Matching modulo superdeveloppements. Application to second-order matching.

Germain Faure

Matching modulo superdevelopments Application to second-order matching

Germain Faure

Université Henri Poincaré & LORIA
BP 239, F-54506 Vandoeuvre-lès-Nancy France
faure@loria.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 based on finite superdevelopments. We consider higher-order matching modulo this equivalence over untyped λ-terms for which we propose a terminating, sound and complete matching algorithm.
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.

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 is usually defined as the following problem: given a set of equations $s_i = t_i$ between *typed* λ-terms where the 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.

In [dMS01] O. De Moor and G. Sittampalam introduced a new approach to higher-order matching for automatic program transformation in an untyped setting. Matching equations are solved modulo a one-step reduction for an appropriate parallel β-reduction notion that does not coincide with the standard one of Tait and Martin-Löf. As the authors suggest in the paper, this operation, that always terminates even in an untyped context, represents in a certain sense an approximation of the β-normalization process.

The standard approximation of β-normal forms is given by complete developments [Bar84]. But the parallel β-normal forms used in [dMS01] provide a more precise approximation than the complete developments. Actually, the latter parallel β-reduction was introduced first by P. Aczel in [Acz78] and the corresponding approximation was introduced by F. van Raamsdonk [vR93] under the

name of superdevelopments. They are an extended notion of finite developments that were introduced to prove the confluence of a general class of reduction systems containing λ-calculus and term rewrite systems. A superdevelopment is a reduction sequence that may reduce the redexes of the term, its residuals and some created redexes but not those created by the substitution of a variable in functional position by a λ-abstraction. The approximation given by superdevelopments coincides with β-normal forms when considering second-order terms.

In this paper, we 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.

The one-step reduction modulo which one considers matching equations in [dMS01] "may be a little difficult to understand" as the authors of the original paper said. In this paper, we shed light on this reduction by giving a clear relationship with superdevelopments (and quoting the original definition of P. Aczel). The original algorithm is presented using transformation rules as suggested in [GS89]. This method provides an abstract and elegant way to give a clear separation between the operational and logical issues.

The main goal of this paper is convince that superdevelopments constitute a right tool for tackling the matching problems. We also claim that the background theory of superdevelopments provides nice intuitions and simplifications of the different proofs (w.r.t. [dMS01,Sit01]) especially *w.r.t.* the application to second-order matching.

In fact, the general approach of solving equations modulo a restricted notion of reduction can be useful to deal with higher-order matching in calculi for which a simple type system that ensures termination is difficult to find. This is for example the case of simply typed ρ-calculus [CLW03] which is not terminating. Higher-order matching in the ρ-calculus or in pure pattern calculus [JK06] are useful in the transformation of pattern-matching programs or in proof theory that handles rich proof-terms in the generalized deduction modulo [Wac06].

Road-map The paper is organized as follows. The first section introduces the syntax and the superdevelopments. The second section is devoted to the presentation of matching modulo superdevelopments and its link with usual higher-order matching (second-order matching, third-order matching and matching of patterns à la Miller). Section 3 presents and studies[1] an algorithm to perform matching modulo superdevelopements in the λ-calculus. Section 4 explicitly analyzes the optional role of the η rule in the matching process. Throughout the paper, many examples are taken.

1 Preliminaries

In this section, we first recall some basic definitions and set some notations related to the λ-calculus. Then we define in two different ways superdevelopments

[1] The proofs not given in the paper will be available in a journal paper.

as a restriction of the β-reduction. We refer the reader to [Bar84] or [Dow01] for the fundamental definitions and results on the λ-calculus.

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 $\alpha, \beta \in \mathfrak{T}$ 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 (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 and disjoint sets 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 $\mathcal{V}$ sometimes called unknowns. We justify the use of the two sets in Section 3.

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. All symbols can be indexed. Positions in λ-terms are denoted by $p_1, ..., p_n$. We denote by $\preceq$ the canonical order on positions. The subterm of A at position p_1 is denoted by $A_{|p_1}$.

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 closed if it contains no matching variables and no free variables. We denote by $\mathcal{FV}(A)$ the set of 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 $B\{A/x\}$.

