Positive and negative results for higher-order disunification

Denis Lugiez

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Positive and negative results for higher-order disunification

Denis LUGIEZ

N° 2492
Février 1995

PROGRAMME 2

Positive and Negative Results for Higher-Order Disunification

Denis Lugiez
CRIN-INRIA Lorraine [1]

Abstract: *This paper is devoted to higher-order disunification which is the process of solving quantified formulae built on simply-typed lambda-terms, the equality induced by the η and the β reductions, boolean connectives and the negation. This problem is motivated by tests of the completeness of definitions in algebraic higher-order specification languages which combine the advantages of algebraic specification languages and higher-order programming languages. We show that higher-order disunification is not semi-decidable and we prove the undecidability of second-order complement problems which are the formulae expressing the completeness of some scheme, by encoding Minsky machines. On the other hand, we show that second-order complement problems are decidable if second-order variables and bound variables satisfy some (reasonable) conditions and that the validity of any quantified equational formula can be checked when all the terms occurring in this formula are patterns i.e s.t. the arguments of free variables are distinct bound variables. Both cases are decided using quantifier elimination techniques*

Résultats positifs et négatifs sur la disunification d'ordre supérieur

Résumé: *Cet article est consacré à la disunification d'ordre supérieur qui consiste à résoudre des formules quantifiées construites avec des lambda-termes simplement typés, l'égalité induite par la éta et la béta réduction, les connecteurs booléens et la négation. Cette étude est motivée par la question de complètude des définitions fonctionnelles dans les langages algébriques d'ordre supérieur, langages qui combinent les avantages des langages de spécification algébrique et des langages de programmation d'ordre supérieur. Nous prouvons que la disunification d'ordre supérieur n'est pas semi-décidable et nous montrons également l'indécidabilité des problèmes de compléments d'ordre 2 qui sont les formules exprimant la complétude d'une définition par cas (à l'ordre 2). La preuve se fait par codage d'une machine de Minsky. D'un autre coté nous montrons que les problèmes de compléments d'ordre 2 sont décidables si les variables d'ordre 2 et les variables liées satisfont des conditions raisonnables, et nous démontrons aussi la décidabilité de la validité d'une formule quelconque quand tous les lambda-termes de cette formules sont des patterns, c.a.d. que les arguments d'une variable libre sont des variables liées distinctes. Ces résultats sont obtenus en utilisant des méthodes d'élimination de quantificateurs.*

[1] 615 rue du Jardin Botanique BP 101 54602 Villers les Nancy Cedex FRANCE. e-mail:lugiez@loria.fr

Introduction

The formal verification of systems and programs is a major challenge of Computer Science and a lot of work has been already devoted to this question. A key part of this process is the specification step since it allows to describe the behaviour of a program and to prove properties of this program independently of its implementation. This requires that specification languages are available and that these languages have some good properties. Firstly a specification language should be simple, expressive and closely related to what it models. Secondly, it must provide facilities for verification purposes. For example, it is recommended that a specification language is executable and that some basic proofs can be done easily. Recently, there have been several propositions of *higher-order specification languages* which are intended to combine the advantages of two existing paradigms, i.e algebraic specification and higher-order programming. These languages allow definitions like:

$$
\begin{aligned}
&0 + x = x \\
&s(x) + y = s(x + y) \\
&0 * x = 0 \\
&s(x) * y = x * y + y \\
&@(\lambda x.F) = 0 \\
&@(\lambda x.(F(x) + G(x))) = \lambda x.(@(F)(x) + @(G)(x)) \\
&@(\lambda x.(F(x) * G(x))) = \lambda x.(@(F)(x) * G(x) + F(x) * @(G)(x))
\end{aligned}
$$

which define basic first-order objects i.e the natural numbers together with addition and multiplication, involving first-order variables only, and functional higher-order objects i.e @ the derivative of polynomial functions, involving second-order function variables. On this example, one can see that these languages inherit the simplicity of equational algebraic specification and rewrite systems and the expressivity of lambda-calculus for higher-order functions. The reduction relation associated to these languages have been studied by several authors [BT88, JO91, Nip91, Wol91] who have found that it has the good properties required for defining the operational semantics of such languages.

