denoted by $p_1, ..., p_n$. We denote by $\preceq$ the order on positions (prefix order). The subterm of A at position p_1 is denoted by $A_{|p_1}$.

Given the term A_1A_2, by definition the term A_2 is said to be in *applicative position* while the term A_1 is said to be in *functional position*.

The *order* of a constant or a matching variable is defined as the order of its type. The order of a redex $(\lambda x. A)\, B$ is defined as the order of the abstraction $\lambda x. A$. We consider the usual notion of free and bound variables that concerns the variables (matching variables cannot be bound). A term is said to be $\mathcal{V}$-closed if it contains no matching variables and it is $\mathcal{X}$-closed if it contains no free variables. We denote by $\mathrm{fv}(A)$ the free variables of A.

The substitution of variables is defined as usual and avoids variable capture using α-conversion when needed. The substitution of the variable x by A in B is denoted by $B[x := A]$.

As in any calculus involving some binders, we work modulo the *α-conversion* of Church, and modulo the *hygiene-convention* of Barendregt, *i.e.*, free and bound variables have different names.

It may be helpful for the reader familiar with the Combinatory Reduction Systems (CRS) terminology [Klo80, KvOvR93] to note (1) that our matching variables are nothing but the meta-variables of CRS, (2) that terms containing matching variables are nothing but meta-terms of CRS and (3) that $\mathcal{V}$-closed terms are nothing but the terms of CRS. In the same way, substitutions of matching variables defined below correspond to assignments of CRS.

The relation β is defined over the set $\mathcal{T}_t$ of typed λ-terms by

$$(\lambda x. A)B \rightarrow_\beta A[x := B]$$

and we denote by $\twoheadrightarrow_\beta$ its reflexive and transitive closure and by $=_\beta$ its reflexive, symmetric and transitive closure. A λ-term is said to be *β-normal* or simply *normal* if it is in normal form for the β-rule. We recall that the β-reduction over typed λ-terms is confluent and strongly normalising.

A *substitution of matching variables* is a function from matching variables to the set of $\mathcal{V}$-closed terms. It is denoted $B\{A/X\}$. We use the standard definitions for domain, codomain, union and composition of substitutions. When it is clear from the context, substitutions of matching variables are simply called substitutions. Substitutions can be compare using the usual *subsumption order*: for two substitutions ψ and φ, we say that $\psi \leq \varphi$ when there exists a substitution ξ such that $\varphi = \xi \circ \psi$ where $\circ$ denotes substitution composition. In this work, we only consider *closed and normal substitutions* that are substitutions of closed and normal terms.

1.2. **Untyped λ-calculus and developments.** We define developments as a subset of the β-reduction which reduces only the redexes initially present in the term as well as its residuals. This is formalized by defining the *underlined λ-calculus* : we initially underline all the redexes present in the term and we replace the β-reduction by the β_u-reduction which only reduces underlined redexes. Then, the redexes created during reduction are no longer reduced (since they are not underlined).

1.2.1. *Untyped underlined λ-calculus and β_u-reduction.* We define the set of underlined terms. In case they do not use any type information, we use without ambiguity the definitions and notations given for typed λ-terms in the previous section.

Definition 1.2 (Underlined λ-terms)**.** Let $\mathcal{K}$ be a set of constants. Let $\mathcal{X}$ and $\mathcal{V}$ be two countably infinite and disjoint sets respectively for variables and matching variables. The set $\mathcal{T}_u$ of underlined λ-terms is defined as the smallest set containing all variables, matching variables, constants and closed under the following rules:

- If A and B are elements of the set $\mathcal{T}_u$ then (AB) is an element of $\mathcal{T}_u$;
- If A is an element of $\mathcal{T}_u$ and x is a variable of $\mathcal{X}$, then $\lambda x.\, A$ is an element of $\mathcal{T}_u$;
- If A and B are elements of $\mathcal{T}_u$ then $(\underline{\lambda} x.A)B$ belong to $\mathcal{T}_u$.

Note that the set of underlined terms is not closed by subterm: for example $\underline{\lambda} x.A$ is not an element of $\mathcal{T}_u$.

The relation β_u is defined over the set $\mathcal{T}_u$ of underlined terms by

$$(\underline{\lambda} x.A)B \rightarrow_{\beta_u} A[x := B]$$

The β_u-reduction is consuming at each step an underlined redex, it can duplicate but cannot create new ones. This relation is thus strongly normalising. Moreover, the relation is confluent. These results are known as the finite developments theorem (see below).

1.2.2. *Untyped λ-calculus and developments.* The set of terms in the pure λ-calculus is defined in the same way of typed λ-terms except that we do take care of type constraints. It is denoted by $\mathcal{T}$. We can define a mapping Υ from underlined terms to terms that replaces β-redexes by their corresponding β-redexes[1]. This mapping can be extended to any sequence of β_u-reductions. We can then define developments.

Definition 1.3 (Developments)**.** A sequence of β-reductions ζ is a development (also called a complete development) if there exists a sequence σ of β_u-reductions in the underlined λ-calculus which terminates on a term in β_u-normal form and such that $\Upsilon(\sigma) = \zeta$.

Theorem 1.4 (Finite developments [Bar84])**.**

- *Every development is finite.*
- *If two developments ζ_1 and ζ_2 start with the same initial term then the two final terms must be equal.*

We now introduce a big-step semantics [Des98] of developments. This definition is due to Tait and Martin-Löf and can be given for every left linear higher-order rewrite systems [Ter03].

[1] The formal definition will be given in the more general case of superdevelopments.

Definition 1.5 (Parallel reduction)**.** The parallel reduction in the λ-calculus is inductively defined by

$$\frac{}{\varepsilon \Longrightarrow_\beta \varepsilon}\ (Red-\varepsilon) \qquad \frac{A_1 \Longrightarrow_\beta A_2}{\lambda x.\, A_1 \Longrightarrow_\beta \lambda x.\, A_2}\ (Red-\lambda)$$

$$\frac{A_1 \Longrightarrow_\beta A_2 \quad B_1 \Longrightarrow_\beta B_2}{A_1 B_1 \Longrightarrow_\beta A_2 B_2}\ (Red-@) \qquad \frac{\lambda x.\, A_1 \Longrightarrow_\beta \lambda x.\, A_2 \quad B_1 \Longrightarrow_\beta B_2}{(\lambda x.\, A_1) B_1 \Longrightarrow_\beta A_2[x := B_2]}\ (Red-\beta)$$

Theorem 1.6 (Parallel reduction and developments)**.** *The notions of parallel reduction and developments coincide in the following sense. For every terms $A, B \in \mathcal{T}$*

$$\textit{there exists a development } A \twoheadrightarrow_\beta B \textit{ iff } A \Longrightarrow_\beta B$$

This characterization of developments is the essence of the corresponding matching algorithm.

1.3. **Untyped λ-calculus and superdevelopments.** We have seen in the previous section that developments reduce the redexes initially present in the term and its residuals. This gives a first approximation of the β-normal form. Unfortunately, this approximation is too rough for being useful in the context of higher-order matching as we will see in Section 3.5.

We then introduce a generalization of developments called superdevelopments. A superdevelopment [vR93] is a reduction sequence that may reduce the redexes of the term, its residuals and some created redexes. The redexes created by the substitution of a variable in functional position by a λ-abstraction are not reduced.

The notion of superdevelopments is related to the *three ways redexes are created* in the λ-calculus. This taxonomy was proposed in [Lév78].

$$\begin{array}{llll}
\text{(type 1)} & ((\lambda x.\, (\lambda y.\, A))B)C & \to_\beta & (\lambda y.\, A[x := B])C \\
\text{(type 2)} & ((\lambda x.\, x)(\lambda y.\, A))B & \to_\beta & (\lambda y.\, A)B \\
\text{(type 3)} & (\lambda x.\, A)(\lambda y.\, B) & \to_\beta & A[x := (\lambda y.\, B)] \\
 & & & \text{if } \exists p \text{ such that } A|_p = x A_0
\end{array}$$

For the first two ways of creating a β-redex, one can say that the creation is “upwards”, whereas in the last case it can be said to be “downwards”.

Note that in the first and second ways, the redex is created by the *reduction* of a term in functional position whereas in the third way the redex is created by the *substitution* for a variable in functional position of a λ-abstraction. This gives the intuition for the equivalence between superdevelopments and the strong parallel reduction defined below.

1.3.1. *Untyped labelled λ-calculus and β_l-reduction.* By definition, labels are simply elements of $\mathbb{N}$.

Definition 1.7 (Labelled λ-terms)**.** Let $\mathcal{K}$ be a set of constants. Let $\mathcal{X}$ and $\mathcal{V}$ be two countably infinite and disjoint sets respectively for variables and matching variables. The set $\mathcal{T}_l$ of labelled λ-terms is defined as the smallest set containing all variables, matching variables, constants and closed under the following rules:

- If $A \in \mathcal{T}_l$ and $p \in \mathbb{N}$, then $\lambda_p x.A \in \mathcal{T}_l$.
- If $M, N \in \mathcal{T}_l$ and $p \in \mathbb{N}$, then $(MN)^p \in \mathcal{T}_l$.

The relation β_l is defined over the set $\mathcal{T}_l$ of labelled λ-terms by

$$((\lambda_p x.A)B)^p \qquad \rightarrow_{\beta_l} \qquad A[x := B]$$

In order to define superdevelopements we will restrict attention to terms that are labelled such that the label of an application cannot be equal to the label of a λ-abstraction that is not in its scope. This corresponds exactly to "upwards" redex creations.

Definition 1.8 (Well-labelled and initially labelled terms)**.** A labelled term $A \in \mathcal{T}_l$ is said to be *well-labelled* if for all positions such that $A_{|p_1} = (B_0 B_1)^p$ and $A_{|p_2} = \lambda_p x.C$ then $p_1 \preceq p_2$. It is *initially labelled* if moreover for all positions such that $A_{|p_1} = \lambda_p x.C$ and $A_{|p_2} = \lambda_p x'.C'$ then $p_1 = p_2$.

In the following, we will suppose that all labelled terms are *well-labelled*. We can remark that the set of well-labelled terms is closed by β_l-reduction.

1.3.2. *Untyped λ-calculus and superdevelopments.* Before giving the formal definition of superdevelopments, we define an erasing morphism from labelled λ-terms to λ-terms. Without any ambiguity, we overload the notation of the previous section.

Definition 1.9 (Erasing mapping)**.** The mapping $\Upsilon : \mathcal{T}_l \rightarrow \mathcal{T}$ that erases labels is defined as follows

- $\Upsilon(\varepsilon) = \varepsilon$
- $\Upsilon((AB)^p) = \Upsilon(A)\Upsilon(B)$
- $\Upsilon(\lambda_p x \,.\, A) = \lambda x.\, \Upsilon(A)$

Note that the mapping Υ can be extended into a morphism from β_l-reductions to β-reductions.

A superdevelopment is a β-rewrite sequence that may reduce both the redexes that are residuals of redex occurrences in the initial term (like in developments) and the redex occurrences that are created in the first or second way. In the λ-calculus, superdevelopments are, as developments, finite.

Definition 1.10 (Superdevelopments)**.** A β-rewrite sequence ς of the λ-calculus is a β-superdevelopment if there exists a β_l-rewrite sequence σ in the labelled λ-calculus that starts with an initially labelled term and stops on a term in β_l-normal form and such that $\Upsilon(\sigma) = \varsigma$.

We extend the finite developments theorem to superdevelopements.

Theorem 1.11 (Finite superdeveloppements [vR96])**.**

- *Every superdevelopment is finite.*

- *If two superdevelopments ζ_1 and ζ_2 start with the same initial term then the two final terms must be equal.*

As finite developments coincide with the parallel reduction of Tait and Martin-Löf, finite superdevelopments coincide with Aczel's parallel reduction [Acz78] called in the following strong parallel reduction. It is denoted by $\Rightarrow_{\beta_{sd}}$ and we say that a term A β_{sd}-reduces to a term B if $A \Rightarrow_{\beta_{sd}} B$.

Definition 1.12 (Strong parallel reduction)**.** The strong parallel reduction in the λ-calculus is defined inductively by

$$\frac{}{\varepsilon \Rightarrow_{\beta_{sd}} \varepsilon}\ (Red\!-\!\varepsilon)$$

$$\frac{A_1 \Rightarrow_{\beta_{sd}} A_2 \quad B_1 \Rightarrow_{\beta_{sd}} B_2}{A_1 B_1 \Rightarrow_{\beta_{sd}} A_2 B_2}\ (Red\!-\!@) \qquad \frac{A_1 \Rightarrow_{\beta_{sd}} A_2}{\lambda x.\, A_1 \Rightarrow_{\beta_{sd}} \lambda x.\, A_2}\ (Red\!-\!\lambda)$$

$$\frac{A_1 \Rightarrow_{\beta_{sd}} \lambda x.\, A_2 \quad B_1 \Rightarrow_{\beta_{sd}} B_2}{A_1 B_1 \Rightarrow_{\beta_{sd}} A_2[x := B_2]}\ (Red\!-\!\beta_s)$$

The only difference with the parallel reduction of Tait and Martin-Löf is the rule $(Red\!-\!\beta_s)$ that replaces the rule $(Red\!-\!\beta)$ of the parallel reduction. The redex reduced by the rule $(Red\!-\!\beta_s)$ is obtained from the *reduct* of A_1. This redex is not necessarily present in the initial term $A_1 A_2$. It may have been created and this creation is of type 1 or 2 but not of type 3 (an upwards creation but not a downwards creation).

