

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

Germain Faure

► To cite this version:

Germain Faure. Higher-order matching modulo (super)developements. Applications to second-order matching. [Research Report] 2009. inria-00429978

HAL Id: inria-00429978

<https://hal.inria.fr/inria-00429978>

Submitted on 5 Nov 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

3 GERMAIN FAURE

4 Parc Orsay Université, 4 rue Jacques Monod, 91893 Orsay Cedex, France
5 *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.

6 CONTENTS

7	1. Normalization in the lambda-calculus	3
8	2. Matching modulo beta (and eta)	9
9	3. Matching modulo superdevelopments (and eta)	11
10	4. Algorithm for matching modulo superdevelopments	15
11	5. Algorithm for matching modulo superdevelopments and eta	23
12	6. Algorithm for second-order matching	26
13	7. Algorithm for matching modulo developments (and eta)	31
14	8. Applications to higher-order rewriting	32

15 INTRODUCTION

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

20 Higher-order matching and unification in the λ -calculus cannot be studied directly since
21 this requires to decide the equality between two terms and the equality modulo β of terms
22 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.

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

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

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

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

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

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

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

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

66 Higher-order matching in an untyped setting was already been studied in [Sit01, dMS01]
 67 where matching equations are solved modulo a one-step reduction that generalizes the Tait

68 and Martin-Löf parallel reduction. But this one-step reduction is, like the authors of the
 69 original paper said, difficult to understand. We show in this paper that it is nothing but a
 70 parallel reduction that corresponds [vR96, vR93] to superdevelopments.

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

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

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

89 1. NORMALIZATION IN THE LAMBDA-CALCULUS

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

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

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

96 **1.1. Typed λ -calculus and β -reduction.** Given a set of base types \mathfrak{T}_0 , we define the set
 97 of types \mathfrak{T} inductively as the smallest set containing \mathfrak{T}_0 and such that if α and $\beta \in \mathfrak{T}$ then
 98 $(\alpha \rightarrow \beta) \in \mathfrak{T}$. The order of a type α denoted $\mathfrak{o}(\alpha)$ is equal to 1, if $\alpha \in \mathfrak{T}_0$. The order of a
 99 type $\alpha \rightarrow \beta$ is equal to $\max(\mathfrak{o}(\alpha) + 1, \mathfrak{o}(\beta))$.

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

- 106 • If $A, B \in \mathcal{T}_t$ with type *resp.* $\alpha \rightarrow \beta$ and α then $(A B) \in \mathcal{T}_t$ with type β .
- 107 • 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 $\alpha \rightarrow \beta$.

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

110 The symbols A, B, C, \dots range over the set \mathcal{T}_t of terms, the symbols x, y, z, \dots range
 111 over the set \mathcal{X} of variables ($\mathcal{X} \subseteq \mathcal{T}_t$), the symbols a, b, c, \dots, f, g, h range over a set \mathcal{K} of
 112 term constants ($\mathcal{K} \subseteq \mathcal{T}_t$). The symbols X, Y, \dots range over the set \mathcal{V} of matching variables.
 113 Finally, the symbol ε ranges over the set of atoms, which consists of variables, matching
 114 variables and constants. To increase the readability, we often write $\varepsilon(A_1, A_2, \dots, A_n)$ for
 115 $(\dots((\varepsilon A_1) A_2) \dots) A_n$ where ε is an atom and A_1, \dots, A_n are arbitrary terms. All symbols
 116 can be indexed. Positions in λ -terms are denoted by p_1, \dots, p_n . We denote by \preceq the order
 117 on positions (prefix order). The subterm of A at position p_1 is denoted by $A_{|p_1}$.

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

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

125 The substitution of variables is defined as usual and avoids variable capture using α -
 126 conversion when needed. The substitution of the variable x by A in B is denoted by
 127 $B[x := A]$.

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

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

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

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

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

142 A *substitution of matching variables* is a function from matching variables to the set
 143 of \mathcal{V} -closed terms. It is denoted $B\{A/X\}$. We use the standard definitions for domain,
 144 codomain, union and composition of substitutions. When it is clear from the context,
 145 substitutions of matching variables are simply called substitutions. Substitutions can be
 146 compare using the usual *subsumption order*: for two substitutions ψ and φ , we say that
 147 $\psi \leq \varphi$ when there exists a substitution ξ such that $\varphi = \xi \circ \psi$ where \circ denotes substitu-
 148 tion composition. In this work, we only consider *closed and normal substitutions* that are
 149 substitutions of closed and normal terms.

150 **1.2. Untyped λ -calculus and developments.** We define developments as a subset of
 151 the β -reduction which reduces only the redexes initially present in the term as well as its
 152 residuals. This is formalized by defining the *underlined λ -calculus*: we initially underline all
 153 the redexes present in the term and we replace the β -reduction by the β_u -reduction which
 154 only reduces underlined redexes. Then, the redexes created during reduction are no longer
 155 reduced (since they are not underlined).

156 **1.2.1. Untyped underlined λ -calculus and β_u -reduction.** We define the set of underlined
 157 terms. In case they do not use any type information, we use without ambiguity the defini-
 158 tions and notations given for typed λ -terms in the previous section.

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

- 163 • If A and B are elements of the set \mathcal{T}_u then (AB) is an element of \mathcal{T}_u ;
- 164 • 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 ;
- 165 • If A and B are elements of \mathcal{T}_u then $(\lambda x.A)B$ belong to \mathcal{T}_u .

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

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

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

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

172 **1.2.2. Untyped λ -calculus and developments.** The set of terms in the pure λ -calculus is
 173 defined in the same way of typed λ -terms except that we do take care of type constraints.
 174 It is denoted by \mathcal{T} . We can define a mapping Υ from underlined terms to terms that
 175 replaces β -redexes by their corresponding β -redexes¹. This mapping can be extended to
 176 any sequence of β_u -reductions. We can then define developments.

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

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

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

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

¹The 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

$$\begin{array}{c} \overline{\varepsilon} \Longrightarrow_{\beta} \varepsilon \quad (Red - \varepsilon) \\ \frac{A_1 \Longrightarrow_{\beta} A_2}{\lambda x. A_1 \Longrightarrow_{\beta} \lambda x. A_2} \quad (Red - \lambda) \\ \frac{A_1 \Longrightarrow_{\beta} A_2 \quad B_1 \Longrightarrow_{\beta} B_2}{A_1 B_1 \Longrightarrow_{\beta} A_2 B_2} \quad (Red - @) \quad \frac{\lambda x. A_1 \Longrightarrow_{\beta} \lambda x. A_2 \quad B_1 \Longrightarrow_{\beta} B_2}{(\lambda x. A_1) B_1 \Longrightarrow_{\beta} A_2[x := B_2]} \quad (Red - \beta) \end{array}$$

187 **Theorem 1.6** (Parallel reduction and developments). *The notions of parallel reduction and*
 188 *developments coincide in the following sense. For every terms $A, B \in \mathcal{T}$*
 189 *there exists a development $A \twoheadrightarrow_{\beta} B$ iff $A \Longrightarrow_{\beta} B$*

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

192 **1.3. Untyped λ -calculus and superdevelopments.** We have seen in the previous sec-
 193 tion that developments reduce the redexes initially present in the term and its residuals.
 194 This gives a first approximation of the β -normal form. Unfortunately, this approximation
 195 is too rough for being useful in the context of higher-order matching as we will see in
 196 Section 3.5.

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

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

$$\begin{array}{l} \text{(type 1)} \quad ((\lambda x. (\lambda y. A))B)C \quad \rightarrow_{\beta} \quad (\lambda y. A[x := B])C \\ \text{(type 2)} \quad ((\lambda x. x)(\lambda y. A))B \quad \rightarrow_{\beta} \quad (\lambda y. A)B \\ \text{(type 3)} \quad (\lambda x. A)(\lambda y. B) \quad \rightarrow_{\beta} \quad A[x := (\lambda y. B)] \\ \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{if } \exists p \text{ such that } A|_p = xA_0 \end{array}$$

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

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

209 1.3.1. *Untyped labelled λ -calculus and β_l -reduction.* By definition, labels are simply elements
210 of \mathbb{N} .