However much less work has been devoted to the proof aspects which are required for verification purposes. Previous works in this direction [ACS94, Pre94a, NQ91, Mil92] deal with equational proofs and relevant methods like narrowing and unification, but nothing has been done on the fundamental issue (from a specification point of view) of the completeness of definitions in this framework. This step consists in checking that a case definition à la ML, like above, handles all possibilities and that there is no missing case. This is a run of the mill check in functional language like ML, but the problem is much more complicated for higher-order algebraic languages since the definitions may contain explicitely lambda-terms when ML pattern definitions contains first-order terms only. Therefore completeness tests amount to solving quantified expressions on lambda-terms and the equality induced by the *eta* and *beta* reductions, denoted by $=_{\eta\beta}$ in the following. Solving quantified formulae on simply-typed lambda-terms is called *higher-order disunification* and the formulae related to completeness of definitions fall in the subclass of *complement problems*. Since higher-order disunification contains higher-order unification, our goal of checking automatically the completeness of definitions seems to have little chance of success. Indeed, we prove in this paper that higher-order disunification, contrary to first-order disunification [CL89], is even not semi-decidable

and that second-order complement are undecidable (by encoding Minsky machines). However, we are able to prove the decidability of second-order complement problems when some conditions are set on second-order variables and bound variables, but not on first-order variables. Moreover, many definitions encountered in practice involve only patterns, i.e. lambda-terms s.t. the arguments of a free variable are distinct bound variables, and we are able to give an algorithm which solves any formula built on $=_{\eta\beta}$ when all terms occurring in the formula are patterns. For example, the above definition of the derivative @ is both a pattern definition and a second-order linear definition, therefore its completeness can be tested, yielding the answer that several cases are missing, like the definition of $@(\lambda x.x)$, but also $@(\lambda x.s(F(x)))$ when F depends of its argument.

This paper is devoted to the study of higher-order disunification and gives both decidability and undecidability results, but we shall not discuss any semantics issue related to higher-order languages, see [BT88, JO91] for this topic. Section 1 contains the basics required to read it and the undecidability results are described in section 2. Then the two last sections (4 and 5) describe our decidability results concerning second-order complement problems and patterns.

1 Definitions and notations

1.1 Typed lambda-terms

Our framework is that of simply typed lambda-calculus and the reader is assumed to be familiar with the usual notions of this calculus, for a comprehensive survey see [HS86] for example.

Types are constructed from a set of base types and the type constructor $\rightarrow$ which is right associative i.e. $\alpha \rightarrow \beta \rightarrow \gamma$ stands for $\alpha \rightarrow (\beta \rightarrow \gamma)$, and the notation $\alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$ denotes $\alpha_1 \rightarrow \ldots \alpha_n \rightarrow \beta$. The order $ord(\alpha)$ of a type is defined by: $\begin{cases} ord(\alpha) = 1 \; if \; \alpha \; is \; a \; base \; type \\ ord(\alpha \rightarrow \beta) = Max(1 + ord(\alpha), ord(\beta)) \end{cases}$

For each type α, we suppose that there is a denumerable set of variables V_α of type α and a *finite* set of constants C_α of type α. Moreover we shall assume that the signature $\mathcal{C} = \cup_\alpha C_\alpha$ is finite, i.e. there is only a finite number of constants. A lambda-term is an atom i.e a variable or a constant, or an abstraction $\lambda x.t$ where x is a variable of type σ_1, t a term of type σ_2 and the abstraction has type $\sigma_1 \rightarrow \sigma_2$, or else an application $(s\ t)$ where s is a term of type $\sigma_1 \rightarrow \sigma_2$, t is a term of type σ_1 and the application has type σ_2.

The order of a term of type α is the order of its type, $ord(\alpha)$. The term $(\ldots(a\ t_1)\ldots t_n)$ where a is an atom will be written $a(t_1, \ldots, t_n)$ or $a(\bar{t}_n)$. An abstraction $\lambda x_1 \ldots x_n.t$ where t is not an abstraction is written $\lambda \bar{x}_n.t$ with the convention that if $n = 0$ then $\lambda \bar{x}_n.t$ stands for t. The *root* of a term $\lambda \bar{x}_n.t$ is t if t is a variable or a constant, otherwise it is a if $t = a(t_1, \ldots, t_m)$. A language is of order n if all constants have order at most $n+1$ and variables have order at most n.

A term is *flexible* if its root is a free variable, it is *rigid* otherwise. The set of *free variables* of a term t is denoted by $FV(t)$ and a lambda-term is *closed* if it does not contain any free variable.

Free variables are denoted with capital letters $F, G, X, \ldots$ and bound variables are denoted with lower-case letters $x, y, z, \ldots$ The notation $\bar{x}_n$ (resp. $\bar{X}_n$ or $\bar{u}_n$) denotes $x_1, \ldots, x_n$ (resp. $X_1, \ldots, X_n$ or $u_1, \ldots, u_n$). For simplicity, we may drop the index n when it is useless.

A position is a sequence of integers and we denote the empty sequence by ϵ. A variable F occurs at position ϵ in a term t if $t = \lambda\bar{x}_m.F(\bar{u}_n)$, it occurs at position $i.p$ if $t = \lambda\bar{x}_m.a(u_1, \ldots, u_p)$ and F occurs at position p in u_i.