As in any calculus involving binders, we work modulo the α-*conversion* of Church, and modulo the *hygiene-convention* of Barendregt [Bar84], *i.e.*, free and bound variables have different names.

We denote a β-reduction step by $\to_\beta$, 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 $\to_\beta$.

1.2 Untyped labelled λ-calculus and β_l-reduction

When no ambiguity is possible, we use the same notation for both typed and untyped (labelled) terms. Labels are simply elements of $\mathbb{N}$.

Definition 1 (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 $(MN)^p \in \mathcal{T}_l$.*

We define β_l-reduction on the set of labelled λ-terms as follows:

$$((\lambda_p x.A)B)^p \to_{\beta_l} A\{B/x\}$$

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.

Definition 2 (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 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 Untyped λ-calculus and superdevelopments

The untyped λ-calculus is defined as the labelled λ-calculus by simply erasing all labels (both in terms and reduction). The set of untyped λ-terms is denoted by $\mathcal{T}$. We introduce a generalization of (finite) developments [Bar84] called superdevelopments. This notion, initially introduced in [vR93], is related to the three ways to create redexes in the λ-calculus [Lév78]:

1.	$((\lambda x\,.\,\lambda y\,.\,A)\;B)\;C$	$\to_\beta$	$(\lambda y\,.\,A\{B/x\})\;C$
2.	$((\lambda x\,.\,x)\;(\lambda y\,.\,A))\;B$	$\to_\beta$	$(\lambda y\,.\,A)\;B$
3.	$(\lambda x\,.\,A)(\lambda y\,.\,B)$	$\to_\beta$	$A\{\lambda y\,.\,B/x\}$ if there is a position p_1 such that $A_{\|p_1} = xA_0$

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". By restricting to well-labelled terms we exactly restrict to upwards creations.

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 3 (Superdevelopments). *A β-rewrite sequence ς of the λ-calculus is a β-superdevelopment if it 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 $\Upsilon(\sigma) = \varsigma$, where Υ is the canonical mapping from labelled λ-terms to λ-terms and from β_l-reduction to β-reduction that simply erases labels.*

For example, the β-rewrite sequence $(\lambda x\,.\,\lambda y\,.\,xy)zz' \to_\beta (\lambda y\,.\,zy)z' \to_\beta zz'$ is a superdevelopment since it corresponds to the β_l-rewrite sequence $(((\lambda_1 x.\lambda_2 y.xy)z)^1)z')^2 \to_{\beta_l} ((\lambda_2 y.zy)z')^2 \to_{\beta_l} zz'$.

However, the rewrite sequence $(\lambda x\,.\,xx)(\lambda x\,.\,xx) \to_\beta (\lambda x\,.\,xx)(\lambda x\,.\,xx) \to_\beta \ldots$ is not a superdevelopment.

Given a λ-term, we can "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 (this a good exercise left to the reader). We now give four examples of β-reductions that are superdeveloppements.

[**Finite development**] Residuals of redexes present in the initial term can be contracted:

$$\begin{aligned}&(\lambda x\,.\,f(x,x))\ ((\lambda y\,.\,y)\ a)\\ &\to_\beta f((\lambda y\,.\,y)\ a, (\lambda y\,.\,y)\ a)\\ &\to_\beta f(a, (\lambda y\,.\,y)\ a)\\ &\to_\beta f(a,a)\end{aligned}$$

[**Redex creation of type 1**] In the following superdevelopment, the new redex obtained after one β-rewrite step is reduced:

$$\begin{aligned}&((\lambda x\,.\,\lambda y\,.\,f(x,y))a)b\\ &\to_\beta (\lambda y\,.\,f(a,y))b\\ &\to_\beta f(a,b)\end{aligned}$$

[**Redex creation of type 2**] As in the previous example, a redex is created and reduced during reduction, but in a different way:

$$\begin{aligned}&((\lambda x\,.\,x)(\lambda y\,.\,y))a\\ &\to_\beta (\lambda y\,.\,y)a\\ &\to_\beta a\end{aligned}$$