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

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

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

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

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

221 **Definition 1.8** (Well-labelled and initially labelled terms). A labelled term $A \in \mathcal{T}_l$ is said
222 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
223 $p_1 \preceq p_2$. It is *initially labelled* if moreover for all positions such that $A|_{p_1} = \lambda_p x.C$ and
224 $A|_{p_2} = \lambda_p x'.C'$ then $p_1 = p_2$.

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

227 1.3.2. *Untyped λ -calculus and superdevelopements.* Before giving the formal definition of su-
228 perdevelopements, we define an erasing morphism from labelled λ -terms to λ -terms. Without
229 any ambiguity, we overload the notation of the previous section.

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

- 232 • $\Upsilon(\varepsilon) = \varepsilon$
- 233 • $\Upsilon((AB)^p) = \Upsilon(A)\Upsilon(B)$
- 234 • $\Upsilon(\lambda_p x.A) = \lambda x.\Upsilon(A)$

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

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

241 **Definition 1.10** (Superdevelopements). A β -rewrite sequence ς of the λ -calculus is a β -
242 superdevelopment if there exists a β_l -rewrite sequence σ in the labelled λ -calculus that
243 starts with an initially labelled term and stops on a term in β_l -normal form and such that
244 $\Upsilon(\sigma) = \varsigma$.

245 We extend the finite developments theorem to superdevelopements.

246 **Theorem 1.11** (Finite superdevelopements [vR96]).

- 247 • *Every superdevelopment is finite.*

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

250 As finite developments coincide with the parallel reduction of Tait and Martin-Löf, finite
251 superdevelopments coincide with Aczel's parallel reduction [Acz78] called in the following
252 strong parallel reduction. It is denoted by $\Rightarrow_{\beta_{sd}}$ and we say that a term A β_{sd} -reduces to a
253 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

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

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

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

254 The only difference with the parallel reduction of Tait and Martin-Löf is the rule
255 $(Red-\beta_s)$ that replaces the rule $(Red-\beta)$ of the parallel reduction. The redex reduced
256 by the rule $(Red-\beta_s)$ is obtained from the *reduct* of A_1 . This redex is not necessarily
257 present in the initial term $A_1 A_2$. It may have been created and this creation is of type 1
258 or 2 but not of type 3 (an upwards creation but not a downwards creation).

259 **Theorem 1.13** (Strong parallel reduction and superdevelopments). *The notions of strong
260 parallel reduction and superdevelopments coincide in the following sense. For every terms
261 $A, B \in \mathcal{T}$*

262 *there exists a superdevelopment $A \rightarrow_{\beta} B$ iff $A \Rightarrow_{\beta_{sd}} B$.*

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

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

- The β -rewrite sequence

$$(\lambda x. \lambda y. xy)zz' \rightarrow_{\beta} (\lambda y. zy)z' \rightarrow_{\beta} zz'$$

is a superdevelopment since it corresponds to the β_l -rewrite sequence

$$(((\lambda_1 x. \lambda_2 y. xy)z)^1 z')^2 \rightarrow_{\beta_l} ((\lambda_2 y. zy)z')^2 \rightarrow_{\beta_l} zz' .$$

- However, the rewrite sequence

$$(\lambda x. xx)(\lambda x. xx) \rightarrow_{\beta} (\lambda x. xx)(\lambda x. xx) \rightarrow_{\beta} \dots$$

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

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

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

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

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

274 **Example 1.15.** In this example, we illustrate the link with developments, superdevelop-
275 ments and redex creations.

276 • **Finite development** Residuals of redexes present in the initial term can be con-
277 tracted:

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

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

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

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

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

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

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

284 2. MATCHING MODULO BETA (AND ETA)

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

287 **2.1. Matching modulo β .**

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

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

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

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

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

299 **2.2. Nth order matching.** In practice, we only consider algorithms to solve a subset of
 300 β -equations (higher-order matching modulo β is undecidable [Loa03]). Either we consider
 301 some restrictions on the order of matching variables and constants or we consider matching
 302 modulo $\beta\eta$.

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

308 For example, the above equation $XY \leq_{\beta} a$ 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

$$(\eta) \quad \lambda x. (Ax) \quad = \quad A$$

if $x \notin \text{fv}(A)$

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

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

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

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

320 **2.4. Matching of Miller patterns.** We first define Miller patterns.

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

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

329 3. MATCHING MODULO SUPERDEVELOPMENTS (AND ETA)

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

332 **3.1. Matching modulo superdevelopments.** In this section, we consider terms of pure
333 λ -calculus (elements of \mathcal{T}). We are going to solve equations modulo superdevelopments.
334 We first define the notion of equation and then the corresponding notion of solutions.

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

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

340 For example, the pairs $(XY, \lambda x. x)$ and $((\lambda x. x)X, a)$ are β_{sd} -matching equations whereas
341 $(XY, (\lambda x. x)a)$ is not since the term $(\lambda x. x)a$ is not normal.

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

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

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

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

356 The application of a substitution to a matching equation $B \leq_{\beta_{sd}} C$ is the equation
357 $B\varphi \leq_{\beta_{sd}} C$. The application of a substitution φ to a system, denoted $\mathbb{S}\varphi$ consists in the
358 application of the substitution φ to each matching equation of \mathbb{S} .

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

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

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

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

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

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

359 even if these terms are β -convertible.

360 A β_{sd} -system can have an infinite number of solutions but we can always find a finite
361 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,

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

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

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

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

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

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

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

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

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

379 The following lemma gives the relevance of solved forms.

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

381 **3.2. Comparison with second-order matching.** In this section, we are going to prove
 382 that matching modulo superdevelopments is complete for second-order matching. The fol-
 383 lowing results were already proved in [dMS01]. Nevertheless, the technical material pre-
 384 sented here (creations of redexes, superdevelopments) gives simple and intuitive proofs.
 385 First, a technical result on the creation of redexes.

386 **Lemma 3.9.** *For all terms A_1, A_2, \dots, A_n such that A_n contains a redex of third order (or
 387 more) and there exists a superdevelopment $A_1 \rightarrow_{\beta} A_2 \rightarrow_{\beta} \dots \rightarrow_{\beta} A_n$, then A_1 contains also
 388 a redex of third order (or more).*

389 *Proof.* We prove the result by induction on n . We look at the induction case. By induction
 390 hypothesis, we know that A_2 contains a redex of a least third order that we call in the
 391 following $R = (\lambda x. C) D$. First, if R is a residual of a redex of A_1 then the result is obvious.
 392 Secondly, if not, and if R is created during the reduction from A_1 to A_2 in the first way
 393 mentioned before then A_1 must contain a subterm of the form $((\lambda z. \lambda x. C') E) D$ with
 394 $C = C'[z := E]$. Then the order of the redex $(\lambda z. \lambda x. C') E$ is greater or equal to the order
 395 of R . This concludes the case. Finally, if not, and if R is created during the reduction from
 396 A_1 to A_2 in the second way mentioned before then A_1 must contain a subterm of the form
 397 $(\lambda y. y) (\lambda x. C) D$. The order of the redex $(\lambda y. y) (\lambda x. C)$ is strictly greater than the one of
 398 R . This concludes the case. \square