The height $h(t)$ of a term t is defined by $\begin{cases} h(\lambda\bar{x}_n.c) = 1 \\ h(\lambda\bar{x}_n.a(t_1, \ldots, t_n)) = 1 + Max_{i=1,\ldots,n}(h(t_i)) \end{cases}$

A *context*, denoted by $C[\,]$ is a closed term built on $\mathcal{C} \cup \{[\,]\}$ where $[\,]$ is a new constant of arity 0. A context $C[\,]$ may contain one or several occurrences of $[\,]$. The term obtained by substituting the occurrences of $[\,]$ in $C[\,]$ by some term t is denoted by $C[t]$. If the context $C[\,]$ has m occurrences of $[\,]$, ordered from left to right, $C[t_1, \ldots, t_m]$ denotes the replacement of the first one by t_1, the second one by t_2, ... For example, $\lambda xy.a(a([\,], x), [\,])$ is a context $C[\,]$ and $C[b, F]$ denotes $\lambda xy.a(a(b, x), F)$.

The reduction relation is the usual *eta-beta* reduction and the related equality is denoted by $=_{\eta\beta}$. A classical result states that each term has a long $\eta\beta$-normal form which is unique up to renaming (α-conversion). Throughout the paper, *terms, including contexts, are supposed to be in long $\eta\beta$-normal form.*

A type σ is *finitary* if there is only a finite number of closed normal forms of this type (up to renaming) otherwise it is infinitary. We shall consider variables of infinitary types [1] only (in the other case perform a case analysis on the possible closed instances). A substitution σ is a set $\{X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n\}$ where for each i, X_i is a variable and t_i is a term of the same type different from X_i. A substitution is a *closed substitution* if all the t_i 's are closed terms. The domain of σ is $\{X_1, \ldots, X_n\}$. The identity substitution *id* is the substitution such that $Dom(id) = \emptyset$. To apply a substitution σ to a term t is defined as usual (it may need some renaming of bound variables) and is denoted $t\sigma$.

1.2 Equational problems

We start with special kinds of formulae which are the most likely to occur, called *equational problems.*

Definition 1 *An equational problem is some expression $\exists\bar{W}_n\ \forall\bar{Y}_m : \mathcal{P}$ with $\bar{W}_n \cap \bar{Y}_m = \emptyset$, $n, m \geq 0$ and where $\mathcal{P}$ is:*

- *either $\top$ (for* true*) or $\bot$ (for* false*),*

[1] finitary types can occur only in very special cases like $\cup C_\alpha = \emptyset$

- *or an equation* $s =_{\eta\beta} t$ *or a disequation* $s \neq_{\eta\beta} t$,

- *or a conjunction of disjunctions of equations, disequations,* $\top$ *or* $\bot$.

The free variables of an equational problem are the variables which are neither bound nor quantified. The right-hand side of an equation $s =_{\eta\beta} t$ *or a disequation* $s \neq_{\eta\beta} t$, *is* t, *the left-hand side is* s.

Example $\forall Z_1, Z_2, Z_3, L:\ map(\lambda x.X(x), Y) \neq_{\eta\beta} map(\lambda z.Z_1(z), nil) \wedge$
$map(\lambda x.X(x), Y) \neq_{\eta\beta} map(\lambda z.Z_2(z), cons(Z_3, L))$
is an equational problem (with no existential variables, i.e $n = 0$) related to the question of the sufficient completeness of the definition of the higher-order function *map*. The free variables of this problem are X and Y. ■

An equational problem usually has free variables, and we are interested in solutions of the equational problem, i.e assignments to these free variables such that the resulting formula is equivalent to *true*. This is formally defined in what follows:

Definition 2 *A solution* σ *of an equational problem* $\mathcal{E}$ *is a closed substitution such that* $Dom(\sigma) = FV(\mathcal{E})$ *which satisfies the requirements:*

- *if* $\mathcal{E}$ *is* $\top$ *then there is no requirement,*

- *if* $\mathcal{E}$ *is* $\bot$ *then no substitution can be a solution,*

- *if* $\mathcal{E}$ *is* $s =_{\eta\beta} t$ *then* $s\sigma$ *and* $t\sigma$ *are equal modulo* $\eta\beta$*-reduction,*

- *if* $\mathcal{E}$ *is* $s \neq_{\eta\beta} t$ *then* $s\sigma$ *and* $t\sigma$ *are not equal modulo* $\eta\beta$*-reduction,*

- *if* $\mathcal{E}$ *is a disjunction* $d_1 \vee \ldots \vee d_n$ *then there is some* d_i *such that the restriction of* σ *to* $FV(d_i)$ *is a solution of* d_i,