[**Redex creation of type 3**] There is no superdevelopment from the term $(\lambda x\,.\,xa)(\lambda y\,.\,y)$ to the term a:

$$\begin{aligned}&(\lambda x\,.\,xa)(\lambda y\,.\,y)\\ &\to_\beta (\lambda y\,.\,y)a\end{aligned}$$

1.4 Another characterization of superdevelopments

As finite developments coincide with the classical parallel reduction of Tait and Martin-Löf, finite superdevelopments coincide with Aczel's parallel reduction [Acz78] called in the following strong parallel β-reduction. It is denoted by $\Longrightarrow_{\beta_{sd}}$ and we say that a term A β_{sd}-reduces to a term B if $A \Longrightarrow_{\beta_{sd}} B$. It is defined inductively in Figure 1.

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 given by

$$\frac{\lambda x\,.\,A_1 \Longrightarrow_\beta \lambda x\,.\,A_2 \quad B_1 \Longrightarrow_\beta B_2}{(\lambda x\,.\,A_1)B_1 \Longrightarrow_\beta A_2\{B_2/x\}}\ (Red - \beta)$$

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

$$\frac{A_1 \Longrightarrow_{\beta_{sd}} A_2 \quad B_1 \Longrightarrow_{\beta_{sd}} B_2}{A_1 B_1 \Longrightarrow_{\beta_{sd}} A_2 B_2}\ (Red-@)$$

$$\frac{A_1 \Longrightarrow_{\beta_{sd}} \lambda x\,.\,A_2 \quad B_1 \Longrightarrow_{\beta_{sd}} B_2}{A_1 B_1 \Longrightarrow_{\beta_{sd}} A_2\{B_2/x\}}\ (Red-\beta_s)$$

Fig. 1. Strong parallel β-reduction

The following result states that superdevelopments coincide with strong parallel β-reduction. This characterization is the essence of the matching algorithm.

Theorem 2 *There exists a superdevelopment from A to B iff $A \Longrightarrow_{\beta_{sd}} B$.*

2 Matching modulo superdevelopments

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

2.1 Definition of β_{sd}-matching

Definition 3 (Substitution) *A* matching substitution *or simply a* substitution $\varphi : \mathcal{V} \to \mathcal{T}$ *is a function from matching variables to terms. If $\varphi = \{A_1/X_1, \ldots, A_n/X_n\}$ then the domain of φ is the set $\{X_i\}_{i=1}^n$. We overload the notation used for substitutions of variables and we denote by $B\{A/X\}$ the substitution of A for the matching variable X in B. In this work, we only consider closed and normal substitutions that are substitutions of closed normal terms.*

Since we consider classes of terms modulo α-conversion, when applying a substitution the appropriate representatives are always chosen in order to avoid potential variable captures.

Definition 4 (Union) *Two substitutions coincide if their images* coincide *on the intersection of their domains. We then straightforwardly define the* union *of two substitutions σ and φ that coincide and denote it by $\sigma \cup \varphi$.*

Definition 5 (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 does not contain matching variables. A matching* system *is a multiset of matching equations.*

For example, $XY \leqslant_{\beta_{sd}} \lambda x\,.\,x$ and $(\lambda x\,.\,x)X \leqslant_{\beta_{sd}} a$ are β_{sd}-matching equations whereas $XY \leqslant_{\beta_{sd}} (\lambda x\,.\,x)a$ is not.

Every solution of a matching equation is supposed to be a *closed* substitution. 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 $\mathbb{S}$.

Definition 6 (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 and only if $A\varphi \Longrightarrow_{\beta_{sd}} B$. 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 $\mathbb{M}(\mathbb{S})$.*

For example, $\{\lambda xy\,.\,y/X\}$ and $\{\lambda y\,.\,y/X, \lambda x\,.\,x/Y\}$ are β_{sd}-matches for the equation $XY \leqslant_{\beta_{sd}} \lambda x\,.\,x$. The substitution $\{\lambda z\,.\,z(\lambda x\,.\,x)/X, \lambda y\,.\,y/Y\}$ is not a β_{sd}-match because $(\lambda z\,.\,z(\lambda x\,.\,x))(\lambda y\,.\,y)$ does not β_{sd}-reduce to $\lambda x\,.\,x$. (although it β-reduces).

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 $\mathbb{S}$.