Theorem 1.13 (Strong parallel reduction and superdevelopments)**.** *The notions of strong parallel reduction and superdevelopments coincide in the following sense. For every terms $A, B \in \mathcal{T}$*

$$\textit{there exists a superdevelopment } A \twoheadrightarrow_\beta B \textit{ iff } A \Rightarrow_{\beta_{sd}} B.$$

This characterization of superdevelopments is the essence of the corresponding matching algorithm. We conclude this section by examples.

Example 1.14. In this example, we show how to associate, when it exists, the β_l-rewrite sequence corresponding to a β-rewrite sequence.

- The β-rewrite sequence
$$(\lambda x.\, \lambda y.\, xy)zz' \quad \rightarrow_\beta \quad (\lambda y.\, zy)z' \quad \rightarrow_\beta \quad zz'$$
is a superdevelopment since it corresponds to the β_l-rewrite sequence
$$(((\lambda_1 x.\lambda_2 y.xy)z)^1)z')^2 \quad \rightarrow_{\beta_l} \quad ((\lambda_2 y.zy)z')^2 \quad \rightarrow_{\beta_l} \quad zz'\ .$$
- However, the rewrite sequence
$$(\lambda x.\, xx)(\lambda x.\, xx) \quad \rightarrow_\beta \quad (\lambda x.\, xx)(\lambda x.\, xx) \quad \rightarrow_\beta \quad \ldots$$
is not a superdevelopment. We show that it is not possible to label the term $(\lambda x.\, xx)(\lambda x.\, xx)$ and find an adequate β_l-reduction sequence. We can first try to label the λ-abstractions. Since we consider only initially well-labeled λ-terms, each λ-abstraction must have a different label. Moreover, if we want the term to be β_l-reducible, we necessary have
$$((\lambda_1 x\, .\, xx)(\lambda_2 x\, .\, xx))^1.$$

To conclude the labelling of $(\lambda x.\, xx)(\lambda x.\, xx)$, we have to find two labels p_1 and p_2 such that the term

$$((\lambda_1 x\,.\,(xx)^{p_1})(\lambda_2 x\,.\,(xx)^{p_2}))^1$$

is well-labelled. Since we want the term to be reducible after one β_l-reduction then p_1 must be equal to 2. But this is impossible since we consider a well-labelled term.

Given a λ-term, we can thus "label" this term (and thus obtaining a labelled λ-term) in order to β_l-reduce redexes created in the first or in the second way but not in the third way. This is exactly why we restrict ourselves to well-labelled terms.

The corresponding β_l-rewrite sequence associated to a superdevelopment is no more given in the following.

Example 1.15. In this example, we illustrate the link with developments, superdevelopments and redex creations.

- **Finite development** Residuals of redexes present in the initial term can be contracted:
$$\begin{array}{l}(\lambda x.\, f(x,x))\,((\lambda y.\, y)\, a)\\ \to_\beta f((\lambda y.\, y)\, a, (\lambda y.\, y)\, a)\\ \to_\beta f(a, (\lambda y.\, y)\, a)\\ \to_\beta f(a,a)\end{array}$$
- **Redex creation of type 1** In the following superdevelopment, the new redex obtained after one β-rewrite step is reduced:
$$\begin{array}{l}((\lambda x.\, \lambda y.\, f(x,y))a)b\\ \to_\beta (\lambda y.\, f(a,y))b\\ \to_\beta f(a,b)\end{array}$$
- **Redex creation of type 2** As in the previous example, a redex is created and reduced during the reduction, but in a different way:
$$\begin{array}{l}((\lambda x.\, x)(\lambda y.\, y))a\\ \to_\beta (\lambda y.\, y)a\\ \to_\beta a\end{array}$$
- **Redex creation of type 3** There is no superdevelopment from the term $(\lambda x.\, xa)(\lambda y.\, y)$ to the term a:
$$\begin{array}{l}(\lambda x.\, xa)(\lambda y.\, y)\\ \to_\beta (\lambda y.\, y)a\end{array}$$

2. Matching modulo beta (and eta)

In this section, we consider terms of the simply λ-calculus that is, elements of $\mathcal{T}_t$. We are going to solve equations modulo the full β-equivalence (possibly with η).

2.1. Matching modulo β.

Definition 2.1 (β-equation/system)**.** A β-equation is a pair of β-normal typed λ-terms of the same type denoted $A \leqslant_\beta B$ such that B is $\mathcal{V}$-closed. A β-system is a multiset (possibly empty) of β-equations.

Union of multisets is written using the symbol $\cup$. If E_1 and E_2 are matching equations, we simply write $(E_1) \cup (E_2)$ for the multiset of the two matching equations.

For example, let us consider a base type ι, a constant a of type ι and two matching variables X and Y with respective types $\iota \to \iota$ and ι. Then, the pair (XY, a) is a β-equation.

Definition 2.2 (β-match)**.** A substitution φ, that preserves types, is a β-match for the matching equation $A \leqslant_\beta B$ if and only if $A\varphi =_\beta B$. A substitution is a β-match for a system $\mathbb{S}$ if it is a β-match for each equation of $\mathbb{S}$.

For example, the substitution $\{\lambda x.\, x/X,\ a/Y\}$ is a β-match for $XY \leqslant_\beta a$.

2.2. Nth order matching.

In practice, we only consider algorithms to solve a subset of β-equations (higher-order matching modulo β is undecidable [Loa03]). Either we consider some restrictions on the order of matching variables and constants or we consider matching modulo $\beta\eta$.

Definition 2.3 (Order of an equation/system)**.** A β-equation is said to be of order at most n if all its matching variables are of order at most n and all its constants are of order at most $n+1$. A system is said to be of order at most n if it is composed of equations of order at most n. We define the nth order matching as the operation that solves β-equations of order at most n.

For example, the above equation $XY \leqslant_\beta a$ is of order 2.

2.3. Matching modulo $\beta\eta$.

In practice, the adequate equivalence modulo which we want to consider matching equations is the $\beta\eta$-equivalence where the equation η is defined as

$$(\eta) \qquad \lambda x.\,(Ax) \quad = \quad A \quad \textbf{if } x \notin \mathrm{fv}(A)$$

The decidability of second-order matching was proved in [HL78], of third order matching in [Dow94] and of fourth order matching in [Pad00]. The general decidability proof appears recently in [Sti09] using game theory.

In this section, we simply reformulate the previously given definitions to take into account η-equivalence.

Definition 2.4 ($\beta\eta$-equation/system)**.** A $\beta\eta$-equation is a pair of typed λ-terms in $\beta\eta$-long normal form of the same type denoted $A \leqslant_{\beta\eta} B$ such that B is $\mathcal{V}$-closed. A $\beta\eta$-system is a multiset (possibly empty) of $\beta\eta$-equations.

Definition 2.5 ($\beta\eta$-match)**.** A substitution φ, that preserves types, is a $\beta\eta$-match for the matching equation $A \leqslant_\beta B$ if and only if $A\varphi =_{\beta\eta} B$. A substitution is a $\beta\eta$-match for a system $\mathbb{S}$ if it is a $\beta\eta$-match for each equation of $\mathbb{S}$.

2.4. **Matching of Miller patterns.** We first define Miller patterns.

Definition 2.6 (Miller Patterns)**.** A simply typed λ-term A in β-normal form is a (higher-order) Miller pattern if every free occurrence of a variable X is in a subterm $X(u_1, \dots, u_n)$ of A such that $(u_1, \dots, u_n)$ is η-equivalent to a list of distinct bound variables.

Unlike the higher-order unification which is undecidable in general [Hue73, Gol81], the unification of Miller patterns is decidable and there exists a more general unifier (when the equation has a solution) that can be computed in linear time [Qia96]. In Section 5, we will give an algorithm for matching modulo superdevelopments and η that gives exactly (when it exists) the more general match for equations based on Miller patterns.

3. Matching modulo superdevelopments (and eta)

In this section, we first define matching modulo superdevelopments, that we also call β_{sd}-matching. We then relate it with second and third order matching.

3.1. **Matching modulo superdevelopments.** In this section, we consider terms of pure λ-calculus (elements of $\mathcal{T}$). We are going to solve equations modulo superdevelopments. We first define the notion of equation and then the corresponding notion of solutions.

Definition 3.1 (β_{sd}-matching equation/system)**.** A *β_{sd}-matching equation* or simply a *matching equation* is a pair of terms denoted $A \leqslant_{\beta_{sd}} B$ such that B is normal and $\mathcal{V}$-closed. A matching *system* is a multiset (potentially empty) of matching equations.

We say that a matching variable belongs to a system $\mathbb{S}$ and we note $X \in \mathbb{S}$ if X occurs in one equation of $\mathbb{S}$.

For example, the pairs $(XY,\, \lambda x.\, x)$ and $((\lambda x.\, x)X,\, a)$ are β_{sd}-matching equations whereas $(XY,\, (\lambda x.\, x)a)$ is not since the term $(\lambda x.\, x)a$ is not normal.

Higher-order matching and unification algorithms that can be found in the literature [Hue75, SG89, Dow01] usually consider matching equations of terms in normal form and this property is preserved during the matching process by using normalizing substitutions. Note that this is not the case here: the term A of an equation $A \leqslant_{\beta_{sd}} B$ is not necessarily in normal form.

Definition 3.2 (β_{sd}-match)**.** A substitution φ on matching variables is a *β_{sd}-match* or simply a *match* for the matching equation $A \leqslant_{\beta_{sd}} B$ if there exists a superdevelopment from $A\varphi$ and B (that is, $A\varphi \Rightarrow_{\beta_{sd}} B$). A substitution is a match of a system if it matches each equation. The set of all matches of a system $\mathbb{S}$ is denoted $\mathbb{M}(\mathbb{S})$.

We can associate to a typed λ-term an untyped term. In the same way, we can associate to a β-equation a β_{sd}-equation. By a little abuse of notation, we simply denote by $A \leqslant_{\beta_{sd}} B$ the β_{sd}-equation associated to the β-equation $A \leqslant_{\beta} B$.

Recall that we only consider substitutions of closed and normal terms. In particular, a β_{sd}-match is thus a substitution of closed and normal terms.

The application of a substitution to a matching equation $B \leqslant_{\beta_{sd}} C$ is the equation $B\varphi \leqslant_{\beta_{sd}} C$. The application of a substitution φ to a system, denoted $\mathbb{S}\varphi$ consists in the application of the substitution φ to each matching equation of $\mathbb{S}$.

Example 3.3 (β_{sd}-matches)**.** Consider the equation $(XY)Z \leqslant_{\beta_{sd}} ab$ and the substitutions

$$\begin{array}{lclrlrll}
\sigma_1 & = & \{ & \lambda x.\, x & /X, & a/Y, & b & /Z\} \\
\sigma_2 & = & \{ & \lambda x.\, \lambda y.\, xy & /X, & a/Y, & b & /Z\} \\
\sigma_3 & = & \{ & \lambda x.\, \lambda y.\, y & /X, & & ab & /Z\} \\
\sigma_4 & = & \{ & \lambda y.\, \lambda x.\, xy & /X, & b/Y, & \lambda z.\, az & /Z\}
\end{array}$$

The substitutions σ_1, σ_2 and σ_3 are β_{sd}-matches since

$$\begin{array}{lcll}
((XY)Z)\sigma_1 & = & ((\lambda x.\, x)a)b & \Rightarrow_{\beta_{sd}} \; ab \\
((XY)Z)\sigma_2 & = & ((\lambda x.\, \lambda y.\, xy)a)b & \Rightarrow_{\beta_{sd}} \; ab \\
((XY)Z)\sigma_3 & = & ((\lambda x.\, \lambda y.\, y)Y)\,(ab) & \Rightarrow_{\beta_{sd}} \; ab
\end{array}$$

The substitution σ_4 is not a β_{sd}-match since

$$((XY)Z)\sigma_4 \;=\; ((\lambda y.\, \lambda x.\, xy)b)(\lambda z.\, az) \qquad \not\Rightarrow_{\beta_{sd}} \; ab$$

even if these terms are β-convertible.

A β_{sd}-system can have an infinite number of solutions but we can always find a finite set of minimal solutions such that every solution is subsumed by a minimal solution.

Example 3.4 (Infinite number of β_{sd}-matches)**.** Following Example 3.3, we can notice that all the substitutions that coincide with σ_3 are solutions of the equations, independently from the term associated to the matching variable Y. For example,

$$\sigma_5 = \{\lambda x.\, \lambda y.\, y/X, \lambda x.\, x/Y, ab/Z\} \qquad \text{and} \qquad \sigma_6 = \{\lambda x.\, \lambda y.\, y/X, \lambda x.\, \lambda y.\, y/Y, ab/Z\}.$$

are solutions. They are subsumed by σ_3 in the following sense

$$\sigma_3 \leq \sigma_5 \qquad \text{and} \qquad \sigma_3 \leq \sigma_6$$

The following example shows that there is no most general match.

Example 3.5 (Independant β_{sd}-match)**.** Following example 3.3 we can notice that the substitution σ_1 and σ_2 are not comparable since $\sigma_1 \not\leq \sigma_2$ and $\sigma_2 \not\leq \sigma_1$.

In the following we are going to solve β_{sd}-equations by transformation rules. We simplify a system until getting a 'normal form' for which we can extract a substitution (if it exists). Such a system is said to be in solved form in the sense of the following definition.