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

401 *Proof.* The proof is by contradiction. Let φ be a β -match of the β -matching equation
 402 $A \leq_{\beta} B$ that is not a β_{sd} -match. Then we have that A does not contain any β -redex and
 403 that φ does not contain any term of order greater than 2. Finally, $A\varphi \not\equiv_{\beta_{sd}} B$ and $A\varphi =_{\beta} B$.
 404 Thus there exist $(A_i)_i$ such that $A\varphi \rightarrow_{\beta} A_1 \rightarrow_{\beta} \dots \rightarrow_{\beta_{sd}} A_n$ is a superdevelopments and
 405 A_n contains a β -redex $(\lambda x. C)D$ which is not reduced by superdevelopments. This means
 406 that this redex is a residual of a redex created when reducing A_{i_0} . Since the redex is not
 407 reduced by superdevelopments then this creation is of type 3 and thus induces a redex of
 408 order at least 3. Lemma 3.9 implies that $A\varphi$ contains a redex of order at least 3. Since
 409 both A and φ range in the set of β -normal forms, then there exists a position p_1 and a term
 410 E such that $A|_{p_1} = XE$ where X is mapped by φ to a λ -abstraction of at least third order.
 411 This contradicts the hypothesis on the order of the initial matching problem. \square

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

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

418 **3.3. Comparison with third-order matching.** As soon as we consider third-order match-
 419 ing problems, the set of minimal solutions may be infinite. Since matching modulo superde-
 420 velopments generates finitely many minimal solutions, we remark that matching modulo
 421 superdevelopments cannot be complete *w.r.t.* third-order matching.

422 **Example 3.11.** The substitution $\{\lambda x. \lambda f. fx/X\}$ is a β -match for the matching equation
 423 $\lambda z. (X z (\lambda y. y)) \leq \lambda z. z$ whereas it is not a β_{sd} -match. In fact, $\lambda z. ((\lambda x. \lambda f. fx) z (\lambda y. y))$
 424 β_{sd} -reduces to $\lambda z. (\lambda y. y)z$ but not to $\lambda z. z$.

425 The last example is classical and taken from [Dow01]. The third-order matching equa-
 426 tion has an infinite number of (minimal) solutions of type $\iota \rightarrow (\iota \rightarrow \iota) \rightarrow \iota$ that are given
 427 by the Church numbers $\lambda x. \lambda f. (f \dots (f x) \dots)$.

428 **3.4. Comparison with matching of Miller patterns.** In the case of matching of Miller
 429 patterns [Mil91, Qia96], the restriction of the β -reduction given by superdevelopments is
 430 powerful enough:

431 **Proposition 3.12.** *Let φ be a match of an equation $P \leq_{\beta} A$ where P is a Miller pattern.*
 432 *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

$$(\lambda y. M)x \quad \rightarrow_{\beta_0} \quad M[y := x]$$

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

437 **3.5. Matching modulo superdevelopments and eta.** In the simply typed λ -calculus,
 438 matching modulo $\beta\eta$ strongly relies on the η -long normal form. This makes an important
 439 difference with matching modulo β : using the η -equivalence, we move from a undecid-
 440 able [Loa03] to a decidable problem [Sti09].

441 In the context of matching modulo superdevelopments, the use of η does not influence
 442 neither fundamentally nor technically the design of matching algorithms, as we will see in
 443 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.

$$\lambda x. (Ax) \quad \rightarrow_{\eta} \quad A$$

if $x \notin \text{fv}(A)$

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

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

449 These two notions will be illustrated in Section 5.

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

Figure 1: Matching modulo superdevelopments

450

4. ALGORITHM FOR MATCHING MODULO SUPERDEVELOPMENTS

451

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.

452

453

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,

454

455

456

- a system is transformed step by step until getting to a normal form (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);

457

458

459

- 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).

460

461

462

463

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 $n \geq 0$ systems $\mathbb{S}_1, \dots, \mathbb{S}_n$ such that $\mathbb{S} = \mathbb{S}_1 \rightarrow \dots \rightarrow \mathbb{S}_n = \mathbb{S}'$.

464

465

466

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.

467

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

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

$$(X \leq_{\beta_{sd}} A) \cup \mathbb{S} \xrightarrow{\varepsilon_x^\downarrow} (X \leq_{\beta_{sd}} A) \cup (\mathbb{S}\{A/X\}) \downarrow$$

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

and the following equation

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

which can be transformed into

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

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

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

476 We obtain a normal form in solved form whereas the corresponding substitution is not a
 477 match modulo superdevelopments (even if this is of course a match modulo β): the rule
 478 $(\varepsilon_x^\downarrow)$ is thus not sound.

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

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

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

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

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

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

508 **Example 4.1** (Computing the solutions of a β_{sd} -equation). We consider the equation
 509 $XY \leq_{\beta_{sd}} ab$. As the left and right hand sides of the equation are applications, we can
 510 apply the rules $(@_{@})$, $(@_{\pi})$ or $(@_{\beta})$.

(1) Rule $(@_{@})$:

$$(XY \leq_{\beta_{sd}} ab) \rightarrow (X \leq_{\beta_{sd}} a) \cup (Y \leq_{\beta_{sd}} b).$$

(2) Rule $(@_{\pi})$:

$$(XY \leq_{\beta_{sd}} ab) \rightarrow (X \leq_{\beta_{sd}} \lambda x. ab).$$

511 (3) Rule $(@_{\beta})$: to find A_1 and A_2 such that $A_1[x := A_2] = ab$, we first choose A_2 as
 512 a subterm of “ ab ”: a , b and ab . The set of positions of ab for which A_2 is the
 513 corresponding subterm of C is a singleton since each subterm of ab appears only
 514 once in ab . We obtain three ways to apply the rule $(@_{\beta})$ corresponding to the three
 515 subterms of the right hand side of the equation $XY \leq_{\beta_{sd}} ab$:

516 (a) $(XY \leq_{\beta_{sd}} ab) \rightarrow (X \leq_{\beta_{sd}} \lambda x. xb) \cup (Y \leq_{\beta_{sd}} a)$.

517 (b) $(XY \leq_{\beta_{sd}} ab) \rightarrow (X \leq_{\beta_{sd}} \lambda x. ax) \cup (Y \leq_{\beta_{sd}} b)$.

518 (c) $(XY \leq_{\beta_{sd}} ab) \rightarrow (X \leq_{\beta_{sd}} \lambda x. x) \cup (Y \leq_{\beta_{sd}} ab)$.

519 **Example 4.2** (Computing the solutions of a β_{sd} -equation). We consider the equation
 520 $X(YX) \leq_{\beta_{sd}} a$. We can apply the rules $(@_{\pi})$ or $(@_{\beta})$.

(1) Rule $(@_{\pi})$:

$$(X(YX) \leq_{\beta_{sd}} a) \rightarrow (X \leq_{\beta_{sd}} \lambda x. a).$$

(2) Rule $(@_{\beta})$:

$$(X(YX) \leq_{\beta_{sd}} a) \rightarrow (X \leq_{\beta_{sd}} \lambda x. x) \cup (YX \leq_{\beta_{sd}} a).$$

521 To simplify $YX \leq_{\beta_{sd}} a$ we can apply the rule $(@_{\pi})$ or the rule $(@_{\beta})$.

(a) Rule $(@_{\pi})$:

$$(X \leq_{\beta_{sd}} \lambda x. x) \cup (YX \leq_{\beta_{sd}} a) \rightarrow (X \leq_{\beta_{sd}} \lambda x. x) \cup (Y \leq_{\beta_{sd}} \lambda x. a).$$

(b) Rule $(@_{\beta})$:

$$\begin{aligned} & (X \leq_{\beta_{sd}} \lambda x. x) \cup (YX \leq_{\beta_{sd}} a) \\ \rightarrow & (X \leq_{\beta_{sd}} \lambda x. x) \cup (Y \leq_{\beta_{sd}} \lambda x. x) \cup (X \leq_{\beta_{sd}} a) \\ \rightarrow & (X \leq_{\beta_{sd}} \lambda x. x) \cup (Y \leq_{\beta_{sd}} \lambda x. x) \cup (\lambda x. x \leq_{\beta_{sd}} a). \end{aligned}$$