Definition 7 (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 $[A\backslash X]$. 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 susbtitution 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 8 (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 9 *If $\mathbb{S}$ is a system in solved form then $\{\sigma_{\mathbb{S}}\}$ is a complete match set of $\mathbb{S}$.*

2.2 Comparison with usual higher-order matching

Comparison with second-order matching First, we relate β_{sd}-matching with second-order matching (*i.e.*, typed higher-order β-matching where all matching variables are second-order and where constants are third-order). We show that all solutions of a given second-order system are β_{sd}-matches. In this section, all terms are supposed to be typable, *i.e.*, belong to $\mathcal{T}_t$.

We simply recall the definition of β-matching and we refer to [GS89] for a more complete and self-contained presentation.

Definition 10 (β-matching equation and β-match) *A* β-matching equation *is a pair of β-normal typed λ-terms of the same type denoted $A \leqslant_\beta B$ such that B does not contain matching variables. A substitution φ, that preserves types, is a β-match for the matching equation $A \leqslant_\beta B$ if and only if $A\varphi =_\beta B$. We generalize the definition to matching systems as in Def. 6.*

If we erase the types, then all β-matching equations are β_{sd}-matching equations. We will switch from the former to the latter without explicit mentions.

The following results were already proved in [dMS01]. Nevertheless, the formalization using superdevelopments (and not only strong β-parallel reduction) introduced in this paper gives quite simple and clear proofs.

First, a technical result on the creation of redexes.

Lemma 11 *For all terms $A_1, \ldots, A_n$ such that there exists a superdevelopment $A_1 \to_\beta \ldots \to_\beta A_n$ and A_n contains a redex of third order order (or more), 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 $C = C'\{E/z\}$. Then the order of the redex $(\lambda z \,.\, \lambda x \,.\, C')\ E$ is greater or equal to the of order 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. □

Proposition 12 *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 $A\varphi =_\beta B$, that A does not contain any β-redex, that φ does not contain any term of order greater than 2. Finally, $A\varphi \not\Rightarrow_{\beta_{sd}} B$ and $A\varphi =_\beta B$. Thus there exist $(A_i)_i$ such that $A\varphi \to_\beta A_1 \to_\beta \cdots \to_{\beta_{sd}} A_n$ 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 11 implies that $A\varphi$ contains a redex of order at least three. 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. □

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 this matching process, we can restrict β-normalization to β_{sd}-normalization.

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 13 *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 $\lambda x \,.\, \lambda f \,.\, (f \ldots (f\ x) \ldots)$.

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

Proposition 14 *Let* φ *be a match of an equation* $P \leqslant_\beta A$ *where* P *is a pattern à la Miller. Then there exists a superdevelopment* $P\varphi \twoheadrightarrow\!\!\!\!\!\!\mid_\beta A$.

3 An algorithm for matching modulo superdevelopments

In this section, we first present an algorithm for matching modulo superdevelopments. We illustrate it on several examples and finally state its main properties.

3.1 Presentation of the algorithm

We propose in Figure 2 an algorithmic description of matching modulo superdevelopments using transformation rules [GS89]:
— A system is transformed by successively applying the rules until we get to a normal form (it always exists since the rules terminate) that gives a solution (the algorithm is sound).
— By exploring all possible reductions (the rule are non-deterministic in the sense that at each step there are possibly several rules that can applied) and collecting all solved forms we get a complete match set (since the algorithm is sound and complete).

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

Fig. 2. Matching algorithm for higher-order matching modulo superdevelopments

We write $\mathbb{S} \rightarrow \mathbb{S}'$ if there exists a transformation rule that can be applied to transform $\mathbb{S}$ into $\mathbb{S}'$ and $\mathbb{S} \twoheadrightarrow \mathbb{S}'$ if there exist $n \geq 0$ systems $\mathbb{S}_1, \ldots, \mathbb{S}_n$ such that $\mathbb{S} \rightarrow \mathbb{S}_1 \rightarrow \ldots \mathbb{S}_n \rightarrow \mathbb{S}'$. The matching algorithm follows the definition of strong β-parallel reduction:

The ε rules: deal with atoms. The rules (ε_c) and (ε_v) are trivial rules dealing with variables and constants. The rule (ε_X) substitutes a matching variable by its corresponding value. Remark first that we do not substitute by terms containing free variables and then that we do not normalize when applying a substitution (otherwise the rule (ε_X) would not be sound ; see the long version for further details). In the rule (ε_X), we compel that the substituted term contains no free variables. This is not a stricly needed condition but if the condition is not verified there is no interest to apply the rule since the system will never lead to a solved form (precisely because of the condition is not verified).