Definition 3.6 (Solved form)**.** A matching equation $X \leqslant_{\beta_{sd}} A$ is in *solved form* if A contain no free variables. The corresponding substitution is defined by $\{A/X\}$. A system is in solved form if all its equations are in solved form and if the left-hand sides are pairwise disjoint. The corresponding substitution of such a system is the union of the corresponding substitutions of each equation (of the system). It is denoted by $\sigma_{\mathbb{S}}$.

Definition 3.7 (Complete match set)**.** Let $\mathbb{S}$ be a matching system. A *complete match set* of $\mathbb{S}$ is a set of substitutions $\mathbb{M}$ such that:

(1) **Soundness** For all $\varphi \in \mathbb{M}$, φ is a β_{sd}-match of $\mathbb{S}$.
(2) **Completeness** For all φ such that φ is a β_{sd}-match of $\mathbb{S}$ there exists $\psi \in \mathbb{M}$ such that $\psi \leq \varphi$, *i.e.*, there exists a substitution ξ such that $\varphi = \xi \circ \psi$ where $\circ$ denotes substitution composition.

The following lemma gives the relevance of solved forms.

Lemma 3.8. *If $\mathbb{S}$ is a system in solved form then $\{\sigma_{\mathbb{S}}\}$ is a complete match set of $\mathbb{S}$.*

3.2. **Comparison with second-order matching.** In this section, we are going to prove that matching modulo superdevelopments is complete for second-order matching. The following results were already proved in [dMS01]. Nevertheless, the technical material presented here (creations of redexes, superdevelopments) gives simple and intuitive proofs. First, a technical result on the creation of redexes.

Lemma 3.9. *For all terms $A_1, A_2, \cdots, A_n$ such that A_n contains a redex of third order (or more) and there exists a superdevelopment $A_1 \twoheadrightarrow_\beta A_2 \twoheadrightarrow_\beta \ldots \twoheadrightarrow_\beta A_n$, then A_1 contains also a redex of third order (or more).*

Proof. We prove the result by induction on n. We look at the induction case. By induction hypothesis, we know that A_2 contains a redex of a least third order that we call in the following $R = (\lambda x.\, C)\ D$. First, if R is a residual of a redex of A_1 then the result is obvious. Secondly, if not, and if R is created during the reduction from A_1 to A_2 in the first way mentioned before then A_1 must contain a subterm of the form $(((\lambda z.\, \lambda x.\, C')\ E)\ D$ with $C = C'[z := E]$. Then the order of the redex $(\lambda z.\, \lambda x.\, C')\ E$ is greater or equal to the order of R. This concludes the case. Finally, if not, and if R is created during the reduction from A_1 to A_2 in the second way mentioned before then A_1 must contain a subterm of the form $(\lambda y.\, y)\ (\lambda x.\, C)\ D$. The order of the redex $(\lambda y.\, y)\ (\lambda x.\, C)$ is strictly greater than the one of R. This concludes the case. $\square$

Proposition 3.10. *Consider a second-order β-matching equation. If a substitution φ is a β-match then it is a β_{sd}-match.*

Proof. The proof is by contradiction. Let φ be a β-match of the β-matching equation $A \leqslant_\beta B$ that is not a β_{sd}-match. Then we have that A does not contain any β-redex and that φ does not contain any term of order greater than 2. Finally, $A\varphi \not\twoheadrightarrow_{\beta_{sd}} B$ and $A\varphi =_\beta B$. Thus there exist $(A_i)_i$ such that $A\varphi \twoheadrightarrow_\beta A_1 \twoheadrightarrow_\beta \cdots \twoheadrightarrow_{\beta_{sd}} A_n$ is a superdevelopments and A_n contains a β-redex $(\lambda x.\, C)D$ which is not reduced by superdevelopments. This means that this redex is a residual of a redex created when reducing A_{i_0}. Since the redex is not reduced by superdevelopments then this creation is of type 3 and thus induces a redex of order at least 3. Lemma 3.9 implies that $A\varphi$ contains a redex of order at least 3. Since both A and φ range in the set of β-normal forms, then there exists a position p_1 and a term E such that $A_{|p_1} = XE$ where X is mapped by φ to a λ-abstraction of at least third order. This contradicts the hypothesis on the order of the initial matching problem. $\square$

This proposition for second-order β-equations can be easily generalised to second-order β-systems.

Creations of redexes in the third way induce intrinsically redexes of at least third order. This intuitively explains why second-order matches modulo β are β_{sd}-matches. The reader familiar with the second-order matching algorithm of G. Huet and B. Lang may notice that during their matching process, we can restrict β-normalization to β_{sd}-normalization.

3.3. **Comparison with third-order matching.** As soon as we consider third-order matching problems, the set of minimal solutions may be infinite. Since matching modulo superdevelopments generates finitely many minimal solutions, we remark that matching modulo superdevelopments cannot be complete *w.r.t.* third-order matching.

Example 3.11. The substitution $\{\lambda x.\, \lambda f.\, fx/X\}$ is a β-match for the matching equation $\lambda z.\, (X\ z\ (\lambda y.\, y)) \leqslant \lambda z.\, z$ whereas it is not a β_{sd}-match. In fact, $\lambda z.\, ((\lambda x.\, \lambda f.\, fx)\ z\ (\lambda y.\, y))$ β_{sd}-reduces to $\lambda z.\, (\lambda y.\, y)z$ but not to $\lambda z.\, z$.

The last example is classical and taken from [Dow01]. The third-order matching equation has an infinite number of (minimal) solutions of type $\iota \to (\iota \to \iota) \to \iota$ that are given by the Church numbers $\lambda x.\, \lambda f.\, (f \ldots (f\ x) \ldots)$.

3.4. **Comparison with matching of Miller patterns.** In the case of matching of Miller patterns [Mil91, Qia96], the restriction of the β-reduction given by superdevelopments is powerful enough:

Proposition 3.12. *Let φ be a match of an equation $P \leqslant_\beta A$ where P is a Miller pattern. Then there exists a superdevelopment $P\varphi \twoheadrightarrow_\beta A$.*

Proof. For all Miller patterns, only a subset of the β-reduction is needed [Mil91]. This restriction is defined by

$$(\lambda y.\, M)x \qquad \to_{\beta_0} \qquad M[y := x]$$

We recall that redex creations of type 2 or 3 require to reduce a redex whose term in application position is a λ-abstraction. Then since β_0-reduction only reduces redexes whose term in application position is a variable, only redex creations of type 1 can occur in the context of the β_0-reduction. These created redexes will be reduced by superdevelopments. □

3.5. **Matching modulo superdevelopments and eta.** In the simply typed λ-calculus, matching modulo $\beta\eta$ strongly relies on the η-long normal form. This makes an important difference with matching modulo β: using the η-equivalence, we move from a undecidable [Loa03] to a decidable problem [Sti09].

In the context of matching modulo superdevelopments, the use of η does not influence neither fundamentally nor technically the design of matching algorithms, as we will see in Section 5.

The use of η-long normal form is here replaced by the use of η-reduction to consider terms in η-normal form. We recall the definition of η-reduction.

$$\lambda x.\, (Ax) \qquad \to_\eta \qquad A \quad \text{if } x \notin \mathrm{fv}(A)$$

Definition 3.13 ($\beta_{sd}\eta$-equation and $\beta_{sd}\eta$-system)**.** A $\beta_{sd}\eta$-matching equation is defined by a pair of terms (A, B) such that B is $\beta\eta$-normal and $\mathcal{V}$-closed. We denote such an equation by $A \leqslant^\eta_{\beta_{sd}} B$. A $\beta_{sd}\eta$-system is a multiset of equations.

Definition 3.14 ($\beta_{sd}\eta$-match)**.** We say that φ is a $\beta_{sd}\eta$-match for the $\beta_{sd}\eta$-equation $A \leqslant^\eta_{\beta_{sd}} B$ if there exists a term C such that $A\varphi \Rightarrow_{\beta_{sd}} C \twoheadrightarrow_\eta B$.

These two notions will be illustrated in Section 5.

$$
\begin{array}{lll}
(x \leqslant_{\beta_{sd}} x) \cup \mathbb{S} & \rightarrow_{\varepsilon_v} & \mathbb{S} \\
(a \leqslant_{\beta_{sd}} a) \cup \mathbb{S} & \rightarrow_{\varepsilon_c} & \mathbb{S} \\
(X \leqslant_{\beta_{sd}} A) \cup \mathbb{S} & \rightarrow_{\varepsilon_X} & (X \leqslant_{\beta_{sd}} A) \cup \{A/X\}\mathbb{S} \\
 & & \textbf{if } \mathrm{fv}(A) = \emptyset \textbf{ and } X \in \mathbb{S} \\
(\lambda x.\, A \leqslant_{\beta_{sd}} \lambda x.\, B) \cup \mathbb{S} & \rightarrow_{\lambda_\lambda} & (A \leqslant_{\beta_{sd}} B) \cup \mathbb{S} \\
(A_1 B_1 \leqslant_{\beta_{sd}} A_2 B_2) \cup \mathbb{S} & \rightarrow_{@_@} & (A_1 \leqslant_{\beta_{sd}} A_2) \cup (B_1 \leqslant_{\beta_{sd}} B_2) \cup \mathbb{S} \\
(A_1 B_1 \leqslant_{\beta_{sd}} C) \cup \mathbb{S} & \rightarrow_{@_\pi} & (A_1 \leqslant_{\beta_{sd}} \lambda x.\, C) \cup \mathbb{S} \\
 & & \textbf{where } x \textbf{ fresh variable} \\
(A_1 B_1 \leqslant_{\beta_{sd}} C) \cup \mathbb{S} & \rightarrow_{@_\beta} & (A_1 \leqslant_{\beta_{sd}} \lambda x.\, A_2) \cup (B_1 \leqslant_{\beta_{sd}} B_2) \cup \mathbb{S} \\
 & & \textbf{where } A_2[x := B_2] = C \\
 & & \textbf{and } x \textbf{ fresh variable } x \in \mathrm{fv}(A_2) \\
 & & \textbf{and } A_2,\, B_2\ \beta\textbf{-normal}
\end{array}
$$

Figure 1: Matching modulo superdevelopments

4. Algorithm for matching modulo superdevelopments

In this section, we present an algorithm for matching modulo superdevelopments in the pure λ-calculus. We illustrate it on examples and we finally study its main properties.

4.1. **Presentation of the algorithm.** Rules for matching modulo superdevelopments are given in Figure 1 using transformation rules [MM82, Kir84, SG89, JK91]. By transformation rules, we mean rewriting rules only applied at the top position. Then,

- a system is transformed step by step until getting to a normal from (the set of rules is terminating); this normal form can be a resolved form and then gives a solution to the original matching problem (the algorithm is sound);
- by exploring all the possible reductions (the application of rules is non deterministic meaning that two reductions of the same initial system may lead to different normal forms) and collecting all the resolved forms we get a complete match set (the algorithm is sound and complete).

We denote $\mathbb{S} \rightarrow \mathbb{S}'$ when one of the transformation rules given in Figure 1 can be applied to transform $\mathbb{S}$ into $\mathbb{S}'$. We also denote $\mathbb{S} \twoheadrightarrow \mathbb{S}'$ when it exists $n \geqslant 0$ systems $\mathbb{S}_1, \ldots, \mathbb{S}_n$ such that $\mathbb{S} = \mathbb{S}_1 \rightarrow \ldots \rightarrow \mathbb{S}_n = \mathbb{S}'$.

The transformation rules of Figure 1 mimicked the definition of strong parallel reduction as we explain now in details by examining transformations rules one by one.

The ε rules deal with atoms. The rules (ε_c) and (ε_v) are removing directly solved equations. Note that when the rules (ε_c) is applied to the singleton system $a \leqslant_{\beta_{sd}} a$ then the resulting system is the empty multiset.

The rule (ε_X) is substituting a matching variable with its corresponding value. Note that we substitute only $\mathcal{X}$-closed terms. For the reader familiar with 'classical' presentation of higher-order matching algorithms, it may be useful to recall again that we do not consider normalizing substitutions. This is not a matter of taste but rather a matter of soundness. In fact, let us consider the following variation of the ε_X rule

$$(X \leqslant_{\beta_{sd}} A) \cup \mathbb{S} \quad \xrightarrow[\varepsilon_X^{\downarrow}]{} \quad (X \leqslant_{\beta_{sd}} A) \cup (\mathbb{S}\{A/X\}) \downarrow$$

where $(\mathbb{S}\{A/X\}) \downarrow$ denotes the β_{sd}-normal form of $\mathbb{S}\{A/X\}$

and the following equation

$$f(XYZ, X, Y, Z) \leqslant_{\beta_{sd}} f(1, \lambda x.\, \lambda y.\, xy, \lambda z.\, z, 1)$$

which can be transformed into

$$(XYZ \leqslant_{\beta_{sd}} 1) \cup (X \leqslant_{\beta_{sd}} \lambda x.\, \lambda y.\, xy) \cup (Y \leqslant_{\beta_{sd}} \lambda z.\, z) \cup (Z \leqslant_{\beta_{sd}} 1)$$

By applying the $(\varepsilon_X^{\downarrow})$ rule three times and the (ε_c) rule once we get:

$$(X \leqslant_{\beta_{sd}} \lambda x.\, \lambda y.\, xy) \cup (Y \leqslant_{\beta_{sd}} \lambda z.\, z) \cup (Z \leqslant_{\beta_{sd}} 1)$$

We obtain a normal form in solved form whereas the corresponding substitution is not a match modulo superdevelopments (even if this is of course a match modulo β): the rule $(\varepsilon_X^{\downarrow})$ is thus not sound.