- *if* $\mathcal{E}$ *is a conjunction* $c_1 \wedge \ldots \wedge c_n$ *then, for each* c_i, *the restriction of* σ *to* $FV(c_i)$ *is a solution of* c_i,

- *if* $\mathcal{E}$ *is* $\exists \bar{W}_n\ \forall \bar{Y}_m : P$ *then there exists a closed substitution* $\theta = \{W_1 \leftarrow s_1, \ldots, W_n \leftarrow s_n\}$ *such that for all ground substitutions* $\rho = \{Y_1 \leftarrow t_1, \ldots, Y_m \leftarrow t_m\}$, *the substitution* σ *is a solution of* $P\theta\rho$.

Example $Y \leftarrow nil$ is a solution of $\exists X\ \forall Z, X', L : map(\lambda x.X(x), Y) \neq_{\eta\beta} map(\lambda z.Z(z), cons(X', L))$.
■

This notion of solution is extended to any quantified formula built on $=_{\eta\beta}, \neq_{\eta\beta}, \wedge, \vee$ in a straightforward way. Our purpose is to decide the existence and to compute the solutions of equational problem if any.

Remark 1 *The introduction of both free and existential variables is for technical reasons. In some applications, we are interested in a yes or no answer to some question, therefore free variable are useless but in other applications, we may be interested in the values of the original variables such that the formula is true, therefore free variables are required.*

In higher-order specification languages, case definition looks like

$$map(\lambda x.F(x), nil) \rightarrow nil$$
$$map(\lambda x.F(x), cons(X, L)) \rightarrow cons(F(X), map(\lambda x.F(x), L))$$

and the completeness of this definition is checked by solving the following equational problem:

$$\exists X, Y\ \forall F, Z, L : \begin{array}{l} map(\lambda x.X(x), Y) \neq_{\eta\beta} map(\lambda x.F(x), nil) \wedge \\ map(\lambda x.X(x), Y) \neq_{\eta\beta} map(\lambda x.F(x), cons(Z, L)) \end{array}$$

This problem belongs to the class of complement problems, i.e. formulae of the form

$$\exists \bar{X} \forall \bar{Y} : t \neq_{\eta\beta} t_1 \wedge \ldots \wedge t \neq_{\eta\beta} t_n$$

with $\bar{X} = FV(t)$ and $\bar{Y} = FV(t_1) \cup \ldots \cup FV(t_n)$ which are relevant in this completeness of definition issue and in other applications in computer science. This explains why they are extensively discussed in the following.

2 Undecidability results

2.1 Higher-order disunification is not semi-decidable

In this section we prove that higher-order disunification is not semi-decidable even when it is restricted to second-order terms. Let s, t be two second-order terms then the problem

$$\exists \bar{X} : s =_{\eta\beta} t$$

where $\bar{X} = Var(t) \cup Var(s)$ is an equational problem, as well as the problem

$$\forall \bar{X} : s \neq_{\eta\beta} t$$

where $\bar{X} = Var(t) \cup Var(s)$ and the second one is the negation of the first one.

If solving equational problem were semi-decidable, one could run the procedure on both problems in parallel, and one of them will stop with success. Therefore second order unification would be decidable which is not the case [Gol81]. Looking at this disappointing result, one could ask whether some simpler problems are decidable, and a good candidate is the second-order complement problem, which would have useful practical applications. Unfortunately, this kind of formulae is still too general, as proven by our next result.

2.2 Second-order complement problem is undecidable

2.2.1 Two-counter automata

We show that second-order complement problem is undecidable by encoding two-counter finite state machines. A non-deterministic two-counter automaton without input is a tuple (Q, q_0, Q_F, a, δ), where

- Q is a finite set of states,
- q_0 is the initial state,
- Q_F is the set of final states,
- a is the single element of the stack alphabet,
- $\delta \subseteq Q \times \{0,1\}^2 \times Q \times \{-1,0,1\}^2$ is the transition relation of the automaton.

The notation $\wedge$ is for the empty word of the stack alphabet and $|\alpha|$ denotes the length of the word α. A configuration of the machine is some triple (C_1, q, C_2) where $C_1, C_2 \in a^*$ are the values of the counters and q is the current state. The relation $\Rightarrow$ on the set of configurations is defined by $(C_1, q, C_2) \Rightarrow (C_1', q', C_2')$, iff there is some $(q, x_1, x_2, q', y_1, y_2) \in \delta$ s.t.:

- if $|C_i| = 0$ then $x_i = 0$ else $x_i = 1$,
- $|C_i'| = |C_i| + y_i$

Moreover we assume that if $x_i = 0$ then $y_i \geq 0$, i.e. the content of a counter cannot be negative. A computation is a sequence of configuration $Conf_1\ Conf_2 \ldots\ Conf_m$ s.t. $Conf_1 = (\wedge, q_0, \wedge)$, and $Conf_i \Rightarrow Conf_{i+1}$. It is accepted if $Conf_m = (C_1, q_f, C_2)$ with $q_f \in Q_F$. It is well know that it is undecidable whether a two-counter automaton has an accepting computation or not.