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

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

529 **4.2. Termination property.** We show that the algorithm is terminating. We first define
 530 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

$$\begin{aligned} \mathfrak{S}(\varepsilon) &= 1 && \text{for all atoms } \varepsilon \in \mathcal{E} \\ \mathfrak{S}(\lambda x. B) &= \mathfrak{S}(B) + 1 \\ \mathfrak{S}(BC) &= \mathfrak{S}(B) + \mathfrak{S}(C) + 1 \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned}
@_{@} \quad & \mathfrak{G}((A_1 B_1 \leq_{\beta_{sd}} A_2 B_2) \cup \mathbb{S}) \\
& = \mathfrak{G}(A_1 B_1) + \mathfrak{G}(\mathbb{S}) \\
& = 1 + \mathfrak{G}(A_1) + \mathfrak{G}(B_1) + \mathfrak{G}(\mathbb{S}) \\
& > \mathfrak{G}(A_1 \leq_{\beta_{sd}} A_2) + \mathfrak{G}(B_1 \leq_{\beta_{sd}} B_2) + \mathfrak{G}(\mathbb{S}) \\
@_{\pi} \quad & \mathfrak{G}((A_1 B_1 \leq_{\beta_{sd}} A_2) \cup \mathbb{S}) \\
& = \mathfrak{G}(A_1 B_1) + \mathfrak{G}(\mathbb{S}) \\
& = 1 + \mathfrak{G}(A_1) + \mathfrak{G}(B_1) + \mathfrak{G}(\mathbb{S}) \\
& > \mathfrak{G}(A_1 \leq_{\beta_{sd}} \lambda x. A_2) + \mathfrak{G}(\mathbb{S}) \\
@_{\beta} \quad & \mathfrak{G}((A_1 B_1 \leq_{\beta_{sd}} C) \cup \mathbb{S}) \\
& = \mathfrak{G}(A_1 B_1) + \mathfrak{G}(\mathbb{S}) \\
& = 1 + \mathfrak{G}(A_1) + \mathfrak{G}(B_1) + \mathfrak{G}(\mathbb{S}) \\
& > \mathfrak{G}(A_1 \leq_{\beta_{sd}} \lambda x. A_2) + \mathfrak{G}(B_1 \leq_{\beta_{sd}} B_2) + \mathfrak{G}(\mathbb{S})
\end{aligned}$$

for all normal terms A_2 and B_2

546

□

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

549 *Proof.* For all systems \mathbb{S} and \mathbb{S}' such that $\mathbb{S} \rightarrow \mathbb{S}'$, the lexical product of the number of
550 unsolved variables $\mathfrak{U}(-)$ and the size of the system $\mathfrak{G}(-)$ decreases for each rule:

	$\mathfrak{U}(-)$	$\mathfrak{G}(-)$
ε_v	=	>
ε_c	=	>
λ_λ	\geq	>
ε_X	>	
$@_{@}$	\geq	>
$@_{\pi}$	\geq	>
$@_{\beta}$	\geq	>

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

553 **4.3. Completeness property.** We define an extension of the relation $\Rightarrow_{\beta_{sd}}$ to multisets.
554 To ease the reading in this section, we write $(A_1, B_1) \in \beta_s$ for $A_1 \Rightarrow_{\beta_{sd}} B_1$.

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

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

$$\frac{(A'_1, B'_1) \in \beta_s \quad (A''_1, B''_1) \in \beta_s}{(A_1, B_1) \in \beta_s} \quad \frac{(A'_1, B'_1) \cup (A''_1, B''_1) \cup E \in \beta_s}{(A_1, B_1) \cup E \in \beta_s}$$

$$\frac{(A'_1, B'_1) \in \beta_s}{(A_1, B_1) \in \beta_s} \quad \frac{(A'_1, B'_1) \cup E \in \beta_s}{(A_1, B_1) \cup E \in \beta_s}$$

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

$$\mathbb{S} = \mathbb{S}_0 \rightarrow \mathbb{S}_1 \rightarrow \dots \rightarrow \mathbb{S}_n$$

559 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

$$\mathbb{S} = \mathbb{S}_0 \rightarrow \mathbb{S}_1 \rightarrow \dots \rightarrow \mathbb{S}_n$$

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

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

560 Note that $\varphi \in \mathbb{M}(\mathbb{S}_0)$ is equivalent to $E_0 \in \beta_s$. We show the proposition by induction on
 561 $E_0 \in \beta_s$.

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

$$\begin{aligned} \mathbb{S}'_0 &= (A_2 \leq_{\beta_{sd}} B_2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ &\rightarrow \dots \\ &\rightarrow \mathbb{S}'_n \end{aligned}$$

564 such that $\sigma_{\mathbb{S}'_n} \leq \varphi$. Thus if $X \in \text{Dom}(\sigma'_n)$ then $X\sigma'_n = B_1$.

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

$$\begin{aligned} \mathbb{S}_0 &\xrightarrow[\varepsilon_v]{\varepsilon_c} (A_2 \leq_{\beta_{sd}} B_2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ &\rightarrow \dots \\ &\rightarrow \mathbb{S}'_n \end{aligned}$$

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

$$\begin{aligned} & (X \leq_{\beta_{sd}} B_1) \cup (A_2 \leq_{\beta_{sd}} B_2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & \dots \\ \rightarrow & (X \leq_{\beta_{sd}} B_1) \cup \mathbb{S}'_n \end{aligned}$$

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:

$$\begin{aligned} & (X \leq_{\beta_{sd}} B_1) \cup (A_2 \leq_{\beta_{sd}} B_2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & \dots \\ \rightarrow_{i_0} & (B_1 \leq_{\beta_{sd}} B_1) \cup \mathbb{S}'_{i_0+1} \\ \rightarrow & \mathbb{S}'_{i_0+1} \\ \rightarrow & \dots \\ \rightarrow & \mathbb{S}'_n \end{aligned}$$

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

$$\begin{aligned} & (A_1^1 \leq_{\beta_{sd}} B_1^1) \cup (A_1^2 \leq_{\beta_{sd}} B_1^2) \cup (A_2 \leq_{\beta_{sd}} B_2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & \dots \\ \rightarrow & \mathbb{S}_n \end{aligned}$$

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

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)

$$\begin{aligned} & (A_2 \leq_{\beta_{sd}} B_2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & \dots \\ \rightarrow & \mathbb{S}'_n \end{aligned}$$

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

If not then the derivation

$$\begin{aligned} & (A_1 \leq_{\beta_{sd}} B_1) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ = & (A_1^1 A_1^2 \leq_{\beta_{sd}} B_1^1 B_1^2) \cup (A_2 \leq_{\beta_{sd}} B_2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & (A_1^1 \leq_{\beta_{sd}} B_1^1) \cup (A_1^2 \leq_{\beta_{sd}} B_1^2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & \dots \\ \rightarrow & \mathbb{S}_n \end{aligned}$$

571 is the desired one.

- **Rule (Red – β_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

$$\begin{aligned} & (A_1^1 \leq_{\beta_{sd}} \lambda x. B_1^1) \cup (A_1^2 \leq_{\beta_{sd}} B_1^2) \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & \dots \\ \rightarrow & \mathbb{S}'_n \end{aligned}$$

with $\sigma'_{\mathbb{S}'_n} \leq \varphi$. The derivation

$$\begin{aligned} & (A_1^1 A_1^2 \leq_{\beta_{sd}} B_1^1[x := B_1^2]) \cup (A_1^1 \leq_{\beta_{sd}} \lambda x. B_1^1) \cup (A_1^2 \leq_{\beta_{sd}} B_1^2) \\ & \quad \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & \dots \\ \rightarrow & \mathbb{S}_n \end{aligned}$$

is the desired one if $x \in \text{fv}(B_1^1)$. If not, the derivation

$$\begin{aligned} & (A_1^1 A_1^2 \leq_{\beta_{sd}} B_1^1[x := B_1^2]) \cup (A_1^1 \leq_{\beta_{sd}} \lambda x. B_1^1) \\ & \quad \cup \dots \cup (A_p \leq_{\beta_{sd}} B_p) \\ \rightarrow & \dots \\ \rightarrow & \mathbb{S}_n \end{aligned}$$

is the desired one

572

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

574

□

575 4.4. Correctness property.