The λ rule: deals with abstraction by mimicking the $(Red - \lambda)$ rule (thanks to the implicit α-renaming, we can suppose that the two bound variables are the same). This rule illustrates the use of two different sets of "variables": we can "unbind"a variable safely without possible confusion with a matching variable (recall that we only consider *closed* substitutions). Many algorithms use a single set of variables. In this case, since matching variables are the free variables of the left-hand side of the equation, we have to remember the variables that were bound in the initial equation. The two choices are relevant.
A similar rule is used in the works on unification in the λ-calculus with explicit substitutions and de Bruijn indices [DHK00].

The @ rules: deal with application. The $(@_@)$ rule is in one-to-one correspondance with the rule $(Red - @)$ and thus does not need further comments. The rules $(@_\pi)$ and $(@_\beta)$ are both related to the rule $(Red - \beta_s)$. We try to express the right-hand side C of the equation as the result of a β-reduction let us say $A_2\{B_2/x\}$. Depending on the presence of x in A_2, we obtain the rule $(@_\pi)$ or $(@_\beta)$. If x does not belong to A_2, then we obtain the rule $(@_\pi)$: the left-hand side of the application is mapped to an abstraction that ignores its argument and returns the right-hand side of the matching equation. Otherwise (if x belongs to A_2), we obtain the rule $(@_\beta)$ by mimicking the $(Red - \beta_s)$ for all terms such that $A_2\{B_2/x\} = C$ where x belongs to A_2 and A_2, B_2 are normal. To find the terms A_2 and B_2 we first remark that B_2 must be a subterm of C (since x belongs to A_2). We can thus choose one of them. Then, choose a subset of the set of positions on which B_2 appears in C. Then A_2 is obtained from C by putting x at every position of the chosen set. Notice that there are finitely many pairs (A_2, B_2) satisfying the conditions.

Notice that the matching algorithm does not introduce new matching variables (this is not the case in [HL78]). This is for example pertinent in [Ali05].

Example 15 *We consider the matching equation $XY \leqslant_{\beta_{sd}} ab$. Since the left and right-hand sides of the matching equation are applications, we can apply the rules $(@_\pi)$, $(@_@)$ or $(@_\beta)$.*

1. *Rule $(@_\pi)$:* $XY \leqslant_{\beta_{sd}} ab \rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,ab$.
2. *Rule $(@_@)$:* $XY \leqslant_{\beta_{sd}} ab \rightarrow X \leqslant_{\beta_{sd}} a\ ,\ Y \leqslant_{\beta_{sd}} b$.
3. *Rule $(@_\beta)$: to find A_1 and A_2 such that $A_1\{A_2/x\} = ab$ first we choose A_2 as one of the subterm of "ab": a, b and ab. There is only one subset of the set of positions on which A_2 appears in ab (since each subterm of ab appears once). Then we can apply the rule $(@_\beta)$ in different ways corresponding to the three subterms of the right-hand side of the equation:*
   (a) $XY \leqslant_{\beta_{sd}} ab \rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,xb\ ,\ Y \leqslant_{\beta_{sd}} a$.
   (b) $XY \leqslant_{\beta_{sd}} ab \rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,ax\ ,\ Y \leqslant_{\beta_{sd}} b$.
   (c) $XY \leqslant_{\beta_{sd}} ab \rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,x\ ,\ Y \leqslant_{\beta_{sd}} ab$.