2.2.2 Encoding computations of two-counter automata

We show how to encode this problem as a second-order complement problem. The set of types is $\{\mathcal{N}at, Conf, State\}$ and the set of constants is:

$$0 :\rightarrow \mathcal{N}at$$
$$s : \mathcal{N}at \rightarrow \mathcal{N}at$$
$$\# :\rightarrow Conf$$
$$q :\rightarrow State \quad for \quad each \quad q \in Q$$
$$h : \mathcal{N}at \times State \times \mathcal{N}at \times Conf \rightarrow Conf$$

From this signature, one realizes immediately that a term of type $\mathcal{N}at$ is either 0 or some $s^n(0)$ which models perfectly the content of a counter (0 for $\wedge$ and $s^n(0)$ for a^n), that a term of type $Conf$ is either # or some $h(C_1^1, q^1, C_2^1, h(C_1^2, q^2, C_2^2, h(\ldots, h(C_1^m, q^m, C_2^m, \#))))$ which models a sequence of configuration $Conf_1\ Conf_2\ \ldots\ Conf_m$ with $Conf_i = (C_1^i, q^i, C_2^i)$.

We aim at proving that a two-counter automaton $\mathcal{M}$ has an accepting computation iff some complement problem has a solution. To achieve this, we define a set of terms $t_1, \ldots, t_n$ and a term t s.t. each ground instance of t is some $t_i\theta$ iff $\mathcal{M}$ has no accepting computation. Therefore the set of t_i is intended to reduce each sequence of configurations which is not an accepted computation. First we give a simple-minded solution and show why it does not work. This will explain why the actual encoding is somewhat contorted and where the difficulty lies.

A straightforward encoding is to take t as some variable of sort $Conf$ and the t_i as terms of the form $Y(\textit{some bad move})$ where Y is a linear second-order variable but we show in section 4 that such problems are decidable. Where is the trap? It lies in the mere fact that the terms $Y(\textit{some bad move})$ are intended to represent bad sequence only, but they also encode good sequences: let $Comp$ be the encoding of an accepted computation of $\mathcal{M}$, then choosing $Y = \lambda z.Comp$ returns the value $Comp$ as an instance of the above term. Therefore one cannot discriminate between good and bad sequences of configurations. It is interesting to realize that the existence of lambda terms representing functions which do not depend on some of their arguments is what invalidates this encoding and makes the decidability proof for linear complement work.

Now, we give the actual encoding for bad sequences. Since one has to consider pairs of configurations, one adds a new type $\mathcal{P}air$ and a new pairing operator $[\_,\_] : Conf \times Conf \rightarrow \mathcal{P}air$. The pairing of two configurations is denoted by $[Conf_1, Conf_2]$. The term t is chosen as:

$$t = [C(\#), C(h(0, q_0, 0, \#))]$$

where C is a second order variable of type $Conf \rightarrow Conf$. The argument of C in the second argument does not really matter, provided that the two components of the pair have different values if C is not a constant. Our goal is to reduce all instances of t s.t. the first component of C is a not an accepted computation. We perform a case analysis on how these instances can be reduced and for each case, we give the suitable t_i('s). For simplicity, the notation _ stands for some anonymous linear variable (à la ML).

- First, we want to get rid of all instances of t coming from the instantiation of C by some constant. This is achieved by setting $t_1 = [U, U]$ with U a first-order variable of type $Conf$. From now on, we consider only the instances of C by a non-constant function. In the following, we focus on the first component of pairs (and point out when the second component is

relevant).

- We reduce the sequences such that the first configuration is bad:

  - the first counter is not 0 at starting point:

$$t_2^1 = [h(s(x), \_, \_, \_), \_]$$

  - the second counter is not 0 at starting point:

$$t_2^2 = [h(\_, \_, s(x), \_), \_]$$

  - The state is not the initial state at starting point:

$$t_2^2 = [h(\_, q, \_, \_), \_]$$

    for each $q \neq q_0$:

In the following, the terms t_i match the template $[Y(\_), Y(\_)]$ where only instances of Y by non-constant functions are relevant since instantiations by constant functions correspond to instances of t of the form $[U, U]$ which have been already reduced.