576 **Lemma 4.11.** For all systems \mathbb{S} and \mathbb{S}' such that $\mathbb{S} \rightarrow \mathbb{S}'$ using the rules (ε_c) , (ε_v) , (λ_λ)
577 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 $\text{fv}(A) = \emptyset$ and $X \in \mathbb{S}$.

$$\begin{aligned} \varphi \in \mathbb{M}((X \leq_{\beta_{sd}} A) \cup \mathbb{S}) & \Leftrightarrow X\varphi \Rightarrow_{\beta_{sd}} A \text{ and } \varphi \in \mathbb{M}(\mathbb{S}) \\ & \stackrel{\varphi \text{ normal}}{\Leftrightarrow} X\varphi = A \text{ and } \varphi \circ \{A/X\} \in \mathbb{M}(\mathbb{S}) \\ & \Leftrightarrow X\varphi \Rightarrow_{\beta_{sd}} A \text{ and } \varphi \in \mathbb{M}(\mathbb{S}\{A/X\}) \\ & \Leftrightarrow \varphi \in \mathbb{M}((X \leq_{\beta_{sd}} A) \cup \mathbb{S}\{A/X\}) \end{aligned}$$

578

□

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

Proof. We prove the result for each rule.

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

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

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

581

□

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

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

586 **4.5. Finite complete match set property.** The correctness and completeness properties
 587 entail that there exists a complete match set. It is obtained by exploring all the possible
 588 reductions and by constructing the set of substitutions obtained from the solved forms. We
 589 know that this process is finite since the algorithm terminates and thus the complete match
 590 set is also finite.

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

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

593

5. ALGORITHM FOR MATCHING MODULO SUPERDEVELOPMENTS AND ETA

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

$(\lambda\bar{x}_n. x \leq_{\beta_{sd}} \lambda\bar{x}_n. x) \cup \mathbb{S}$	$\rightarrow_{\varepsilon_v}$	\mathbb{S}
$(\lambda\bar{x}_n. a \leq_{\beta_{sd}} \lambda\bar{x}_n. a) \cup \mathbb{S}$	$\rightarrow_{\varepsilon_c}$	\mathbb{S}
$(\lambda\bar{x}_n. X \leq_{\beta_{sd}} \lambda\bar{x}_n. A) \cup \mathbb{S}$	$\rightarrow_{\varepsilon_X}$	$(\lambda\bar{x}_n. X \leq_{\beta_{sd}} \lambda\bar{x}_n. A) \cup \mathbb{S}\{A/X\}$ if $\text{fv}(A) = \emptyset$ and $X \in \mathbb{S}$
$(\lambda\bar{x}_n. A \leq_{\beta_{sd}} \lambda\bar{x}_{n-k}. B) \cup \mathbb{S}$	\rightarrow_{λ_-}	$(\lambda\bar{x}_n. A \leq_{\beta_{sd}} \lambda\bar{x}_{n-k+1}. Bx_{n-k+1}) \cup \mathbb{S}$
$(\lambda\bar{x}_n. (A_1B_1) \leq_{\beta_{sd}} \lambda\bar{x}_n. (A_2B_2)) \cup \mathbb{S}$	$\rightarrow_{@@}$	$(\lambda\bar{x}_n. A_1 \leq_{\beta_{sd}} \lambda\bar{x}_n. A_2) \cup$ $(\lambda\bar{x}_n. B_1 \leq_{\beta_{sd}} \lambda\bar{x}_n. B_2) \cup \mathbb{S}$
$(\lambda\bar{x}_n. (A_1B_1) \leq_{\beta_{sd}} \lambda\bar{x}_n. C) \cup \mathbb{S}$	$\rightarrow_{@_\pi}$	$(\lambda\bar{x}_n. A_1 \leq_{\beta_{sd}} \lambda\bar{x}_n. (\lambda y. C)) \cup \mathbb{S}$ where y fresh variable
$(\lambda\bar{x}_n. (A_1B_1) \leq_{\beta_{sd}} \lambda\bar{x}_n. C) \cup \mathbb{S}$	$\rightarrow_{@_\beta}$	$(\lambda\bar{x}_n. A_1 \leq_{\beta_{sd}} \lambda\bar{x}_n. (\lambda x. A_2)) \cup$ $(\lambda\bar{x}_n. B_1 \leq_{\beta_{sd}} \lambda\bar{x}_n. B_2) \cup \mathbb{S}$ where $A_2[x := B_2] = C$ and x fresh variable, $x \in \text{fv}(A_2)$ and $\lambda x. A_2, B_2$ $\beta\eta$-normal

Figure 2: Transformation for matching modulo superdevelopments and η

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

$$(\lambda x. A \leq_{\beta_{sd}}^\eta B) \cup \mathbb{S} \rightarrow (A \leq_{\beta_{sd}}^\eta Bx) \cup \mathbb{S}$$

if B is not λ -abstraction
and x is a fresh variable

594 This rule replaces the right hand side B by $\lambda x. Bx$ and eliminates (like in the rule (λ_λ) of
595 Figure 1) one head λ -abstraction. Then, we must add a side condition to the rule $(@_\beta)$ for
596 $\lambda x. A_2$ and A_1 to be in $\beta\eta$ -normal form (and not only in β -normal form).

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

601 We can prove that this algorithm has the termination, soundness and completeness
602 property. The termination and the correctness proofs are similar to the one given in the
603 previous section. The completeness proof is slightly more technical although it is funda-
604 mentally 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

$$(X \leq_{\beta_{sd}} a) \cup (Y \leq_{\beta_{sd}} b) \quad \text{and} \quad (X \leq_{\beta_{sd}} \lambda x. ax) \cup (Y \leq_{\beta_{sd}} b)$$

605 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

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

In fact,

$$\begin{aligned} (\lambda x. (X(Yx) \leq_{\beta_{sd}}^{\eta} a)) &\rightarrow (\lambda x. X(Yx) \leq_{\beta_{sd}}^{\eta} \lambda x. ax) \\ &\rightarrow (\lambda x. X \leq_{\beta_{sd}}^{\eta} \lambda x. a) \cup (\lambda x. Yx \leq_{\beta_{sd}}^{\eta} \lambda x. x) \\ &\rightarrow (\lambda x. X \leq_{\beta_{sd}}^{\eta} \lambda x. a) \cup (\lambda x. Y \leq_{\beta_{sd}}^{\eta} \lambda x. \lambda z. z) \end{aligned}$$

$$\begin{aligned} (\lambda x. X(Yx) \leq_{\beta_{sd}}^{\eta} a) &\rightarrow (\lambda x. X(Yx) \leq_{\beta_{sd}}^{\eta} \lambda x. ax) \\ &\rightarrow (\lambda x. X \leq_{\beta_{sd}}^{\eta} \lambda x. \lambda z. z) \cup (\lambda x. Yx \leq_{\beta_{sd}}^{\eta} \lambda x. ax) \\ &\rightarrow (\lambda x. X \leq_{\beta_{sd}}^{\eta} \lambda x. \lambda z. z) \cup (\lambda x. Y \leq_{\beta_{sd}}^{\eta} \lambda x. a) \end{aligned}$$

606 **5.2. Minimality for Miller patterns.** In Section 3.4, we show that matching modulo
607 superdevelopments is complete for matching of Miller patterns. In this section, by Miller
608 pattern equations we mean a β_{sd} -equation whose first term is a Miller pattern.