Example 16 *We consider the equation $X(YX) \leqslant_{\beta_{sd}} a$. We can apply either the rule $(@_\pi)$ or $(@_\beta)$.*

1. *Rule $(@_\pi)$:* $X(YX) \leqslant_{\beta_{sd}} a \rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,a$.
2. *Rule $(@_\beta)$:* $X(YX) \leqslant_{\beta_{sd}} a \rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,x,\ YX \leqslant_{\beta_{sd}} a$.
   To simplify $YX \leqslant_{\beta_{sd}} a$ we can apply either the rule $(@_\pi)$ or the rule $(@_\beta)$.
   (a) *Rule $(@_\pi)$:* $X \leqslant_{\beta_{sd}} \lambda x\,.\,x,\ YX \leqslant_{\beta_{sd}} a \rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,x, Y \leqslant_{\beta_{sd}} \lambda x\,.\,a$.
   (b) *Rule $(@_\beta)$*
   $$X \leqslant_{\beta_{sd}} \lambda x\,.\,x,\ YX \leqslant_{\beta_{sd}} a \rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,x, Y \leqslant_{\beta_{sd}} \lambda x\,.\,x, X \leqslant_{\beta_{sd}} a$$
   $$\rightarrow X \leqslant_{\beta_{sd}} \lambda x\,.\,x, Y \leqslant_{\beta_{sd}} \lambda x\,.\,x,$$
   $$\lambda x\,.\,x \leqslant_{\beta_{sd}} a.$$

In the last case, the system is not in solved form (although no transformation rules can be applied) and thus it gives no solutions. The initial matching equation has only two solutions.

3.2 Properties

Proposition 17 (Termination) *The set of transformation rules of Figure 2 is terminating.*

Proposition 18 (Correctness) *For all systems $\mathbb{S}$ and $\mathbb{S}'$ such that $\mathbb{S} \nmapsto\!\!\!\!\twoheadrightarrow \mathbb{S}'$ and $\mathbb{S}'$ is in solved form, we have $\sigma_{\mathbb{S}'} \in \mathbb{M}(\mathbb{S})$.*

Proposition 19 (Completeness) *For any system $\mathbb{S}$, if $\varphi \in \mathbb{M}(\mathbb{S})$ then there exists a sequence of transformations starting from $\mathbb{S}$ and ending on a system $\mathbb{S}_n$ such that $\mathbb{S}_n$ is in solved form and $\sigma_{\mathbb{S}_n} \leq \varphi$.*

Proof. By induction on the appropriate extension of the $\Longrightarrow_{\beta_{sd}}$ on multisets.

Theorem 20 (Finite complete match set) *Let $A \leqslant_{\beta_{sd}} B$ be a matching equation and $\mathbb{M} = \{\sigma_{\mathbb{S}} \mid A \leqslant_{\beta_{sd}} B \nmapsto\!\!\!\!\twoheadrightarrow \mathbb{S}$ and $\mathbb{S}$ is in solved form $\}$. Then the set $\mathbb{M}$ is a complete match set for the equation $A \leqslant_{\beta_{sd}} B$. It is always finite.*

We can remark that there are some second-order β-match equations that have no solutions but that the corresponding β_{sd}-equation has a solution[2]:

Example 21 *Let $g(XY, XZ) \leqslant_{\beta} g(fa, fb)$ be a β-match equation with types $a : \iota_2$, $b : \iota_2$, $f : \iota_2 \to \iota_1$, $g : \iota_1 \to \iota_1 \to \iota_1$, $X : \iota_3 \to \iota_1$, $Y : \iota_3$ and $Z : \iota_3$.*

We consider the solutions of the β_{sd}-equations $XY \leqslant_{\beta_{sd}} fa$ and $XZ \leqslant_{\beta_{sd}} fb$

$XY \leqslant_{\beta_{sd}} fa$	$XZ \leqslant_{\beta_{sd}} fb$
$X \leqslant_{\beta_{sd}} f, \quad Y \leqslant_{\beta_{sd}} a$	$X \leqslant_{\beta_{sd}} f, \quad Z \leqslant_{\beta_{sd}} b$
$X \leqslant_{\beta_{sd}} \lambda x\,.\,fa$	$X \leqslant_{\beta_{sd}} \lambda x\,.\,fb$
$X \leqslant_{\beta_{sd}} \lambda x\,.\,fx, Y \leqslant_{\beta_{sd}} a$	$X \leqslant_{\beta_{sd}} \lambda x\,.\,fx, Z \leqslant_{\beta_{sd}} b$
$X \leqslant_{\beta_{sd}} \lambda x\,.\,xa, Y \leqslant_{\beta_{sd}} f$	$X \leqslant_{\beta_{sd}} \lambda x\,.\,xb, Z \leqslant_{\beta_{sd}} f$
$X \leqslant_{\beta_{sd}} \lambda x\,.\,x, \quad Y \leqslant_{\beta_{sd}} fa$	$X \leqslant_{\beta_{sd}} \lambda x\,.\,x, \quad Z \leqslant_{\beta_{sd}} fb$