Finally, note that the side condition $X \in \mathbb{S}$ of the (ε_X) rule is used to guarantee the termination of the algorithm (see the proof of Proposition 4.8).

The (λ_λ) rule is dealing with λ-abstractions like the (Red-λ) does. Note that this way to deal with abstractions is sound since we only consider closed substitutions. In many higher-order matching algorithms, the λ-abstractions are kept prenex. The two choices are possible. We will present the algorithm for matching modulo superdevelopments using the second choice. A similar rule to (λ_λ) can be found in the context of higher-order unification in the λ-calculus with de Bruijn indices and explicit substitutions [DHK00].

The @ rules deal with the application. The rule $(@_@)$ is directly related to the rule (Red-@) and thus requires no more comments. The rules $(@_\pi)$ and $(@_\beta)$ are both related to the rule (Red-β_s). We are trying to express the right hand side C of the equation as the result of a β-reduction, let us say $A_2[x := B_2]$. Depending on the belonging of x in A_2, we obtain either the rule $(@_\pi)$ or the rule $(@_\beta)$.

- If x does not belong to A_2, we obtain the rule $(@_\pi)$: we associate the left hand side with an abstraction ignoring its argument and returning the right hand side of the initial equation.
- If not, that is if x belongs to A_2, then we obtain the $(@_\beta)$ rule by mimicking the (Red-β_s) rule for all terms such that $C = A_2[x := B_2]$ where x belongs to A_2 and A_2 and B_2 are β-normal. Let us examine how we can find such terms. First, remark that B_2 is necessarily a subterm of C (since x belongs to A_2). Let us consider one of this subterm. We choose a subset of the set of positions of C such that the subterm of C at these positions is B_2. The term A_2 is obtained from C by replacing the

subterm at each of the position of the chosen subset by x. Note that there exists only a finite number of such pairs (A_2, B_2) which satisfy these conditions.

Comparing with the approach like the one presented in [HL78], we do not introduce new matching variables during the matching process. The solutions of a system $\mathbb{S}$ given by our algorithm have then a domain included in the matching variables of $\mathbb{S}$. The advantage is that when we compare such a solution with an arbitrary solution of $\mathbb{S}$, it is not technically necessary to restrict this comparison to matching variables of $\mathbb{S}$ unlike for example in [Bür90].

Example 4.1 (Computing the solutions of a β_{sd}-equation)**.** We consider the equation $XY \leqslant_{\beta_{sd}} ab$. As the left and right hand sides of the equation are applications, we can apply the rules $(@_@)$, $(@_\pi)$ or $(@_\beta)$.

(1) Rule $(@_@)$:
$$(XY \leqslant_{\beta_{sd}} ab) \to (X \leqslant_{\beta_{sd}} a) \cup (Y \leqslant_{\beta_{sd}} b).$$
(2) Rule $(@_\pi)$:
$$(XY \leqslant_{\beta_{sd}} ab) \to (X \leqslant_{\beta_{sd}} \lambda x.\, ab).$$
(3) Rule $(@_\beta)$: to find A_1 and A_2 such that $A_1[x := A_2] = ab$, we first choose A_2 as a subterm of "ab": a, b and ab. The set of positions of ab for which A_2 is the corresponding subterm of C is a singleton since each subterm of ab appears only once in ab. We obtain three ways to apply the rule $(@_\beta)$ corresponding to the three subterms of the right hand side of the equation $XY \leqslant_{\beta_{sd}} ab$:
   (a) $(XY \leqslant_{\beta_{sd}} ab) \to (X \leqslant_{\beta_{sd}} \lambda x.\, xb) \cup Y \leqslant_{\beta_{sd}} a)$.
   (b) $(XY \leqslant_{\beta_{sd}} ab) \to (X \leqslant_{\beta_{sd}} \lambda x.\, ax) \cup (Y \leqslant_{\beta_{sd}} b)$.
   (c) $(XY \leqslant_{\beta_{sd}} ab) \to (X \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (Y \leqslant_{\beta_{sd}} ab)$.

Example 4.2 (Computing the solutions of a β_{sd}-equation)**.** We consider the equation $X(YX) \leqslant_{\beta_{sd}} a$. We can apply the rules $(@_\pi)$ or $(@_\beta)$.

(1) Rule $(@_\pi)$:
$$(X(YX) \leqslant_{\beta_{sd}} a) \to (X \leqslant_{\beta_{sd}} \lambda x.\, a).$$
(2) Rule $(@_\beta)$:
$$(X(YX) \leqslant_{\beta_{sd}} a) \to (X \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (YX \leqslant_{\beta_{sd}} a).$$
To simplify $YX \leqslant_{\beta_{sd}} a$ we can apply the rule $(@_\pi)$ or the rule $(@_\beta)$.
   (a) Rule $(@_\pi)$:
$$(X \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (YX \leqslant_{\beta_{sd}} a) \to (X \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (Y \leqslant_{\beta_{sd}} \lambda x.\, a).$$
   (b) Rule $(@_\beta)$:
$$\begin{array}{ll} & (X \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (YX \leqslant_{\beta_{sd}} a) \\ \to & (X \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (Y \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (X \leqslant_{\beta_{sd}} a) \\ \to & (X \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (Y \leqslant_{\beta_{sd}} \lambda x.\, x) \cup (\lambda x.\, x \leqslant_{\beta_{sd}} a). \end{array}$$

In the last case, the system is not in solved form (even if it is in normal form) and thus does not give a solution. The initial matching problem has thus only two solutions.

Remark 4.3 (Application of the $(@_\beta)$ rule)**.** The application of the rule $(@_\beta)$ is driven by the choice of the term B_2 which must match the term B_1. In implementations of the algorithm, this restriction on the choice of B_2 is very useful. For example, if B_1 is a constant or a variable then necessarily $B_2 = B_1$. This is always the case in the case of Miller patterns since a matching variable can only be applied to bound variables.

4.2. **Termination property.** We show that the algorithm is terminating. We first define the size of a term (*resp.* of a matching equation, *resp.* of a system).

Definition 4.4 (Size)**.** The size of a term A denoted $\mathfrak{S}(A)$ is defined by induction

$$\begin{array}{lcll} \mathfrak{S}(\varepsilon) & = & 1 & \text{for all atoms } \varepsilon\ \varepsilon \\ \mathfrak{S}(\lambda x.\, B) & = & \mathfrak{S}(B) + 1 & \\ \mathfrak{S}(BC) & = & \mathfrak{S}(B) + \mathfrak{S}(C) + 1 & \end{array}$$

The size of a matching equation $A \leqslant_{\beta_{sd}} B$ is the size of A. The size of a system is the sum of the sizes of each equation of the system. We use the same notation for the size of terms, equations and systems.

For every system $\mathbb{S}$, we denote by $\mathfrak{U}(\mathbb{S})$ *the number of unsolved variables* of $\mathbb{S}$ in the sense of the following definition.

Definition 4.5 (Solved variable)**.** A matching variable of an equation $X \leqslant_{\beta_{sd}} A$ belonging to a system $\mathbb{S}$ is a solved variable if X occurs nowhere else in $\mathbb{S}$.

Lemma 4.6 (Unsolved variables)**.** *For all systems $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$ we have the following inequality*

$$\mathfrak{U}(\mathbb{S}) \geqslant \mathfrak{U}(\mathbb{S}')$$

The inequality is strict if the reduction is done using the rule (ε_X).

Lemma 4.7 (Size decreasing)**.** *For all systems $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$ using any of the transformation rules of Figure 1 except the rule (ε_X), we have*

$$\mathfrak{S}(\mathbb{S}) > \mathfrak{S}(\mathbb{S}')$$

Proof. We prove that the inequality is valid for each transformation rule. Note that the size only depends on the right hand side of equations: the inequality is thus in particular true for rule ($@_\beta$).

$$\begin{array}{ll} \varepsilon_v & \mathfrak{S}((x \leqslant_{\beta_{sd}} x) \cup \mathbb{S}) \\ & = \mathfrak{S}(x \leqslant_{\beta_{sd}} x) + \mathfrak{S}(\mathbb{S}) \\ & > \mathfrak{S}(\mathbb{S}) \\ \\ \varepsilon_c & \mathfrak{S}((c \leqslant_{\beta_{sd}} c) \cup \mathbb{S}) \\ & = \mathfrak{S}(c \leqslant_{\beta_{sd}} c) + \mathfrak{S}(\mathbb{S}) \\ & > \mathfrak{S}(\mathbb{S}) \\ \\ \lambda_\lambda & \mathfrak{S}((\lambda x.\, A \leqslant_{\beta_{sd}} \lambda x.\, B) \cup \mathbb{S}) \\ & = \mathfrak{S}(\lambda x.\, A \leqslant_{\beta_{sd}} \lambda x.\, B) + \mathfrak{S}(\mathbb{S}) \\ & = 1 + \mathfrak{S}(A) + \mathfrak{S}(\mathbb{S}) \\ & > \mathfrak{S}(A \leqslant_{\beta_{sd}} B) + \mathfrak{S}(\mathbb{S}) \end{array}$$

$$
\begin{array}{ll}
@_@ & \mathfrak{S}\left((A_1B_1 \leqslant_{\beta_{sd}} A_2B_2) \cup \mathbb{S}\right) \\
& = \mathfrak{S}\left(A_1B_1\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
& = 1 + \mathfrak{S}\left(A_1\right) + \mathfrak{S}\left(B_1\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
& > \mathfrak{S}\left(A_1 \leqslant_{\beta_{sd}} A_2\right) + \mathfrak{S}\left(B_1 \leqslant_{\beta_{sd}} B_2\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
@_\pi & \mathfrak{S}\left((A_1B_1 \leqslant_{\beta_{sd}} A_2) \cup \mathbb{S}\right) \\
& = \mathfrak{S}\left(A_1B_1\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
& = 1 + \mathfrak{S}\left(A_1\right) + \mathfrak{S}\left(B_1\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
& > \mathfrak{S}\left(A_1 \leqslant_{\beta_{sd}} \lambda x.\, A_2\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
@_\beta & \mathfrak{S}\left((A_1B_1 \leqslant_{\beta_{sd}} C) \cup \mathbb{S}\right) \\
& = \mathfrak{S}\left(A_1B_1\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
& = 1 + \mathfrak{S}\left(A_1\right) + \mathfrak{S}\left(B_1\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
& > \mathfrak{S}\left(A_1 \leqslant_{\beta_{sd}} \lambda x.\, A_2\right) + \mathfrak{S}\left(B_1 \leqslant_{\beta_{sd}} B_2\right) + \mathfrak{S}\left(\mathbb{S}\right) \\
& \text{for all normal terms } A_2 \text{ and } B_2
\end{array}
$$

□

Proposition 4.8 (Termination of the algorithm)**.** *The set of transformation rules given in Figure 1 is terminating.*

Proof. For all systems $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$, the lexical product of the number of unsolved variables $\mathfrak{U}\left(\_\right)$ and the size of the system $\mathfrak{S}\left(\_\right)$ decreases for each rule:

	$\mathfrak{U}\left(\_\right)$	$\mathfrak{S}\left(\_\right)$
ε_v	$=$	$>$
ε_c	$=$	$>$
λ_λ	$\geqslant$	$>$
ε_X	$>$	
$@_@$	$\geqslant$	$>$
$@_\pi$	$\geqslant$	$>$
$@_\beta$	$\geqslant$	$>$

Note that the rule (ε_X) makes decrease the number of unsolved variables thanks to the side condition $X \in \mathbb{S}$. □

4.3. **Completeness property.** We define an extension of the relation $\Rightarrow_{\beta_{sd}}$ to multisets. To ease the reading in this section, we write $(A_1, B_1) \in \beta_s$ for $A_1 \Rightarrow_{\beta_{sd}} B_1$.

Definition 4.9 (Multiset extension of $\Rightarrow_{\beta_{sd}}$)**.** We note $\emptyset$ the empty multiset. The multiset extension of $\Rightarrow_{\beta_{sd}}$ is defined by

$$\frac{}{\emptyset \in \beta_s} \qquad\qquad \frac{\dfrac{}{(A_1, B_1) \in \beta_s} \quad E \in \beta_s}{(A_1, B_1) \cup E \in \beta_s}$$

$$\frac{\dfrac{(A_1', B_1') \in \beta_s \quad (A_1'', B_1'') \in \beta_s}{(A_1, B_1) \in \beta_s} \qquad (A_1', B_1') \cup (A_1'', B_1'') \cup E \in \beta_s}{(A_1, B_1) \cup E \in \beta_s}$$

$$\frac{\dfrac{(A_1', B_1') \in \beta_s}{(A_1, B_1) \in \beta_s} \quad (A_1', B_1') \cup E \in \beta_s}{(A_1, B_1) \cup E \in \beta_s}$$

Proposition 4.10 (Completeness)**.** *For every system $\mathbb{S}$, if $\varphi \in \mathbb{M}(\mathbb{S})$ then there exists a sequence of transformations*

$$\mathbb{S} = \mathbb{S}_0 \to \mathbb{S}_1 \to \ldots \to \mathbb{S}_n$$

where $\mathbb{S}_n$ is in solved form and $\sigma_{\mathbb{S}_n} \leq \varphi$.