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

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

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

- (1) If $P = \lambda \bar{x}_n. f(A_1, \dots, A_n)$ then for the equation to be a solution, the term B must be of the form $B = \lambda \bar{x}_n. f(B_1, \dots, B_p)$. All derivations that lead to a solution must reduce $P \leq_{\beta_{sd}} B$ into

$$(\lambda \bar{x}_n. A_1 \leq_{\beta_{sd}} \lambda \bar{x}_n. B_1) \cup \dots \cup (\lambda \bar{x}_n. A_p \leq_{\beta_{sd}} \lambda \bar{x}_n. B_p)$$

620 (modulo the application order of the rules).

Since $\lambda \bar{x}_n. A_1, \dots, \lambda \bar{x}_n. A_p$ are Miller patterns we can apply the induction hypothesis and conclude that

$$\lambda \bar{x}_n. A_1 \leq_{\beta_{sd}} \lambda \bar{x}_n. B_1 \quad \text{has } \sigma_1 \text{ as a unique solution}$$

...

$$\lambda \bar{x}_n. A_p \leq_{\beta_{sd}} \lambda \bar{x}_n. B_p \quad \text{has } \sigma_p \text{ as a unique solution}$$

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

- 622 (2) The case $P = \lambda\bar{x}_n.x(A_1, \dots, A_n)$ is similar to the previous one.
 (3) If $P = \lambda\bar{x}_n.X(x_{i_1}, \dots, x_{i_p})$ where the variables x_{i_1}, \dots, x_{i_p} are pairwise distinct. Then three rules $(@_\pi)$, $(@_\alpha)$ 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 $(@_\alpha)$ leads to a solution then we must have $B = B_1x_{i_p}$. In the same way, if the rule $(@_\beta)$ leads to a solution then

$$\begin{aligned}
 (\lambda\bar{x}_n.X(x_{i_1}, \dots, x_{i_p}) \leq_{\beta_{sd}} \lambda\bar{x}_n.B) &\rightarrow (\lambda\bar{x}_n.X(x_{i_1}, \dots, x_{i_{p-1}}) \leq_{\beta_{sd}} \lambda\bar{x}_n.\lambda z.C_1 \cup) \\
 &(\lambda\bar{x}_n.x_{i_p} \leq_{\beta_{sd}} \lambda\bar{x}_n.x_{i_p}) \\
 &\text{with } B = C_1[z := x_{i_p}] \text{ and } z \in \text{fv}(C_1)
 \end{aligned}$$

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

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

$$\begin{aligned}
 (\lambda\bar{x}_n.X(x_{i_1}, \dots, x_{i_p}) \leq_{\beta_{sd}} \lambda\bar{x}_n.B) \\
 \rightarrow (\lambda\bar{x}_n.X(x_{i_1}, \dots, x_{i_{p-1}}) \leq_{\beta_{sd}} \lambda\bar{x}_n.\lambda y.B)
 \end{aligned}$$

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

Let us suppose now that the application of the rule $(@_\alpha)$ 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

$$\begin{aligned}
 (\lambda\bar{x}_n.X(x_{i_1}, \dots, x_{i_p}) \leq_{\beta_{sd}} \lambda\bar{x}_n.B_1x_{i_p}) \\
 \rightarrow (\lambda\bar{x}_n.X(x_{i_1}, \dots, x_{i_{p-1}}) \leq_{\beta_{sd}} \lambda\bar{x}_n.B_1) \cup (\lambda\bar{x}_n.x_{i_p} \leq_{\beta_{sd}} \lambda\bar{x}_n.x_{i_p})
 \end{aligned}$$

626 From the second equation, we deduce $x_{i_p} \in \text{fv}(B)$ and thus (see below) the rule
 627 $(@_\pi)$ does not lead to a solution. From the first equation we deduce $x_{i_p} \notin \text{fv}(B_1)$.
 628 Let us try to apply the rule $(@_\beta)$. We are looking for a term C_1 such that we
 629 have $B_1x_n = C_1[z := x_n]$. But since $x_{i_p} \notin \text{fv}(B_1)$, we must have $C_1 = B_1z$ with
 630 $z \notin \text{fv}(B_1)$ but the term $\lambda z.B_1z$ is not $\beta\eta$ -normal and thus the rule cannot be
 631 applied. This concludes the proof.

632 □

633 6. ALGORITHMS FOR SECOND-ORDER MATCHING

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

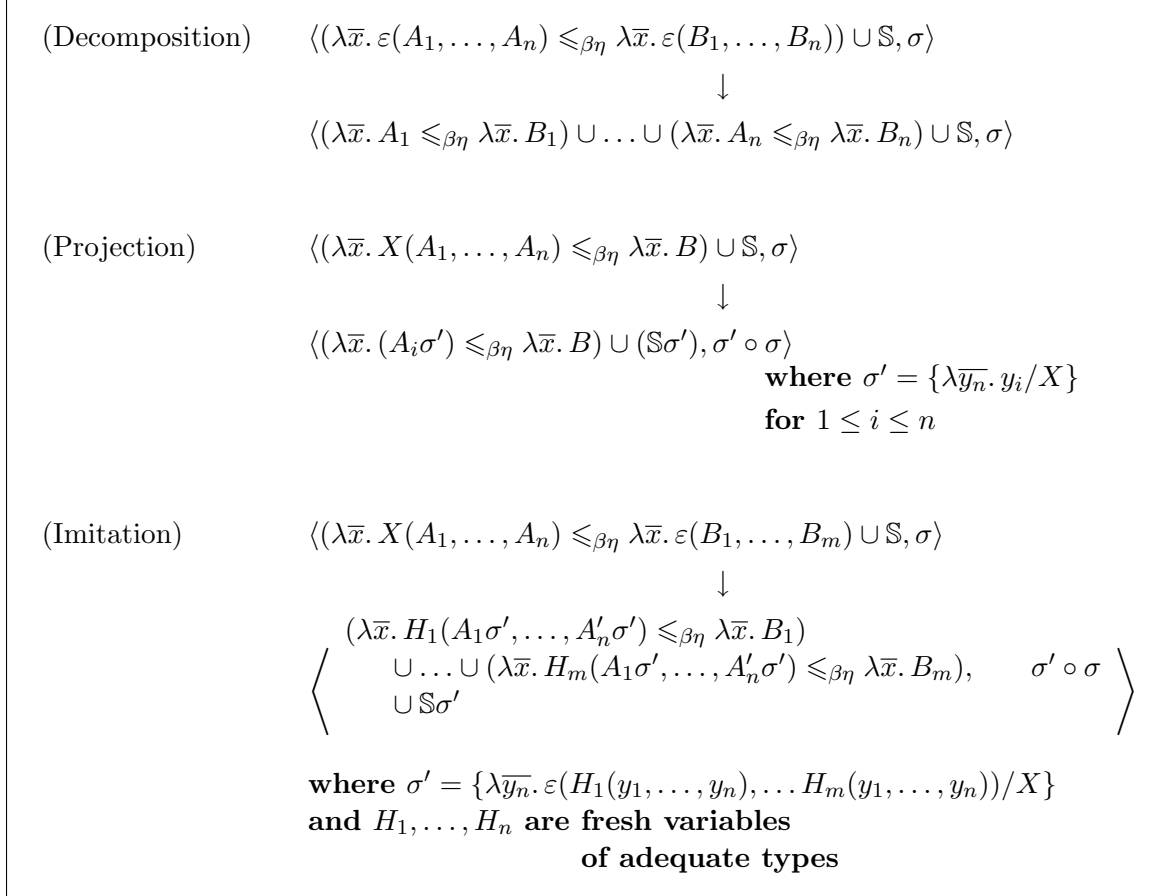


Figure 3: Transformation rules for Huet and Lang second-order matching algorithm

638 **6.1. Second-order Huet and Lang matching algorithm.** The second-order match-
639 ing algorithm presented in [HL78] is a refinement of the unification algorithm proposed
640 in [Hue75] and presented by transformation rules in [SG89]. We adapt this presentation to
641 the second-order case as it was done in [CQS96]. We must recall that for this algorithm we
642 only consider term in β -long normal form. The application of substitutions is normalising.