- We reduce sequences containing an increment of the first counter by 2 or more.

$$t_3 = [Y(h(Z(0), \_, \_, h(s(s(Z(Z'(0))))), \_, \_, \_))), Y(\_)]$$

and the same for the second counter.

- We reduce sequences containing an decrement of the first counter by 2 or more.

$$t_4 = [Y(h(s(s(Z(Z'(0))))), \_, \_, h(Z(0), \_, \_, \_))), Y(\_)]$$

and the same for the second counter.

- We reduce the sequences which do not end with a final state:

$$t_5 = [Y(h(\_, q, \_, \#)), Y(\_)]$$

for each $q \notin Q_F$

- We reduce the sequences where the increment or decrement is 0 or 1 but where some move $Conf_i \Rightarrow Conf_{i+1}$ is not allowed by δ. At this point, irreducible instances contain only moves which change the absolute value of the counters by at most one, and change state q to state q'. We show how to eliminate the remaining illegal sequences: assume that in some step, *each counter is not empty, the first counter is incremented by* 1, *the second counter is decremented by* 1, *the states changes from* q *to* q', but this is not a legal step.

  For each 5-tuple s.t. $(q, 1, 1, q', 1, -1) \notin \delta$ add the term:

  $$t_6 \doteq [Y(h(s(Z(0)), q, s(Z'(0)), h(s(s((Z(0))), q', Z'(0), \_), Y(\_)]$$

  By looking at all possibilities whether the counters are empty or not and whether the counters are decremented or incremented, one complete the set of terms t_i using similar terms.

What remains to do is to prove our claim: any instance of t is reduced iff there is no accepted computation.

2.2.3 The complement problem has a solution iff there is an accepted computation

The key point is to realize that one can associate a unique instance of t to any (encoding of) sequence of configurations if only non-constant functions are allowed. A sequence of configurations is some term $h(\_, \_, \_, h(\ldots, h(\_, \_, \_, \#)))$ where $\#$ occurs once. Therefore there is only one non-constant solution C s.t. $C(\#) = h(\_, \_, \_, h(\ldots, h(\_, \_, \_, \#)))$, i.e $C = \lambda z. = h(\_, \_, \_, h(\ldots, h(\_, \_, \_, z)))$. Moreover, one immediately realizes that the second component of the instance of t is the initial sequence where $\#$ is replaced by $h(0, q_0, 0, \#)$ and that this component is not equal to the first one.

We prove the first part of the equivalence: *if there is no accepted computation then there is no solution to the complement problem or accordingly each instance of* t *is reduced.*

- the instances of t using constant functions C are reduced by $[U, U]$.

- the other instances of t are $[\ldots, \ldots]$ where the first component matches the first component of some t_i. Since the variables Y occurring in the t_i's cannot be instantiated by constants (otherwise C should be constant), it is obvious to find a match for the second argument from a match for the first one, which proves that the instance of t is reducible.

Now, we prove the converse: *if there is no solution to the complement problem then there is no accepted computation.* If there were some accepted computation, there the corresponding instance (by a non-constant C) could not be reduced by any t_i (the variables Y cannot be constants), which means that there would be a solution.

These results are summed up by the following statement:

Theorem 2 *The second-order complement problem is undecidable*

An interesting remark is that our encoding does not contain any bound variable, therefore the undecidability results holds also for terms constructed from first-order term and second-order context variables, which is a stronger result.

3 A set of general computation rules

The undecidability results do not prevent us from trying to solve equational problems. In the next section, we give a set of rules which are useful to simplify equational problems in order to get simpler ones. Moreover we take for granted the classical logical laws such as commutativity, associativity, distributivity of boolean connectives, suppression of useless quantification and the normalization of each term in its long eta-beta normal form. We usually drop irrelevant contexts, for example we would write $t =_{\eta\beta} t \rightarrow \top$ instead of $\exists \bar{W}_l \forall \bar{Y}_n : (t =_{\eta\beta} t \vee P) \wedge Q \rightarrow \exists \bar{W}_l \forall \bar{Y}_n : Q$.

3.1 Basic rules

First, one has a set of basic rules containing rules of higher-order unification and their dual through negation.

Type checking rules

$(TC1)$	$t =_{\eta\beta} s \rightarrow \bot$	$(TC2)$	$t \neq_{\eta\beta} s \rightarrow \top$

if the type of t is not the type of s.