The only two well-typed solutions (that is, solutions such that the term associated to a matching variable has the same type than this matching variable) are respectively $X \leqslant_{\beta_{sd}} \lambda x\,.\,fa$ and $X \leqslant_{\beta_{sd}} \lambda x\,.\,fb$. Of course, they do not lead to a substitution for $g(XY, XZ) \leqslant_{\beta_{sd}} g(fa, fb)$. Thus, we have found a second-order β-match equation that has no solution[3] even if the β_{sd}-equation has.

We now work in the framework of the typed λ-calculus *to solve second-order matching equations*. As in any higher-order matching algorithm for typed λ-calculi, we only consider well-typed equations that are pairs of typed terms of the *same* type. In particular, transformation rules are applied only if the resulting systems is well typed (that is, each equation is well-typed). In this context, we have the following result.

[2] Thus we cannot deduce from the NP-completeness of the second-order matching, the NP-completeness of the matching modulo superdeveloppements

[3] Since there is no well-typed substitutions modulo superdevelopments, there are no second-order substitution for $g(XY, XZ) \leqslant_{\beta} g(fa, fb)$ (applying prop. 12).

Theorem 22 (Second-order matching algorithm) *The rules given in Fig. 2 applied in the context of the typed λ-calculus gives a sound and complete matching algorithm for second-order matching.*

4 Matching modulo superdevelopments and η

The gap between higher-order matching modulo β and higher-order matching modulo $\beta\eta$ is mainly explained by the fundamental use of η-long normal forms when matching is performed modulo $\beta\eta$. In the context of higher-order matching modulo superdevelopments, the use of η-equivalence does not strongly influence our algorithm, as explained below.

A $\beta_{sd}\eta$-matching equation is a pair (A, B) of terms such that B is $\beta\eta$-normal and contains no matching variable. It is denoted by $A \leqslant^{\eta}_{\beta_{sd}} B$. A substitution φ is a $\beta_{sd}\eta$-match if there exists a term C such that $A\varphi \Longrightarrow_{\beta_{sd}} C \to^{*}_{\eta} B$. The algorithm described in Section 3 has to be adapted *w.r.t.* two aspects:

First, η-expansion is performed on demand by adding a rule to the matching algorithm:

$$(\lambda x\,.\,A \leqslant^{\eta}_{\beta_{sd}} B),\ \mathbb{S} \to_{\lambda_{-}} (A \leqslant^{\eta}_{\beta_{sd}} Bx), \mathbb{S} \quad \text{if } B \text{ is not a } \lambda\text{-abstraction and } x \text{ is fresh}$$

In one step, this rule first replaces the right hand side B by $\lambda y\,.\,(By)$ and then performs λ-abstraction elimination as in the rule (λ_{λ}).

Secondly, we must add a side condition in the rule $(@_{\beta})$ so that $\lambda x\,.\,A_2$ and A_1 are in $\beta\eta$-normal form (and not only in β-normal form).

The algorithm enjoys the same properties (termination, soundness and completeness) as before. Moreover, if we apply this algorithm to an equation whose first term is a pattern à la Miller then we obtain a complete match set consisting exactly of *the* more general match.