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

652 equal to the right hand side head symbol; we introduce new matching variables to postpone
653 the remaining choices².

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

Example 6.1. We consider the equation $(\lambda x. X(x, a) \leq_{\beta\eta} \lambda x. f(a, x, a))$ 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

$$\begin{aligned} & \langle (\lambda x. H_1(x, a) \leq_{\beta\eta} \lambda x. a) \\ & \cup (\lambda x. H_2(x, a) \leq_{\beta\eta} \lambda x. x) \quad , \quad \{ \lambda y_1 y_2. f(H(y_1, y_2), H_2(y_1, y_2), H_3(y_1, y_2))/X \} \\ & \cup (\lambda x. H_3(x, a) \leq_{\beta\eta} \lambda x. a) \end{aligned}$$

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

$$\begin{aligned} & \{ \lambda z_1 z_2. a/H_1 \} \quad \text{or} \quad \{ \lambda z_1 z_2. z_2/H_1 \} \\ & \{ \lambda z_1 z_2. z_1/H_2 \} \\ & \{ \lambda z_1 z_2. a/H_3 \} \quad \text{or} \quad \{ \lambda z_1 z_2. z_2/H_3 \} \end{aligned}$$

The initial problem has thus four solutions given by

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

658 6.2. Second-order matching algorithm based on superdevelopments.

659 6.2.1. *Matching modulo superdevelopments and types.* We say that an (untyped) term is
660 typable when it belongs to typed terms. A system is typable when for each equation the
661 two terms are typable and of the same type. Otherwise, we say it is untypable.

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

666 **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
667 then the system \mathbb{S} is also typable.*

668 This proposition is directly obtained from the following lemma

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

²The 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.

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

- 673 • *Rules* (ε_v) and (ε_c) : obvious.
 674 • *Rule* (ε_X) : The systems \mathbb{S} and \mathbb{S}' are defined by $\mathbb{S} = (X \leq_{\beta_{sd}} A) \cup \mathbb{S}_0$ and $\mathbb{S}' =$
 675 $(X \leq_{\beta_{sd}} A) \cup \mathbb{S}_0\{A/X\}$. If the equation $(X \leq_{\beta_{sd}} A)$ is not typable then the result
 676 follows. If not (that is if X and A are typable and of the same type), the system \mathbb{S}_0
 677 is not typable. But since X and A are of the same type, $\mathbb{S}_0\{A/X\}$ is not typable.
 678 • *Rule* (λ_λ) : obvious.
 679 • *Rule* $(@_\beta)$: If $\mathbb{S} = (AB \leq_{\beta_{sd}} C) \cup \mathbb{S}_0$ and \mathbb{S}_0 is not typable then the result follows.
 680 Otherwise, we suppose that the equation $AB \leq_{\beta_{sd}} C$ is not typable. Then let σ be
 681 the type of AB and τ the one of C with $\tau \neq \sigma$. There exist v_0 and v_1 possibly equal
 682 and such that the term A is of type $v_0 \rightarrow \sigma$ and the term $\lambda x.C$ is of type $v_1 \rightarrow \tau$.
 683 The equation $A \leq_{\beta_{sd}} \lambda x.C$ is not typable. This concludes the case.
 684 • *Rule* $(@_\pi)$ and $(@_@)$: Similar to the previous case.

685 □

686 Nevertheless, the reducts of a typable system are not necessarily typable, as the follow-
 687 ing example shows.

688 **Example 6.4** (Reducts of a typable system). Suppose given the equation $XY \leq_{\beta_{sd}} fa$.
 689 If we suppose that X is of type $s \rightarrow \iota$, that Y is of type s , that f is of type $r \rightarrow \iota$
 690 and a is of type r where ι , r and s are three different basic types then the equation is a
 691 typable system. Nevertheless, by applying the transformation rule $(@_@)$ we get the system
 692 $(X \leq_{\beta_{sd}} f) \cup (Y \leq_{\beta_{sd}} a)$ which is not typable. By the way, this equation has a solution
 693 $\{\lambda x. x/X, fa/Y\}$ which is typable.

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

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

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

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

$$\begin{aligned} X \leq_{\beta_{sd}} f & \quad \cup \quad Y \leq_{\beta_{sd}} a \\ X \leq_{\beta_{sd}} \lambda x. fa & \\ X \leq_{\beta_{sd}} \lambda x. fx & \quad \cup \quad Y \leq_{\beta_{sd}} a \\ X \leq_{\beta_{sd}} \lambda x. xa & \quad \cup \quad Y \leq_{\beta_{sd}} f \\ X \leq_{\beta_{sd}} \lambda x. x & \quad \cup \quad Y \leq_{\beta_{sd}} fa \end{aligned}$$

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

$$\begin{aligned} X \leq_{\beta_{sd}} f & \quad \cup \quad Z \leq_{\beta_{sd}} b \\ X \leq_{\beta_{sd}} \lambda x. fb & \\ X \leq_{\beta_{sd}} \lambda x. fx & \quad \cup \quad Z \leq_{\beta_{sd}} b \\ X \leq_{\beta_{sd}} \lambda x. xa & \quad \cup \quad Z \leq_{\beta_{sd}} f \\ X \leq_{\beta_{sd}} \lambda x. x & \quad \cup \quad Z \leq_{\beta_{sd}} fb \end{aligned}$$

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

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

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

700 **6.2.2. Second-order matching algorithm based on superdevelopments.** The context of this
701 section is the typed λ -calculus. We want to determine β -matches of second-order equations.
702 Note that the η -rule does not have a fundamental impact on the algorithm. This section
703 could have been written for second-order matching modulo $\beta\eta$ (including the paragraph related
704 to the comparison with Huet and Lang algorithm).

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

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

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

715 We illustrate the algorithm on an example.

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

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

724 Each of these choices lead to the solutions

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

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

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

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

730 7. ALGORITHM FOR MATCHING MODULO DEVELOPMENTS (AND ETA)

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

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

738 7.1. Matching modulo developments.

739 **Definition 7.1** (β_a -matching equation/system). A β_a -matching equation or simply a *match-*
 740 *ing equation* is a pair of terms denoted $A \leq_{\beta_a} B$ such that B is normal and \mathcal{V} -closed. A
 741 matching *system* is a multiset (potentially empty) of matching equations.

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

Example 7.3 (Incompleteness for second-order matching). We consider the matching equa-
 tion $(XY)Z \leq_{\beta_a} ab$, following Example 3.3. We also consider the substitutions

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

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

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

749 **7.2. Algorithm for matching modulo developments.** The algorithm for matching
 750 modulo developments mimics the definition of parallel reduction given in Section 1.2.2, in
 751 the same way the algorithm for matching modulo superdevelopments mimics the definition
 752 of strong parallel reduction.