Elimination of trivial equations rules

$(ET1)$	$t =_{\eta\beta} t \rightarrow \top$	$(ET2)$	$t \neq_{\eta\beta} t \rightarrow \bot$

Decomposition rules

$(D1)$	$\lambda\bar{x}_n.a(t_1,\ldots,t_n) =_{\eta\beta} \lambda\bar{x}_n.a(s_1,\ldots,s_m) \rightarrow$	$\lambda\bar{x}_n.s_1 =_{\eta\beta} \lambda\bar{x}_n.t_1 \wedge \ldots \wedge \lambda\bar{x}_n.s_n =_{\eta\beta} \lambda\bar{y}.t_n$
$(D2)$	$\lambda\bar{x}_n.a(t_1,\ldots,t_n) \neq_{\eta\beta} \lambda\bar{x}_n.a(s_1,\ldots,s_m) \rightarrow$	$\lambda\bar{x}_n.s_1 \neq_{\eta\beta} \lambda\bar{x}_n.t_1 \vee \ldots \vee \lambda\bar{x}_n.s_n \neq_{\eta\beta} \lambda\bar{y}.t_n$

if a is a constant or belongs to $\bar{x}_n$.

Clash rules

(C1) $\lambda\bar{x}_n.a(\bar{t}_n) =_{\eta\beta} \lambda\bar{x}_n.b(\bar{s}_m) \rightarrow \bot$
(C2) $\lambda\bar{x}_n.a(\bar{t}_n) \neq_{\eta\beta} \lambda\bar{x}_n.b(\bar{s}_m) \rightarrow \top$
if a and b are distinct constants or variables of $\bar{x}_n$.

3.2 The explosion rule

(EX) $\exists\bar{W}_l\ \forall\bar{Y}_n : P \rightarrow \exists\bar{W}_l\ \exists\bar{H}_p\ \forall\bar{Y}_n :\ F =_{\eta\beta} \lambda\bar{x}_m.a(H_1(\bar{x}_n),\ldots,H_k(\bar{x}_n)) \wedge P\{F \leftarrow \lambda\bar{x}_m.a(H_1(\bar{x}_n),\ldots,H_k(\bar{x}_n))\}$

if F is a free or existential variable of type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$, x_i has type τ_i, $a(H_1(\bar{x}_n),\ldots,H_k(\bar{x}_n))$ has type τ and $a \in \mathcal{C} \cup \bar{x}_m$. For simplicity we write $H_i^j(\bar{x}_n)$ instead of its long eta-beta normal form.

This rule contains the classical imitation and projection rules. Given some explosion variable F, any possible a in $\cup_\alpha C_\alpha \cup_{i=1,\ldots,m} \{x_i\}$ must be considered for completeness sake. This means that the procedure builds a finitely branching tree (see the map example in section 3.5).

3.3 The explosion rule for universal variables

A case analysis can be performed on universal variables too. This rule is useful in some cases (see section 5) but usually causes non-termination since one universal variable is replaced by many ones and a formula is replaced by a conjunction of formulae.

$(EXUV)$ $\forall Y : P \rightarrow \forall\bar{H}_p^l\ \bigwedge_{a_i \in \mathcal{C}\cup\bar{x}_n} P\{Y \leftarrow \lambda\bar{x}_n.a_i(H_i^1(\bar{x}_n),\ldots,H_i^{m_i}(\bar{x}_n))\}$

where the type of Y is $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$, the type of x_i is τ_i and the type of each $a_i(H_i^1(\bar{x}_n),\ldots,H_i^{m_i}(\bar{x}_n))$ is τ. For simplicity we write $H_i^j(\bar{x}_n)$ instead of its long eta-beta normal form.

3.4 Correctness and consistency of the rules

A rule is said to be *consistent* if each side has the same set of solutions than the other one, it is said to be *correct* if the set of solution of the right-hand side is included in the set of solutions of the left-hand side. The consistency and correctness of the transformation rules is the first property to check. Before illustrating these rules on some example, we give the following correctness result:

Proposition 1 *Basic rules and the EXUV rule are consistent.*

Proof This is obvious because of the semantics of equational problems and of the existence of the long eta-beta normal form. □

The explosion rule is correct but not consistent since the solutions of the right-hand side are solution of the left-hand side but the converse can be false. To overcome this difficulty, one simply makes all possible transformations using all possible choices of a in $\mathcal{C} \cup \bar{x}_n$, building a proof tree in which branching corresponds to the explosion of an existential or free variable.