Example 23 *If we consider the match-equation of Ex. 15 we can remark that solving the equation modulo $\beta_{sd}\eta$ we get only 4 solutions. In fact, the rule $(@_{\beta})$ applies now only twice. The following two solutions found in Ex. 15 are η-equivalent: $X \leqslant_{\beta_{sd}} a$, $Y \leqslant_{\beta_{sd}} b$ and $X \leqslant_{\beta_{sd}} \lambda x\,.\,ax$, $Y \leqslant_{\beta_{sd}} b$.*

Example 24 *Consider the equation $(\lambda x\,.\,X(Yx), a)$. It has no β_{sd}-solution whereas it has two $\beta_{sd}\eta$-matches given by $\{a/X, \lambda z\,.\,z/Y\}$ and $\{\lambda z\,.\,z/X, a/Y\}$.*

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

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

Conclusion

We proposed a new approach to study higher-order matching following [dMS01]: instead of working in the typed λ-calculus modulo full β-reduction we propose to work in the untyped λ-calculus modulo a restriction of β-equivalence, namely superdevelopments. The essence of the restriction induces that all second-order β-matches are matches modulo superdevelopments. The algorithms are described in a mathematically elegant way that allow us to write intuitive proofs (termination, soundness and completeness). Since we consider untyped frameworks the use of the η-equivalence does not influence the behavior and design of our algorithms.

An implementation of the algorithm of matching modulo superdevelopments was done in the TOM language [MRV03].

Higher-order formalisms and especially higher-order rewriting generally choose the typed λ-calculus modulo β (or $\beta\eta$) as a meta-language. In the case of CRS [KvOvR93], the meta-language is the untyped λ-calculus with developments. The next step is thus to study higher-order rewriting with the untyped λ-calculus modulo superdevelopments as a meta-language (in other words to consider higher-order rewriting with the untyped λ-calculus with superdevelopments as a substitution calculus in the sense of [Oos94]).

As far as it concerns the transformations of pattern-matching programs, the work of [dMS01] motivates by several examples higher-order matching in pattern-calculi such as the ρ-calculus [CLW03] or pure pattern calculi [JK06]. Since a simple type system that ensures termination is difficult to find in this context, this paper should give useful guidelines.

Acknowledgments: 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. Finally, we sincerely thank the referees for their deep remarks on the paper.

References

[Acz78] P. Aczel. A general church rosser theorem. Technical report, University of Manchester, July 1978.

[Ali05] C. Alias. *Program Optimization by Template Recognition and Replacement*. PhD thesis, University of Versailles, Versailles, France, December 2005.

[Bar84] H. Barendregt. *The Lambda-Calculus, its syntax and semantics*. Elsevier Science Publishers B. V. (North-Holland), 1984.

[CLW03] H. Cirstea, L. Liquori, and B. Wack. Rewriting calculus with fixpoints: Untyped and first-order systems. volume 3085. Springer, 2003.

[DHK00] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157(1/2):183–235, 2000.

[dMS01] O. de Moor and G. Sittampalam. Higher-order matching for program transformation. *Theoretical Computer Science*, 269, 2001.

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

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

[GS89] J. Gallier and W. Snyder. Higher-order unification revisited: Complete sets of transformations. *JSCOMP: Journal of Symbolic Computation*, 8, 1989.

[HL78] G. Huet and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11, 1978.

[JK06] C. B. Jay and D. Kesner. Pure pattern calculus. In *Proceedings of the European Symposium on Programming (ESOP) LNCS 3924*, 2006.

[KvOvR93] Klop, van Oostrom, and van Raamsdonk. Combinatory reduction systems: Introduction and survey. *TCS: Theoretical Computer Science*, 121, 1993.

[Lév78] J.-J. Lévy. *Reductions Correctes et Optimales dans le Lambda-Calcul*. Ph.D. thesis, Université de Paris, 1978.

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

[Mil91] D. Miller. A logic programming language wiith lambda-abstraction, function variables, and simple unification. *Jour. of Log. and Comp.*, 1991.

[MN98] R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192, 1998.

[MRV03] P.-E. Moreau, C. Ringeissen, and M. Vittek. A pattern matching compiler for multiple target languages. In *Compiler Construction*, 2003.

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

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

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

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

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

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

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

[vOvR93] V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. volume 816 of *LNCS*, 1993.

[vR93] F. van Raamsdonk. Confluence and superdevelopments. *Rewriting Techniques and Applications*, 1993.

[Wac06] B. Wack. A Curry-Howard-De Bruijn Isomorphism Modulo. *Under submission*, 2006.