Proof. We suppose given a system $\mathbb{S}_0$ such that $\varphi \in \mathbb{M}(\mathbb{S}_0)$. We want to show that there exists a derivation such that

$$\mathbb{S} = \mathbb{S}_0 \to \mathbb{S}_1 \to \ldots \to \mathbb{S}_n$$

where $\mathbb{S}_n$ is in solved form and $\sigma_{\mathbb{S}_n} \leq \varphi$. Let $\mathbb{S}_0 = (A_1 \leqslant_{\beta_{sd}} B_1) \cup \ldots\ (A_p \leqslant_{\beta_{sd}} B_p)$ and let E_0 be the multiset defined by

$$E_0 \quad = \quad (A_1\varphi, B_1) \cup \ldots \cup (A_p\varphi, B_p)$$

Note that $\varphi \in \mathbb{M}(\mathbb{S}_0)$ is equivalent to $E_0 \in \beta_s$. We show the proposition by induction on $E_0 \in \beta_s$.

(1) If $E_0 = \emptyset$ then $\mathbb{S}_0 = \emptyset$ and $\sigma_{\mathbb{S}_0}$ is the identity substitution `id`. The result is obvious since for every subsection φ we have $\mathtt{id} \leqslant \varphi$.

(2) If $E_0 = (A_1\varphi, B_1) \cup E \in \beta_s$ with $(A_1\varphi, B_1) \in \beta_s$ is proved without hypothesis and $E \in \beta_s$. By induction hypothesis, there exists a derivation (D)

$$\begin{array}{l} \mathbb{S}_0' = (A_2 \leqslant_{\beta_{sd}} B_2) \cup \ldots \cup\ (A_p \leqslant_{\beta_{sd}} B_p) \\ \to \ldots \\ \to \mathbb{S}_n' \end{array}$$

such that $\sigma_{\mathbb{S}_n'} \leqslant \varphi$. Thus if $X \in \mathcal{D}om(\sigma_n')$ then $X\sigma_n' = B_1$.

Suppose first that A_1 is a constant or a variable. Then the derivation

$$\begin{array}{ll} \mathbb{S}_0 & \to_{\substack{\varepsilon_c \\ \varepsilon_v}} (A_2 \leqslant_{\beta_{sd}} B_2) \cup \ldots \cup\ (A_p \leqslant_{\beta_{sd}} B_p) \\ & \to \ldots \\ & \to \mathbb{S}_n' \end{array}$$

565 is the desired one.

Suppose now that A_1 is a matching variable. Then if there is no reduction step in the derivation (D) where X is substituted (application of the rule (ε_X) to the matching variable X), the derivation

$$\begin{array}{ll} & (X \leqslant_{\beta_{sd}} B_1) \cup (A_2 \leqslant_{\beta_{sd}} B_2) \cup \ \dots \ \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to & \dots \\ \to & (X \leqslant_{\beta_{sd}} B_1) \cup \mathbb{S}'_n \end{array}$$

566 is the desired one.

In the other cases (that is to say if there exists a derivation i_0 in (D) where X is instantiated) then X must be instantiated by B_1 and the following derivation is the desired one:

$$\begin{array}{ll} & (X \leqslant_{\beta_{sd}} B_1) \cup (A_2 \leqslant_{\beta_{sd}} B_2) \cup \ \dots \ \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to & \dots \\ \to_{i_0} & (B_1 \leqslant_{\beta_{sd}} B_1) \cup \mathbb{S}'_{i_0+1} \\ \twoheadrightarrow & \mathbb{S}'_{i_0+1} \\ \to & \dots \\ \to & \mathbb{S}'_n \end{array}$$

567 (3) If $E_0 = (A_1\varphi, B_1) \cup E \in \beta_s$ with $(A_1\varphi, B_1) \in \beta_s$ is provable using two hypotheses.
568 We analyze the last rule used in the proof of $(A_1\varphi, B_1) \in \beta_s$.

- **Rule (**$Red - @$**)** Then we have $E_0 = (A_1^1\varphi A_1^2\varphi, B_1 B_2) \cup E \in \beta_s$ with

$$\frac{(A_1^1\varphi, B_1^1) \in \beta_s \quad (A_1^2\varphi, B_1^2) \in \beta_s}{(A_1^1\varphi A_1^2\varphi, B_1^1 B_1^2) \in \beta_s}$$

and

$$(A_1^1\varphi, B_1^1) \cup (A_1^2\varphi, B_1^2) \cup E \in \beta_s$$

By induction hypothesis, there exists a derivation such that

$$\begin{array}{ll} & (A_1^1 \leqslant_{\beta_{sd}} B_1^1) \cup (A_1^2 \leqslant_{\beta_{sd}} B_1^2) \cup (A_2 \leqslant_{\beta_{sd}} B_2) \cup \dots \ \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to & \dots \\ \to & \mathbb{S}_n \end{array}$$

569 with $\sigma_{\mathbb{S}_n} \leqslant \varphi$.

If $A_1 = X$ we also have (since then $A_1^1 = B_1^1$ and $A_1^2 = B_1^2$ and thus the two corresponding equations are $\mathcal{V}$-closed and do not influence the derivation)

$$\begin{array}{l} (A_2 \leqslant_{\beta_{sd}} B_2) \cup \dots \ \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to \dots \\ \to \mathbb{S}'_n \end{array}$$

570 such that $\sigma_{\mathbb{S}'_n} \leqslant \varphi$. We conclude like in the previous case.

If not then the derivation

$$\begin{array}{ll} & (A_1 \leqslant_{\beta_{sd}} B_1) \cup \ \dots \ \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ = & (A_1^1 A_1^2 \leqslant_{\beta_{sd}} B_1^1 B_1^2) \cup (A_2 \leqslant_{\beta_{sd}} B_2) \cup \ \dots \ \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to & (A_1^1 \leqslant_{\beta_{sd}} B_1^1) \cup \ (A_1^2 \leqslant_{\beta_{sd}} B_1^2) \cup \dots \ \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to & \dots \\ \to & \mathbb{S}_n \end{array}$$

571 is the desired one.

- **Rule** $(Red - \beta_s)$ Then we have $E_0 = (A_1^1 \varphi A_1^2 \varphi, B_1^1[x := B_1^2]) \cup E$ with
$$\frac{(A_1^1\varphi, \lambda x.\, B_1^1) \in \beta_s \quad (A_1^2\varphi, B_1^2) \in \beta_s}{(A_1^1\varphi A_1^2\varphi, B_1^1[x := B_1^2]) \in \beta_s}$$
and
$$(A_1^1\varphi, \lambda x.\, B_1^1) \cup (A_1^2\varphi, B_1^2) \cup E \in \beta_s$$
By induction hypothesis, there exists a derivation such that
$$\begin{array}{ll} & (A_1^1 \leqslant_{\beta_{sd}} \lambda x.\, B_1^1) \cup (A_1^2 \leqslant_{\beta_{sd}} B_1^2) \cup \ldots \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to & \ldots \\ \to & \mathbb{S}'_n \end{array}$$
with $\sigma'_{\mathbb{S}_n} \leqslant \varphi$. The derivation
$$\begin{array}{ll} & (A_1^1 A_1^2 \leqslant_{\beta_{sd}} B_1^1[x := B_1^2]) \cup (A_1^1 \leqslant_{\beta_{sd}} \lambda x.\, B_1^1) \cup (A_1^2 \leqslant_{\beta_{sd}} B_1^2) \\ & \qquad \cup \ldots \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to & \ldots \\ \to & \mathbb{S}_n \end{array}$$
is the desired one if $x \in \mathrm{fv}(B_1^1)$. If not, the derivation
$$\begin{array}{ll} & (A_1^1 A_1^2 \leqslant_{\beta_{sd}} B_1^1[x := B_1^2]) \cup (A_1^1 \leqslant_{\beta_{sd}} \lambda x.\, B_1^1) \\ & \qquad \cup \ldots \cup (A_p \leqslant_{\beta_{sd}} B_p) \\ \to & \ldots \\ \to & \mathbb{S}_n \end{array}$$
is the desired one

(4) This case is similar to the previous one.

□

4.4. Correctness property.

Lemma 4.11. *For all systems $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$ using the rules (ε_c), (ε_v), (λ_λ) or (ε_X), we have $\mathbb{M}(\mathbb{S}') = \mathbb{M}(\mathbb{S})$.*

Proof. The only trivial case concerns the rule (ε_X). Let X be a matching variable, $\mathbb{S}$ be a system and A be a term such that $\mathrm{fv}(A) = \emptyset$ and $X \in \mathbb{S}$.
$$\begin{array}{rcl} \varphi \in \mathbb{M}((X \leqslant_{\beta_{sd}} A) \cup \mathbb{S}) & \Leftrightarrow & X\varphi \Rightarrow_{\beta_{sd}} A \text{ and } \varphi \in \mathbb{M}(\mathbb{S}) \\ & \overset{\varphi \text{ normal}}{\Leftrightarrow} & X\varphi = A \text{and } \varphi \circ \ \{A/X\} \in \mathbb{M}(\mathbb{S}) \\ & \Leftrightarrow & X\varphi \Rightarrow_{\beta_{sd}} A \text{ and } \varphi \in \mathbb{M}(\mathbb{S}\{A/X\}) \\ & \Leftrightarrow & \varphi \in \mathbb{M}((X \leqslant_{\beta_{sd}} A) \cup \mathbb{S}\{A/X\}) \end{array}$$

□

Lemma 4.12. *For all systems $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$ using the rules ($@_@$), ($@_\pi$) or ($@_\beta$) we have $\mathbb{M}(\mathbb{S}') \subseteq \mathbb{M}(\mathbb{S})$.*

Proof. We prove the result for each rule.

$$
\begin{array}{ll}
@_@ & \varphi \in \mathbb{M}((A_1 \leqslant_{\beta_{sd}} A_2) \cup (B_1 \leqslant_{\beta_{sd}} B_2) \cup \mathbb{S}) \\
& \iff \varphi \in \mathbb{M}(A_1 \leqslant_{\beta_{sd}} A_2) \text{ and } \varphi \in \mathbb{M}(B_1 \leqslant_{\beta_{sd}} B_2) \text{ and } \varphi \in \mathbb{M}(\mathbb{S}) \\
& \iff A_1\varphi \Rightarrow_{\beta_{sd}} A_2 \text{ and } B_1\varphi \Rightarrow_{\beta_{sd}} B_2 \text{ and } \varphi \in \mathbb{M}(\mathbb{S}) \\
& \implies (A_1B_1)\varphi \Rightarrow_{\beta_{sd}} A_2B_2 \text{ and } \varphi \in \mathbb{M}(\mathbb{S}) \\
& \iff \varphi \in \mathbb{M}((A_1B_2 \leqslant_{\beta_{sd}} A_2B_2) \cup \mathbb{S})
\end{array}
$$

$$
\begin{array}{ll}
@_\pi & \varphi \in \mathbb{M}((A_1 \leqslant_{\beta_{sd}} \lambda x.\, A_2) \cup \mathbb{S}) \\
& \iff A_1\varphi \Rightarrow_{\beta_{sd}} \lambda x.\, A_2 \text{ and } \varphi \in \mathbb{M}(S) \\
& \implies (A_1B_1)\varphi \Rightarrow_{\beta_{sd}} A_2[x := (B_1\varphi)] \text{ and } \varphi \in \mathbb{M}(S) \\
& \iff (A_1B_1)\varphi \Rightarrow_{\beta_{sd}} A_2 \text{ and } \varphi \in \mathbb{M}(S) \\
& \text{since } x \text{ is fresh from } A_1, B_1, A_2, \mathbb{S} \\
& \iff \varphi \in \mathbb{M}((A_1B_1 \leqslant_{\beta_{sd}} A_2) \cup \mathbb{S})
\end{array}
$$

$$
\begin{array}{ll}
@_\beta & \varphi \in \mathbb{M}((A_1 \leqslant_{\beta_{sd}} \lambda x.\, A_2) \cup (B_1 \leqslant_{\beta_{sd}} B_2) \cup \mathbb{S}) \\
& \iff A_1\varphi \Rightarrow_{\beta_{sd}} \lambda x.\, A_2 \text{ and } B_1\varphi \Rightarrow_{\beta_{sd}} B_2 \text{ and } \varphi \in \mathbb{S} \\
& \implies (A_1B_1)\varphi \Rightarrow_{\beta_{sd}} A_2[x := B_1] \text{ and } \varphi \in \mathbb{S} \\
& \iff \varphi \in \mathbb{M}((A_1B_1 \leqslant_{\beta_{sd}} A_2[x := B_1]) \cup \mathbb{S})
\end{array}
$$

□

Proposition 4.13 (Correctness)**.** *For all systems $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \twoheadrightarrow \mathbb{S}'$ and $\mathbb{S}'$ is in solved form we have $\sigma_{\mathbb{S}'} \in \mathbb{M}(\mathbb{S})$.*

Proof. The proof is a simple induction on the length of the sequence of transformations and is using the previous lemmas for the induction step. □

4.5. **Finite complete match set property.** The correctness and completeness properties entail that there exists a complete match set. It is obtained by exploring all the possible reductions and by constructing the set of substitutions obtained from the solved forms. We know that this process is finite since the algorithm terminates and thus the complete match set is also finite.