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

758 **Theorem 7.5** (Algorithm for matching modulo developments). *The algorithm for matching*
 759 *modulo developments consisting of the rules $(\varepsilon_v, \varepsilon_c, \varepsilon_x, \lambda_{\lambda}, @_{@})$ given in Figure 1 plus the*
 760 *rules $(@_{\pi})$ and $(@_{\beta})$ given in Figure 4 is sound, complete and terminating. It has the finite*
 761 *complete match set property (there is no most general match).*

762 *Proof.* To prove these results, it is very easy to adapt the proofs given in Section 4. \square

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

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

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

765

8. APPLICATIONS TO HIGHER-ORDER REWRITING

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

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

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

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

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

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

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

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

798

CONCLUSION

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

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

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

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

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

824

ACKNOWLEDGEMENTS

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

827

REFERENCES

828

- 829 [Acz78] Peter Aczel. A general Church-Rosser theorem. Technical report, University of Manchester, July
830 1978. [8](#)
- 831 [Bar84] Henk Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the
832 Foundation of Mathematics. North Holland, 1984. Second edition. [2](#), [3](#), [5](#)
- 833 [Bax77] Lewis Denver Baxter. *The complexity of unification*. Thèse de doctorat, University of Waterloo,
834 1977. [29](#)
- 835 [Bür90] Hans-Jürgen J. Bürckert. Matching — A special case of unification? In *Unification*, pages
836 125–138. Academic Press, 1990. [17](#)
- 837 [CK01] Horatiu Cirstea and Claude Kirchner. The rewriting calculus — Part I and II. *Logic Journal of
838 the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001. [33](#)
- 839 [CQS96] Régis Curien, Zhenyu Qian, and Hui Shi. Efficient second-order matching. In *Rewriting Tech-
840 niques and Applications - RTA'96*, volume 1103 of *Lecture Notes in Computer Science*, 1996.
841 [26](#), [27](#)
- 842 [Des98] Joëlle Despeyroux. Natural semantics: specifications and proofs. Technical report, INRIA,
843 Février 1998. [5](#)
- 844 [DHK00] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher order unification via explicit sub-
845 stitutions. *Information and Computation*, 157(1/2):183–235, 2000. [16](#)
- 846 [dMS01] Oege de Moor and Ganesh Sittampalam. Higher-order matching for program transformation.
847 *Theoretical Computer Science*, 269:218–237, 2001. [2](#), [3](#), [13](#), [33](#)
- 848 [Dow94] Gilles Dowek. Third order matching is decidable. *Annals of Pure and Applied Logic*, 69:135–155,
849 1994. [10](#)
- 850 [Dow01] Gilles Dowek. Higher-order unification and matching. In *Handbook of Automated Reasoning*.
851 Elsevier, 2001. [3](#), [11](#), [14](#)
- 852 [DSP91] Mary Dalrymple, Stuart M. Shieber, and Fernando Pereira. Ellipsis and higher-order unification.
853 *Linguistics and Philosophy*, 14:399–452, 1991. [1](#)
- 854 [Fau06] Germain Faure. Matching modulo superdevelopments application to second-order matching. In
855 *13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning
856 - LPAR'06*, volume 4246 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 2006. [3](#)
- 857 [Fau07] Germain Faure. *Structure et modèles de calcul de réécriture*. Thèse de doctorat, Université Henri
858 Poincaré Nancy, 2007. [3](#)
- 859 [Gol81] Donald Goldfarb. The undecidability of the second order unification. *Journal of Theoretical
860 Computer Science*, 13:225–230, 1981. [11](#), [33](#)
- 861 [HL78] Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with
862 second-order patterns. *Acta Informatica*, 11, 1978. [1](#), [2](#), [10](#), [17](#), [26](#), [27](#)
- 863 [Hue73] Gérard Huet. A mechanization of type theory. In Nils J. Nilsson, editor, *Proceedings of the 3rd
864 International Joint Conference on Artificial Intelligence*, pages 139–146, Standford, CA, 1973.
865 [11](#)
- 866 [Hue75] Gérard Huet. *Résolution d'équations dans les langages d'ordre 1, 2, ..., ω* . Thèse de doctorat
867 d'état, Université Paris, 7, 1975. [2](#), [11](#), [27](#)
- 868 [JK91] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-
869 based survey of unification. In *Computational Logic - Essays in Honor of Alan Robinson*, pages
870 257–321, 1991. [15](#)
- 871 [JK09] C. Barry Jay and Delia Kesner. First-class patterns. *Journal Functional Programming*,
872 19(2):191–225, 2009. [33](#)
- 873 [Kir84] Claude Kirchner. A new equational unification method: A generalization of martelli-montanari's
874 algorithm. pages 224–247, 1984. [15](#)
- 875 [Klo80] Jan Willem Klop. *Combinatory Reduction Systems*. Ph.D. thesis, Mathematisch Centrum, Am-
876 sterdam, 1980. [2](#), [4](#), [32](#)
- 877 [KvOvR93] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction
878 systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993. [2](#), [4](#), [32](#)
- 879 [Lév78] Jean-Jacques Lévy. *Reductions Correctes et Optimales dans le Lambda-Calcul*. PhD thesis, Uni-
880 versité de Paris, 1978. [6](#)
- 881 [Loa03] Ralph Loader. Higher order beta matching is undecidable. *Logic Journal of the IGPL*, 11(1):51–
882 68, 2003. [2](#), [10](#), [14](#)

- 883 [Mil90] Dale Miller. Higher-order logic programming. In *Int. Conf. on Logic Programming*, page 784,
884 1990. [1](#)
- 885 [Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and
886 simple unification. *Journal of Logic and Computation*, 1991. [2](#), [14](#)
- 887 [MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on*
888 *Programming Languages and Systems*, 4(2):258–282, 1982. [15](#)
- 889 [MN98] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical*
890 *Computer Science*, 192, 1998. [1](#), [2](#), [32](#)
- 891 [NP98] Tobias Nipkow and Christian Prehofer. Higher-order rewriting and equational reasoning. In
892 *Automated Deduction: A Basis for Applications*. Kluwer, 1998. [1](#)
- 893 [Oos94] Vincent Van Oostrom. *Confluence for abstract and higher-order rewriting*. PhD thesis, Vrije
894 Universiteit, 1994. [2](#), [32](#)
- 895 [Pad00] Vincent Padovani. Decidability of fourth-order matching. *Mathematical Structures in Computer*
896 *Science*, 3(10):361–372, June 2000. [10](#)
- 897 [Pfe01] Frank Pfenning. Logical frameworks. In *Handbook of Automated Reasoning*, volume II, chap-
898 ter 17, pages 1063–1147. Elsevier Science, 2001. [1](#)
- 899 [Qia96] Zhenyu Qian. Unification of higher-order patterns in linear time and space. *J. Log. Comput*,
900 1996. [2](#), [11](#), [14](#)
- 901 [SG89] Wayne Snyder and Jean Gallier. Higher-order unification revisited: Complete sets of transfor-
902 mations. *JSCOMP: Journal of Symbolic Computation*, 8, 1989. [11](#), [15](#), [27](#)
- 903 [Shi94] Hui Shi. *Extended matching with applications to program transformation*. PhD thesis, Universität
904 Bremen, 1994. [1](#)
- 905 [Sit01] Ganesh Sittampalam. *Higher-order Matching for Program Transformation*. PhD thesis, Mag-
906 dalen College, 2001. [2](#), [3](#)
- 907 [Sti09] Colin Stirling. Decidability of higher-order matching. *Logical Methods in Computer Science*,
908 5(3), 2009. [2](#), [10](#), [14](#)
- 909 [Ter03] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. M. Bezem, J. W. Klop and
910 R. de Vrijer, eds. [2](#), [5](#), [32](#)
- 911 [Vis05] Eelco Visser. A survey of strategies in rule-based program transformation systems. *Journal of*
912 *Symbolic Computation*, 40(1), 2005. [1](#)
- 913 [vOvR93] Vincent van Oostrom and Femke van Raamsdonk. Comparing combinatory reduction systems
914 and higher-order rewrite systems. volume 816 of *Lecture Notes in Computer Science*, 1993. [1](#),
915 [32](#)
- 916 [vR93] Femke van Raamsdonk. Confluence and superdevelopments. In Claude Kirchner, editor, *5th*
917 *International Conference on Rewriting Techniques and Applications - RTA '93*, volume 690 of
918 *Lecture Notes in Computer Science*, pages 168–182. Springer-Verlag, 1993. [2](#), [3](#), [6](#)
- 919 [vR96] Femke van Raamsdonk. *Confluence and Normalisation for higher-order rewriting*. Thèse de
920 doctorat, Vrije Universiteit, 1996. [2](#), [3](#), [7](#), [32](#)