Proposition 4.14 (Finite complete match set)**.** *For a given system $\mathbb{S}$, there exists a finite complete match set $\mathbb{M}$ given by*

$$\mathbb{M} = \{\sigma'_{\mathbb{S}} \mid \mathbb{S} \twoheadrightarrow \mathbb{S}' \textit{ and } \mathbb{S}' \textit{ is in solved form}\}\,.$$

5. Algorithm for matching modulo superdevelopments and eta

5.1. **Presentation of the algorithm.** The algorithm for matching modulo superdevelopments given in Figure 1 must be customized to take into account η-conversion. First,

$$
\begin{array}{lll}
(\lambda\overline{x_n}.\, x \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, x) \cup \mathbb{S} & \to_{\varepsilon_v} & \mathbb{S} \\
(\lambda\overline{x_n}.\, a \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, a) \cup \mathbb{S} & \to_{\varepsilon_c} & \mathbb{S} \\
(\lambda\overline{x_n}.\, X \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, A) \cup \mathbb{S} & \to_{\varepsilon_X} & (\lambda\overline{x_n}.\, X \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, A) \cup \mathbb{S}\{A/X\} \\
 & & \textbf{if } \mathrm{fv}(A) = \emptyset \textbf{ and } X \in \mathbb{S} \\
 & & \\
(\lambda\overline{x_n}.\, A \leqslant_{\beta_{sd}} \lambda\overline{x_{n-k}}.\, B) \cup \mathbb{S} & \to_{\lambda_{\_}} & (\lambda\overline{x_n}.\, A \leqslant_{\beta_{sd}} \lambda\overline{x_{n-k+1}}.\, Bx_{n-k+1}) \cup \mathbb{S} \\
 & & \\
(\lambda\overline{x_n}.\, (A_1B_1) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, (A_2B_2)) \cup \mathbb{S} & \to_{@_@} & (\lambda\overline{x_n}.\, A_1 \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, A_2) \cup \\
 & & \qquad (\lambda\overline{x_n}.\, B_1 \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B_2) \cup \mathbb{S} \\
(\lambda\overline{x_n}.\, (A_1B_1) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, C) \cup \mathbb{S} & \to_{@_\pi} & (\lambda\overline{x_n}.\, A_1 \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, (\lambda y.\, C)) \cup \mathbb{S} \\
 & & \textbf{where } y \textbf{ fresh variable} \\
 & & \\
(\lambda\overline{x_n}.\, (A_1B_1) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, C) \cup \mathbb{S} & \to_{@_\beta} & (\lambda\overline{x_n}.\, A_1 \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, (\lambda x.\, A_2)) \cup \\
 & & \qquad (\lambda\overline{x_n}.\, B_1 \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B_2) \cup \mathbb{S} \\
 & & \textbf{where } A_2[x := B_2] = C \\
 & & \textbf{and } x \textbf{ fresh variable},\ x \in \mathrm{fv}(A_2) \\
 & & \textbf{and } \lambda x.\, A_2,\ B_2\ \beta\eta\textbf{-normal}
\end{array}
$$

Figure 2: Transformation for matching modulo superdevelopments and η

η-expansion is done on the fly using the following rule

$$
\begin{array}{lll}
(\lambda x.\, A \leqslant^{\eta}_{\beta_{sd}} B) \cup \mathbb{S} & \to & (A \leqslant^{\eta}_{\beta_{sd}} Bx) \cup \mathbb{S} \\
 & & \text{if } B \text{ is not } \lambda\text{-abstraction} \\
 & & \text{ and } x \text{ is a fresh variable}
\end{array}
$$

This rule replaces the right hand side B by $\lambda x.\, Bx$ and eliminates (like in the rule (λ_λ) of Figure 1) one head λ-abstraction. Then, we must add a side condition to the rule $(@_\beta)$ for $\lambda x.\, A_2$ and A_1 to be in $\beta\eta$-normal form (and not only in β-normal form).

By making these two changes, we obtain an algorithm for matching modulo superdevelopments and η. We give it explicitly in Figure 2. We have chosen to present this algorithm in a slightly different way: instead of removing λ-abstractions, we keep them prenex (and thus the rule (λ_λ) is no more useful).

We can prove that this algorithm has the termination, soundness and completeness property. The termination and the correctness proofs are similar to the one given in the previous section. The completeness proof is slightly more technical although it is fundamentally the same as the one given in Section 4.3.

Example 5.1 ($\beta_{sd}\eta$-matches)**.** Consider the equation given in Example 4.1. If we solve this equation modulo $\beta_{sd}\eta$ we obtain only 4 solutions. In fact, the two solutions

$$(X \leqslant_{\beta_{sd}} a) \cup (Y \leqslant_{\beta_{sd}} b) \qquad \text{and} \qquad (X \leqslant_{\beta_{sd}} \lambda x.\, ax) \cup (Y \leqslant_{\beta_{sd}} b)$$

are η-equivalents.

Example 5.2 (Solutions of a $\beta_{sd}\eta$-equation)**.** Consider the pair of terms $(\lambda x.\, X(Yx), a)$. This pair is a β_{sd}-equation with no β_{sd}-match. But this equation has two $\beta_{sd}\eta$-matches given by

$$\{a/X, \lambda z.\, z/Y\} \qquad \text{and} \qquad \{\lambda z.\, z/X, a/Y\}.$$

In fact,

$$\begin{array}{lcl}
(\lambda x.\,(X(Yx) \leqslant^{\eta}_{\beta_{sd}} a) & \to & (\lambda x.\, X(Yx) \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, ax) \\
& \to & (\lambda x.\, X \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, a) \cup (\lambda x.\, Yx \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, x) \\
& \to & (\lambda x.\, X \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, a) \cup (\lambda x.\, Y \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, \lambda z.\, z)
\end{array}$$

$$\begin{array}{lcl}
(\lambda x.\, X(Yx) \leqslant^{\eta}_{\beta_{sd}} a) & \to & (\lambda x.\, X(Yx) \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, ax) \\
& \to & (\lambda x.\, X \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, \lambda z.\, z) \cup (\lambda x.\, Yx \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, ax) \\
& \to & (\lambda x.\, X \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, \lambda z.\, z) \cup (\lambda x.\, Y \leqslant^{\eta}_{\beta_{sd}} \lambda x.\, a)
\end{array}$$

5.2. **Minimality for Miller patterns.** In Section 3.4, we show that matching modulo superdevelopments is complete for matching of Miller patterns. In this section, by Miller pattern equations we mean a β_{sd}-equation whose first term is a Miller pattern.

We show that the algorithm for matching modulo superdevelopments and η given in Figure 2 gives at most one solution when it is applied to a Miller pattern equation. Since the algorithm is sound and complete this gives the most general match. First, note that all the reducts of a system of Miller pattern equations are systems of Miller pattern equations.

Proposition 5.3. *Let $P \leqslant^{\eta}_{\beta_{sd}} B$ be a Miller pattern equation which has a solution and let A be a typed term. Then the set of solved forms given by the transformation rules of Figure 2 is a singleton.*

Proof. First, we can remark that when the rule $(\lambda\_)$ can be applied, it is the only one and thus does not introduce any non-determinism in the application of the transformation rules. We thus exclude it in the remaining of this proof. We prove the result by induction on the size of patterns. We distinguish three cases according to the head symbol of the pattern.

(1) If $P = \lambda\overline{x_n}.\, f(A_1, \ldots, A_n)$ then for the equation to be a solution, the term B must be of the form $B = \lambda\overline{x_n}.\, f(B_1, \ldots, B_p)$. All derivations that lead to a solution must reduce $P \leqslant_{\beta_{sd}} B$ into

$$(\lambda\overline{x_n}.\, A_1 \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B_1) \cup \ldots \cup (\lambda\overline{x_n}.\, A_p \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B_p)$$

(modulo the application order of the rules).

Since $\lambda\overline{x_n}.\, A_1, \ldots \lambda\overline{x_n}.\, A_p$ are Miller patterns we can apply the induction hypothesis and conclude that

$$\begin{array}{ll}
\lambda\overline{x_n}.\, A_1 \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B_1 & \text{has } \sigma_1 \text{ as a unique solution} \\
\ldots & \\
\lambda\overline{x_n}.\, A_p \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B_p & \text{has } \sigma_p \text{ as a unique solution}
\end{array}$$

then the initial problem has a unique solution $\sigma_1 \cup \ldots \cup \sigma_p$.

(2) The case $P = \lambda\overline{x_n}.\, x(A_1, \ldots, A_n)$ is similar to the previous one.

(3) If $P = \lambda\overline{x_n}.\, X(x_{i_1}, \ldots, x_{i_p})$ where the variables $x_{i_1}, \ldots, x_{i_p}$ are pairwise distinct. Then three rules $(@_\pi)$, $(@_@)$ and $(@_\beta)$ can be applied. We show that in all cases only one of this choice leads to a solution. In the remaining of this proof, we will always use the fact that if the application of the rule $(@_@)$ leads to a solution then we must have $B = B_1 x_{i_p}$. In the same way, if the rule $(@_\beta)$ leads to a solution then

$$\begin{array}{rcl}(\lambda\overline{x_n}.\, X(x_{i_1}, \ldots, x_{i_p}) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B) & \rightarrow & (\lambda\overline{x_n}.\, X(x_{i_1}, \ldots, x_{i_{p-1}}) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, \lambda z.\, C_1 \cup) \\ & & (\lambda\overline{x_n}.\, x_{i_p} \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, x_{i_p}) \\ & & \text{with } B = C_1[z := x_{i_p}] \text{ and } z \in \mathrm{fv}(C_1)\end{array}$$

and thus $x_{i_p} \in \mathrm{fv}(B)$.

Let us suppose first that the application of the rule $(@_\pi)$ leads to a solution. We show that neither the application of the rule $(@_@)$ nor the application of the rule $(@_\beta)$ lead to a solution. We have

$$\begin{array}{l}(\lambda\overline{x_n}.\, X(x_{i_1}, \ldots, x_{i_p}) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B) \\ \rightarrow\ (\lambda\overline{x_n}.\, X(x_{i_1}, \ldots, x_{i_{p-1}}) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, \lambda y.\, B)\end{array}$$

The term B cannot contain the variable x_{i_p} (since none of the variables $x_{i_1}, \ldots, x_{i_{p-1}}$ are equal to x_{i_p}). The result is thus obvious.

Let us suppose now that the application of the rule $(@_@)$ leads to a solution. We show that neither the application of the rule $(@_\pi)$ nor the application of the rule $(@_\beta)$ lead to a solution. We have

$$\begin{array}{l}(\lambda\overline{x_n}.\, X(x_{i_1}, \ldots, x_{i_p}) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B_1 x_{i_p}) \\ \rightarrow\ (\lambda\overline{x_n}.\, X(x_{i_1}, \ldots, x_{i_{p-1}}) \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, B_1) \cup (\lambda\overline{x_n}.\, x_{i_p} \leqslant_{\beta_{sd}} \lambda\overline{x_n}.\, x_{i_p}\end{array}$$

From the second equation, we deduce $x_{i_p} \in \mathrm{fv}(B)$ and thus (see below) the rule $(@_\pi)$ does not lead to a solution. From the first equation we deduce $x_{i_p} \notin \mathrm{fv}(B_1)$. Let us try to apply the rule $(@_\beta)$. We are looking for a term C_1 such that we have $B_1 x_n = C_1[z := x_n]$. But since $x_{i_p} \notin \mathrm{fv}(B_1)$, we must have $C_1 = B_1 z$ with $z \notin \mathrm{fv}(B_1)$ but the term $\lambda z.\, B_1 z$ is not $\beta\eta$-normal and thus the rule cannot be applied. This concludes the proof.

□

6. Algorithms for second-order matching

In this section, we consider second-order matching modulo $\beta\eta$. We show that the algorithm for matching modulo superdevelopments applied in a typed context gives an algorithm for second-order matching. We first recall the second-order matching algorithm given in [HL78]. A more efficient algorithm has been proposed in [CQS96].

(Decomposition)
$$\langle(\lambda\overline{x}.\,\varepsilon(A_1,\ldots,A_n) \leqslant_{\beta\eta} \lambda\overline{x}.\,\varepsilon(B_1,\ldots,B_n)) \cup \mathbb{S}, \sigma\rangle$$
$$\downarrow$$
$$\langle(\lambda\overline{x}.\,A_1 \leqslant_{\beta\eta} \lambda\overline{x}.\,B_1) \cup \ldots \cup (\lambda\overline{x}.\,A_n \leqslant_{\beta\eta} \lambda\overline{x}.\,B_n) \cup \mathbb{S}, \sigma\rangle$$

(Projection)
$$\langle(\lambda\overline{x}.\,X(A_1,\ldots,A_n) \leqslant_{\beta\eta} \lambda\overline{x}.\,B) \cup \mathbb{S}, \sigma\rangle$$
$$\downarrow$$
$$\langle(\lambda\overline{x}.\,(A_i\sigma') \leqslant_{\beta\eta} \lambda\overline{x}.\,B) \cup (\mathbb{S}\sigma'), \sigma'\circ\sigma\rangle$$
where $\sigma' = \{\lambda\overline{y_n}.\,y_i/X\}$
for $1 \leq i \leq n$

(Imitation)
$$\langle(\lambda\overline{x}.\,X(A_1,\ldots,A_n) \leqslant_{\beta\eta} \lambda\overline{x}.\,\varepsilon(B_1,\ldots,B_m) \cup \mathbb{S}, \sigma\rangle$$
$$\downarrow$$
$$\left\langle \begin{array}{l} (\lambda\overline{x}.\,H_1(A_1\sigma',\ldots,A'_n\sigma') \leqslant_{\beta\eta} \lambda\overline{x}.\,B_1) \\ \quad \cup \ldots \cup (\lambda\overline{x}.\,H_m(A_1\sigma',\ldots,A'_n\sigma') \leqslant_{\beta\eta} \lambda\overline{x}.\,B_m), \qquad \sigma'\circ\sigma \\ \quad \cup\, \mathbb{S}\sigma' \end{array} \right\rangle$$
where $\sigma' = \{\lambda\overline{y_n}.\,\varepsilon(H_1(y_1,\ldots,y_n),\ldots H_m(y_1,\ldots,y_n))/X\}$
and $H_1,\ldots,H_n$ **are fresh variables**
of adequate types

Figure 3: Transformation rules for Huet and Lang second-order matching algorithm

6.1. **Second-order Huet and Lang matching algorithm.** The second-order matching algorithm presented in [HL78] is a refinement of the unification algorithm proposed in [Hue75] and presented by transformation rules in [SG89]. We adapt this presentation to the second-order case as it was done in [CQS96]. We must recall that for this algorithm we only consider term in β-long normal form. The application of substitutions is normalising.

The set of transformations rules given in Figure 3 are managing a pair made with a multiset of equations and a substitution. Given a pair $\langle\mathbb{S}, \mathtt{id}\rangle$ the algorithm is said to be successful if there exists a sequence of transformations ending on $\langle\emptyset,\sigma\rangle$. The substitution σ is a match for $\mathbb{S}$ (correctness). The first rule (Decomposition) simplifies equations whose head symbols are identical. The rules (Projection) and (Imitation) are dealing with equations whose left hand side head symbols are matching variables. In the first case, the variable is instantiated by a projection function. We then have to require that the projected argument is equal to the right hand side. In the second case, the variable is partially instantiated in the following sense: the head symbol of the term associated to that variable is set to be equal to the right hand side head symbol; we introduce new matching variables to postpone the remaining choices[2].

The algorithms given in previous section could also have been presented by constructing the substitution at the same time as we transform the system. We did not do it to simplify the presentation (whereas here it is strictly necessary). We now illustrate the algorithm given in Figure 3 on an example.

Example 6.1. We consider the equation $(\lambda x.\, X(x,a) \leqslant_{\beta\eta} \lambda x.\, f(a,x,a))$ where a and f are constants of adequate types. It is clear that the rule (Decomposition) cannot be applied. Applying the rule (Projection) leads to a blocking state since the two arguments of X (that is, x and a) are not matching with the right hand side. Nevertheless, the rule (Imitation) can be applied and gives

$$\left\langle \begin{array}{l} (\lambda x.\, H_1(x,a) \leqslant_{\beta\eta} \lambda x.\, a) \\ \cup\, (\lambda x.\, H_2(x,a) \leqslant_{\beta\eta} \lambda x.\, x) \\ \cup\, (\lambda x.\, H_3(x,a) \leqslant_{\beta\eta} \lambda x.\, a) \end{array} \quad , \; \{\lambda y_1 y_2.\, f(H(y_1,y_2), H_2(y_1,y_2), H_3(y_1,y_2)/X\} \right\rangle$$

The solving of the equations related to H_1, H_2 and H_3 gives the following solutions

$$\begin{array}{lcl} \{\lambda z_1 z_2.\, a/H_1\} & \text{or} & \{\lambda z_1 z_2.\, z_2/H_1\} \\ \{\lambda z_1 z_2.\, z_1/H_2\} & & \\ \{\lambda z_1 z_2.\, a/H_3\} & \text{or} & \{\lambda z_1 z_2.\, z_2/H_3\} \end{array}$$

The initial problem has thus four solutions given by

$$\begin{array}{c} \{\lambda y_1 y_2.\, f(a,y_1,a)/X\} \\ \{\lambda y_1 y_2.\, f(a,y_1,y_2)/X\} \\ \{\lambda y_1 y_2.\, f(y_2,y_1,a)/X\} \\ \{\lambda y_1 y_2.\, f(y_2,y_1,y_2)/X\} \end{array}$$

6.2. **Second-order matching algorithm based on superdevelopments.**

6.2.1. *Matching modulo superdevelopments and types.* We say that an (untyped) term is typable when it belongs to typed terms. A system is typable when for each equation the two terms are typable and of the same type. Otherwise, we say it is untypable.

First, we can remark that if a system is build only with typable terms then its reducts are also build with typable terms. The algorithm given in Figure 1 has a nice behaviour *w.r.t.* types in the following sense: a sequence of transformations ending with a typable system involve only intermediate systems which are typable.

Proposition 6.2. *For all systems $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \twoheadrightarrow \mathbb{S}'$, if the system $\mathbb{S}'$ is typable then the system $\mathbb{S}$ is also typable.*

This proposition is directly obtained from the following lemma

Lemma 6.3. *For every system $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$, if $\mathbb{S}$ is not typable then $\mathbb{S}'$ is not typable .*

[2]The partial instantiation implies that we can now substitute matching variables by term not necessarily $\mathcal{V}$-closed. We are then out of the scope of the framework defined above but this is only to give the standard presentation of Huet and Lang algorithm.

Proof. We suppose given a system $\mathbb{S}$ which is not typable. We show that all reducts of $\mathbb{S}$ are not typable by case analysis on the transformation rule involved.

- *Rules (ε_v) and (ε_c):* obvious.
- *Rule (ε_X):* The systems $\mathbb{S}$ and $\mathbb{S}'$ are defined by $\mathbb{S} = (X \leqslant_{\beta_{sd}} A) \cup \mathbb{S}_0$ and $\mathbb{S}' = (X \leqslant_{\beta_{sd}} A) \cup \mathbb{S}_0\{A/X\}$. If the equation $(X \leqslant_{\beta_{sd}} A)$ is not typable then the result follows. If not (that is if X and A are typable and of the same type), the system $\mathbb{S}_0$ is not typable. But since X and A are of the same type, $\mathbb{S}_0\{A/X\}$ is not typable.
- *Rule (λ_λ):* obvious.
- *Rule $(@_\beta)$:* If $\mathbb{S} = (AB \leqslant_{\beta_{sd}} C) \cup \mathbb{S}_0$ and $\mathbb{S}_0$ is not typable then the result follows. Otherwise, we suppose that the equation $AB \leqslant_{\beta_{sd}} C$ is not typable. Then let σ be the type of AB and τ the one of C with $\tau \neq \sigma$. There exist υ_0 and υ_1 possibly equal and such that the term A is of type $\upsilon_0 \to \sigma$ and the term $\lambda x. C$ is of type $\upsilon_1 \to \tau$. The equation $A \leqslant_{\beta_{sd}} \lambda x. C$ is not typable. This concludes the case.
- *Rule $(@_\pi)$ and $(@_@)$:* Similar to the previous case.

□

Nevertheless, the reducts of a typable system are not necessarily typable, as the following example shows.

Example 6.4 (Reducts of a typable system)**.** Suppose given the equation $XY \leqslant_{\beta_{sd}} fa$. If we suppose that X is of type $s \to \iota$, that Y is of type s, that f is of type $r \to \iota$ and a is of type r where ι, r and s are three different basic types then the equation is a typable system. Nevertheless, by applying the transformation rule $(@_@)$ we get the system $(X \leqslant_{\beta_{sd}} f) \cup (Y \leqslant_{\beta_{sd}} a)$ which is not typable. By the way, this equation has a solution $\{\lambda x. x/X, fa/Y\}$ which is typable.

At this point, one may wonder if there exists a second-order typable β_{sd}-equation with only untypable solutions. If no, one would prove the NP-completeness of matching modulo superdevelopments from the NP-completeness of second-order matching [Bax77]. But the answer is positive as the following example shows.

Example 6.5 (Typable β_{sd}-equation with only untypable solutions)**.** Let a and b two constants of type r. Let f be a constant of type $r \to \iota$ and let g be a constant of type $\iota \to \iota \to \iota$. Let X be a matching variable of type $s \to \iota$. Let Y and Z be two matching variables of type s. Let us consider the second-order typable equation

$$g(XY, XZ) \leqslant_{\beta_{sd}} g(fa, fb)$$

The equation $XY \leqslant_{\beta_{sd}} fa$ has 5 solutions given by

$$\begin{array}{lcl}
X \leqslant_{\beta_{sd}} f & \cup & Y \leqslant_{\beta_{sd}} a \\
X \leqslant_{\beta_{sd}} \lambda x. fa & & \\
X \leqslant_{\beta_{sd}} \lambda x. fx & \cup & Y \leqslant_{\beta_{sd}} a \\
X \leqslant_{\beta_{sd}} \lambda x. xa & \cup & Y \leqslant_{\beta_{sd}} f \\
X \leqslant_{\beta_{sd}} \lambda x. x & \cup & Y \leqslant_{\beta_{sd}} fa
\end{array}$$

In the same way, the equation $XZ \leqslant_{\beta_{sd}} fb$ has 5 solutions given by:

$$\begin{array}{lcl}
X \leqslant_{\beta_{sd}} f & \cup & Z \leqslant_{\beta_{sd}} b \\
X \leqslant_{\beta_{sd}} \lambda x. fb & & \\
X \leqslant_{\beta_{sd}} \lambda x. fx & \cup & Z \leqslant_{\beta_{sd}} b \\
X \leqslant_{\beta_{sd}} \lambda x. xa & \cup & Z \leqslant_{\beta_{sd}} f \\
X \leqslant_{\beta_{sd}} \lambda x. x & \cup & Z \leqslant_{\beta_{sd}} fb
\end{array}$$

In each case, only one solution is typable: $X \leqslant_{\beta_{sd}} \lambda x.\, fa$ and $X \leqslant_{\beta_{sd}} \lambda x.\, fb$ respectively. But these solutions are of course incompatible. The only solution to the initial equation is

$$(X \leqslant_{\beta_{sd}} f) \cup (Y \leqslant_{\beta_{sd}} a) \cup (Z \leqslant_{\beta_{sd}} b)$$

which is untypable. We have exhibited a second-order β-equation with no solution while the corresponding β_{sd}-equation has one.

6.2.2. *Second-order matching algorithm based on superdevelopments.* The context of this section is the typed λ-calculus. We want to determine β-matches of second-order equations. Note that the η-rule does not have a fundamental impact on the algorithm. This section could have be written for second-order matching modulo $\beta\eta$ (including the paragraph related to the comparison with Huet and Lang algorithm).

First, we can remark that we can restrict ourselves to typable equations (this is justified by Proposition 6.2). Then the rule given in Figure 1 are applied only if the system obtained after reduction is typable. This is a common restriction when dealing with higher-order matching in a typed context (see for example the rule (Projection)).

Moreover, we have shown (Proposition 3.10) that every β-match of a second-order equation is also a β_{sd}-match for the corresponding equation. We thus deduce that the algorithm given in Figure 1 applied in a typed context (only typable systems) gives an algorithm for second-order matching

Theorem 6.6 (Second-order matching algorithm)**.** *The rules of Figure 1 applied in a typed context give a sound and complete algorithm for second-order matching.*

We illustrate the algorithm on an example.

Example 6.7. We consider the equation $\lambda x.\, X(x, a) \leqslant_{\beta\eta} \lambda x.\, f(a, x, a)$ given in example 6.1. We first apply the rule (λ_λ) to remove all head λ-abstractions. We now look at the different opportunities to apply the $(@_\beta)$ rule, namely how we can instantiate the terms B_1 and B_2 (following the notations used in Fig. 1). Since we must have $a \leqslant_{\beta_{sd}} B_2$, necessarily $B_2 = a$. Then there are three choices for B_1

(1) $B_1 = \lambda y_2.\, f(y_2, x, y_2)$;
(2) $B_1 = \lambda y_2.\, f(y_2, x, a)$;
(3) $B_1 = \lambda y_2.\, f(a, x, y_2)$.

Each of these choices lead to the solutions

$$\{\lambda y_1 y_2.\, f(y_2, y_1, y_2)/X\}$$
$$\{\lambda y_1 y_2.\, f(y_2, y_1, a)/X\}$$
$$\{\lambda y_1 y_2.\, f(a, y_1, y_2)/X\}$$

Finally, by applying the rule $(@_\pi)$ we found the fourth solution given by

$$\{\lambda y_1 y_2.\, f(a, y_1, a)/X\}.$$

Remark 6.8 (Comparison of second-order matching algorithms)**.** The Huet and Lang second-order matching algorithms and the one based on superdevelopments are quite different. If we think of terms as trees, Huet and Lang algorithm compares term in a backwards manner (head symbol) while the algorithm based on superdevelopments compares terms in an upwards manner.

7. Algorithm for matching modulo developments (and eta)

In the previous sections, we have presented some works on matching modulo superdevelopments. One may wonder if these works can be adapted for the more restrictive notion of matching modulo developments. We show in this section that it is indeed the case.

This section is organised as follows. We first give a precise definition of matching modulo developments. We show that it is neither complete for second-order matching nor for Miller pattern matching. We finally conclude by giving explicitly a sound, complete and terminating algorithm for matching modulo developments.

7.1. **Matching modulo developments.**

Definition 7.1 (β_d-matching equation/system)**.** A β_d-*matching equation* or simply a *matching equation* is a pair of terms denoted $A \leqslant_{\beta_d} B$ such that B is normal and $\mathcal{V}$-closed. A matching *system* is a multiset (potentially empty) of matching equations.

Definition 7.2 (β_d-match)**.** A substitution φ on matching variables is a β_d-*match* for the equation $A \leqslant_{\beta_d} B$ if there exists a development from $A\varphi$ and B (that is, $A\varphi \Longrightarrow_\beta B$). A substitution is a match of a system if it matches each equation.

Example 7.3 (Incompleteness for second-order matching)**.** We consider the matching equation $(XY)Z \leqslant_{\beta_d} ab$, following Example 3.3. We also consider the substitutions

$$\begin{array}{lcl} \sigma_1 & = & \{ \quad \lambda x.\, x \ /X, \ a \ /Y, \ b \ /Z\} \\ \sigma_2 & = & \{ \lambda x.\, \lambda y.\, xy \ /X, \ a \ /Y, \ b \ /Z\} \end{array}$$

Only the substitution σ_1 is a β_d-match while the substitution σ_2 is a second-order β-match.

Example 7.4 (Incompletenes for Miller pattern matching)**.** Consider for example the equation of Miller patterns $(\lambda x.\, \lambda y.\, ((Xx)y),\ \lambda x.\, \lambda y.\, xy)$. The substitution $\{\lambda z_1.\, \lambda z_2.\, z_1 z_2/X\}$ is not a β_d-match of the corresponding equation while it is a β-match.

7.2. **Algorithm for matching modulo developments.**

The algorithm for matching modulo developments mimics the definition of parallel reduction given in Section 1.2.2, in the same way the algorithm for matching modulo superdevelopments mimics the definition of strong parallel reduction.

We recall that the parallel reduction and the strong parallel reduction differ only on one rule: the ($Red{-}\beta$) of the parallel reduction is replaced by the ($Red{-}\beta_s$) of strong parallel reduction. Then the corresponding algorithms only differ on the rules in relationship with the ($Red{-}\beta$) and ($Red{-}\beta_s$) rules: the transformation rules $@_\pi$ and $@_\beta$ for matching modulo developments are given in Figure 4.

Theorem 7.5 (Algorithm for matching modulo developments)**.** *The algorithm for matching modulo developments consisting of the rules (ε_v, ε_c, ε_X, λ_λ, $@_@$) given in Figure 1 plus the rules ($@_\pi$) and ($@_\beta$) given in Figure 4 is sound, complete and terminating. It has the finite complete match set property (there is no most general match).*

Proof. To prove these results, it is very easy to adapt the proofs given in Section 4. □

$$
\begin{array}{lll}
((\lambda x.\, A_1)B_1 \leqslant_{\beta_{sd}} C) \cup \mathbb{S} & \rightarrow_{@_\pi} & (\lambda x.\, A_1 \leqslant_{\beta_{sd}} \lambda x.\, C) \cup \mathbb{S} \\
 & & \textbf{where } x \textbf{ fresh variable} \\
((\lambda x.\, A_1)B_1 \leqslant_{\beta_{sd}} C) \cup \mathbb{S} & \rightarrow_{@_\beta} & (\lambda x.\, A_1 \leqslant_{\beta_{sd}} \lambda x.\, A_2) \cup (B_1 \leqslant_{\beta_{sd}} B_2) \cup \mathbb{S} \\
 & & \textbf{where } A_2[x := B_2] = C \\
 & & \textbf{and } x \textbf{ fresh variable } x \in \mathrm{fv}(A_2) \\
 & & \textbf{and } A_2,\, B_2\ \beta\textbf{-normal}
\end{array}
$$

Figure 4: The $@_\pi$ and $@_\beta$ transformation rules for matching modulo developments

The case of matching modulo developments and η, not presented here, is also easily deduced from the previous sections.

8. Applications to higher-order rewriting

Higher-order rewriting can be defined [MN98] modulo the $\beta\eta$-equivalence of the simply typed λ-calculus. Since this calculus is used to instantiate terms, we call it a *substitution calculus* according to the terminology of presented in [Oos94, vR96].

Other higher-order rewriting frameworks choose other substitution calculi. For example, Combinatory Reduction Systems (CRS) [Klo80, KvOvR93] use the λ-calculus modulo developments as a substitution calculus. To our knowledge, the work presented in this paper is the first one giving an algorithm for the CRS matching, that is for higher-order matching modulo developments.

Polyadic developments and superdevelopments. Higher-order rewriting frameworks that use the λ-calculus modulo developments as a substitution calculus are enough expressive. We give examples where matching modulo developments is too rough. This was already noticed [vOvR93] when comparing CRS and HRS. Indeed, to encode HRS in CRS we use instead of developments the more general notion of polyadic developments. Polyadic developments are developments with steps of the form

$$(\ldots(\lambda x_1 \ldots \lambda x_n.\, A)B_1)\ldots)B_n \rightarrow A[x_1 := B_1]\ldots[x_n := B_n]$$

It is a generalization of developments that also reduce redexes created in the in the first way (according to the taxonomy given page 6). The only difference with superdevelopments is then that polyadic developments do not reduce redex occurrences that are created in the second way.

In the context of higher-order rewriting, we do not see the added value of reducing the redexes created in the second way when instantiating a term. We believe that polyadic developments and superdevelopments are essentially the same. We consider redex creations of type 2 as syntactical accidents.

Although λ-calculus modulo polyadic developments is a natural choice to be a substitution calculus for higher-order rewriting (as suggested in [Ter03]), we are not aware of any algorithm dealing with higher-order matching modulo polyadic developments. In particular, polyadic developments do not have a big step semantics that can guide the design of a corresponding algorithm in the spirit of this paper.

Untyped λ-calculus modulo superdevelopments as a substitution calculus. It thus appears that λ-calculus modulo superdevelopments is a very good compromise to be a substitution calculus for higher-order rewriting: the matching is complete for second-order and Miller pattern-matching and the corresponding matching algorithm does not depend one a particular type system and makes an optional use of η-equivalence.

Conclusion

We propose a new approach to study higher-order matching: instead of working in the typed λ-calculus modulo the full β-reduction we propose to work in the untyped λ-calculus modulo a restriction of the β-equivalence, namely superdevelopments. This is in the spirit of J.J. Levy works: we prefer to use the (hidden) typing on the reduction than the typing on terms. Higher-order matching modulo superdevelopments is of particular interest since all second-order β-matches are matches modulo.

We present and study an algorithm for matching modulo superdevelopments. We prove it to be sound, complete and terminating. We also show that this algorithm can be adapted to deal with second-order matching as well as with matching modulo developments. We show that since we consider untyped frameworks, the use of the η-equivalence does not influence the design and behavior of our algorithms. We describe the algorithms in a mathematically elegant way that allow us to write intuitive proofs.

We finally apply these ideas in the context of higher-order rewriting. We give an algorithm for matching modulo developments, which is at the heart of Combinatory Reduction Systems. We also notice that higher-order rewriting with the untyped λ-calculus modulo superdevelopments as a meta-language is of particular interest.

Even if unification modulo superdevelopments is undecidable (by reduction to the undecidability of second-order unification [Gol81]), it might be relevant to study it. In particular, one may wonder if the work presented in this paper suggests another approach for unification of Miller patterns.

As far as it concerns the transformations of pattern-matching programs, the work of [dMS01] motivates by several examples higher-order matching in extensions of λ-calculus with patterns such as the pure pattern calculus [JK09] or the rewriting calculus [CK01]. Since a simple type system that ensures termination is difficult to find in this context, this paper should give useful guidelines.

Acknowledgements

We would like to thank E. Bonelli for some comments that motivated this work. It benefited of the discussions we had with H. Cirstea, C. Kirchner and G. Nadathur.

References

[Acz78] Peter Aczel. A general Church-Rosser theorem. Technical report, University of Manchester, July 1978. 8

[Bar84] Henk Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. North Holland, 1984. Second edition. 2, 3, 5

[Bax77] Lewis Denver Baxter. *The complexity of unification*. Thèse de doctorat, University of Waterloo, 1977. 29

[Bür90] Hans-Jürgen J. Bürckert. Matching — A special case of unification? In *Unification*, pages 125–138. Academic Press, 1990. 17

[CK01] Horatiu Cirstea and Claude Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001. 33

[CQS96] Régis Curien, Zhenyu Qian, and Hui Shi. Efficient second-order matching. In *Rewriting Techniques and Applications - RTA'96*, volume 1103 of *Lecture Notes in Computer Science*, 1996. 26, 27

[Des98] Joëlle Despeyroux. Natural semantics: specifications and proofs. Technical report, INRIA, Février 1998. 5

[DHK00] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher order unification via explicit substitutions. *Information and Computation*, 157(1/2):183–235, 2000. 16

[dMS01] Ooge de Moor and Ganesh Sittampalam. Higher-order matching for program transformation. *Theoretical Computer Science*, 269:218–237, 2001. 2, 3, 13, 33

[Dow94] Gilles Dowek. Third order matching is decidable. *Annals of Pure and Applied Logic*, 69:135–155, 1994. 10

[Dow01] Gilles Dowek. Higher-order unification and matching. In *Handbook of Automated Reasoning*. Elsevier, 2001. 3, 11, 14

[DSP91] Mary Dalrymple, Stuart M. Shieber, and Fernando Pereira. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14:399–452, 1991. 1

[Fau06] Germain Faure. Matching modulo superdevelopments application to second-order matching. In *13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning - LPAR'06*, volume 4246 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 2006. 3

[Fau07] Germain Faure. *Structure et modèles de calcul de réécriture*. Thèse de doctorat, Université Henri Poincaré Nancy, 2007. 3

[Gol81] Donald Goldfarb. The undecidability of the second order unification. *Journal of Theoretical Computer Science*, 13:225–230, 1981. 11, 33

[HL78] Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11, 1978. 1, 2, 10, 17, 26, 27

[Hue73] Gérard Huet. A mechanization of type theory. In Nils J. Nilsson, editor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 139–146, Standford, CA, 1973. 11

[Hue75] Gérard Huet. *Résolution d'équations dans les langages d'ordre 1, 2, ..., ω*. Thèse de doctorat d'état, Université Paris, 7, 1975. 2, 11, 27

[JK91] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In *Computational Logic - Essays in Honor of Alan Robinson*, pages 257–321, 1991. 15

[JK09] C. Barry Jay and Delia Kesner. First-class patterns. *Journal Functional Programming*, 19(2):191–225, 2009. 33

[Kir84] Claude Kirchner. A new equational unification method: A generalization of martelli-montanari's algorithm. pages 224–247, 1984. 15

[Klo80] Jan Willem Klop. *Combinatory Reduction Systems*. Ph.D. thesis, Mathematisch Centrum, Amsterdam, 1980. 2, 4, 32

[KvOvR93] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993. 2, 4, 32

[Lév78] Jean-Jacques Lévy. *Reductions Correctes et Optimales dans le Lambda-Calcul*. PhD thesis, Université de Paris, 1978. 6

[Loa03] Ralph Loader. Higher order beta matching is undecidable. *Logic Journal of the IGPL*, 11(1):51–68, 2003. 2, 10, 14

[Mil90] Dale Miller. Higher-order logic programming. In *Int. Conf. on Logic Programming*, page 784, 1990. 1

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1991. 2, 14

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982. 15

[MN98] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192, 1998. 1, 2, 32

[NP98] Tobias Nipkow and Christian Prehofer. Higher-order rewriting and equational reasoning. In *Automated Deduction: A Basis for Applications*. Kluwer, 1998. 1

[Oos94] Vincent Van Oostrom. *Confluence for abstract and higher-order rewriting*. PhD thesis, Vrije Universiteit, 1994. 2, 32

[Pad00] Vincent Padovani. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, 3(10):361–372, June 2000. 10

[Pfe01] Frank Pfenning. Logical frameworks. In *Handbook of Automated Reasoning*, volume II, chapter 17, pages 1063–1147. Elsevier Science, 2001. 1

[Qia96] Zhenyu Qian. Unification of higher-order patterns in linear time and space. *J. Log. Comput*, 1996. 2, 11, 14

[SG89] Wayne Snyder and Jean Gallier. Higher-order unification revisited: Complete sets of transformations. *JSCOMP: Journal of Symbolic Computation*, 8, 1989. 11, 15, 27

[Shi94] Hui Shi. *Extended matching with applications to program transformation*. PhD thesis, Universität Bremen, 1994. 1

[Sit01] Ganesh Sittampalam. *Higher-order Matching for Program Transformation*. PhD thesis, Magdalen College, 2001. 2, 3

[Sti09] Colin Stirling. Decidability of higher-order matching. *Logical Methods in Computer Science*, 5(3), 2009. 2, 10, 14

[Ter03] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. M. Bezem, J. W. Klop and R. de Vrijer, eds. 2, 5, 32

[Vis05] Eelco Visser. A survey of strategies in rule-based program transformation systems. *Journal of Symbolic Computation*, 40(1), 2005. 1

[vOvR93] Vincent van Oostrom and Femke van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. volume 816 of *Lecture Notes in Computer Science*, 1993. 1, 32

[vR93] Femke van Raamsdonk. Confluence and superdevelopments. In Claude Kirchner, editor, *5th International Conference on Rewriting Techniques and Applications - RTA'93*, volume 690 of *Lecture Notes in Computer Science*, pages 168–182. Springer-Verlag, 1993. 2, 3, 6

[vR96] Femke van Raamsdonk. *Confluence and Normalisation for higher-order rewriting*. Thèse de doctorat, Vrije Universiteit, 1996. 2, 3, 7, 32