3.5 An example

We show how one can use these rules to do some simplification on equational problems. The reader will see the limitations of these rules since we shall get stuck at some point because we cannot get rid of universal variables. The limitations will be overcomed and the example continued later on in section 4. The proposed example is the equational problem related to the completeness of the map function, given in section 1.2.

$$\exists F, L \forall Z_1, Z_2, X, L_1 : \begin{array}{l} map(\lambda x.F(x), L) \neq_{\eta\beta} map(\lambda z.Z_1(z), nil) \wedge \\ map(\lambda x.F(x), L) \neq_{\eta\beta} map(\lambda z.Z_2(z), cons(X, L_1)) \end{array}$$

$$\rightarrow_{Decomposition}$$

$$\exists F, L \forall Z_1, Z_2, X, L_1 : \begin{array}{l} (\lambda x.F(x) \neq_{\eta\beta} \lambda x.Z_1(x) \vee L \neq_{\eta\beta} nil) \wedge \\ map(\lambda x.F(x), L) \neq_{\eta\beta} map(\lambda z.Z_2(z), cons(X, L_1)) \end{array}$$

$$\rightarrow_{Decomposition}$$

$$\exists F, L \forall Z_1, Z_2, X, L_1 : \begin{array}{l} (\lambda x.F(x) \neq_{\eta\beta} \lambda x.Z_1(x) \vee L \neq_{\eta\beta} nil) \wedge \\ (\lambda x.F(x) \neq_{\eta\beta} \lambda z.Z_2(z) \vee L \neq_{\eta\beta} cons(X, L_1)) \end{array}$$

At this point the variable L is exploded yielding two branches. The first one corresponds to the explosion $L = nil$ and the second one to the explosion $L = cons(X_1, L_1)$.

The first case $L = nil$ is continued as follows:

$$\exists F, L \forall Z_1, Z_2, X, L_1 : \begin{array}{l} (\lambda x.F(x) \neq_{\eta\beta} \lambda x.Z_1(x) \vee nil \neq_{\eta\beta} nil) \wedge \\ (\lambda x.F(x) \neq_{\eta\beta} \lambda z.Z_2(z) \vee nil \neq_{\eta\beta} cons(X, L_1)) \wedge L = nil \end{array}$$

$$\rightarrow_{Clash}$$

$$\exists F, L \forall Z_1, Z_2, X, L_1 : \lambda x.F(x) \neq_{\eta\beta} \lambda x.Z_1(x) \wedge L = nil$$

Although common sense tells us that the last expression should evaluate to $\perp$, none of the basic rule is applicable and using explosion rules quickly enters an infinite loop, therefore we have to wait for some new rule before achieving our goal. Meanwhile, what has happened on the other branch $L = cons(X', L')$? In fact the same kind of situation shows in and we get stuck with the formulae

$$\exists X', L', F, L \forall Z_1, Z_2, X, L_1 : (\lambda x.F(x) \neq_{\eta\beta} \lambda z.Z_2(z) \vee X' \neq_{\eta\beta} X \vee L \neq_{\eta\beta} L_1) \wedge L = cons(X', L')$$

The rest of the paper is devoted to several approaches for solving this problem. Since there is no hope to get a general solution, we shall describe some particular decidable cases.

4 A decidable case of second-order complement problems

In this section we consider a second-order language, which means that constants have order 3 at most, and variable have order 2 or 1. Therefore a variable has type τ or type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$ where τ and the τ_i's are basic types.

4.1 Restricted top-linear complement problems

In the following, we are interested in second-order complement problems where second-order variables are assumed to be *top-linear*. To define this notion, we need some terminology: a position p is a *top-position* of a variable F in t if F occurs in t at position p and no variable occurs in t at position p' with $p = p'.q$ and a free variable F of a term t is *top-linear* iff it has at most one top-position.

Example In $\lambda x, y.b(F(a, F(a, y)), F(G(y), x), G(y))$ the variable F occurs at top-positions 1 and 2 and at position 1.2, the variable G occurs at top-position 3 and at position 2.1. Hence F is not top-linear but G is top-linear. ■

A complement problem where each second-order free variable of the t_i's is top-linear is called a *top-linear* complement problem. These problems are decidable when some restrictions are set on bound variables of the t_i's [2]:

For each bound variable x_i of t_i which is a free variable of $Y(\bar{u}_m)$ a subterm of t_i at top-position p, there exists some u_i s.t. x_i is a free variable of u_i and u_i is in the language $U ::= x_i \mid T \mid Z(U, \ldots, U)$ where T is any closed term or first-order variable, Z is a linear variable of t_i.

Example The term $@(\lambda x.F(G(x)))$, needed to define the derivative of a composition of function, satisfies the condition as well as the term $\lambda x, y.b(Y_1(\lambda z.Y_2(y, a), \lambda z.z, Y_3(y, y, Y_4)), Y_2(x, y))$ but the term $\lambda x, y.Y_1(\lambda z.Y_2(y, a, z), \lambda z.z)$ does not. ■

Top-linear terms satisfying the previous condition are called *restricted*. This limitation on bound variables is a syntactic criteria to ensure that the following lemma holds:

[2] In [Lug94b], first-order variables had to be top-linear too, and bound variables were not dealt with correctly

Lemma 1 *Let s be a restricted term, let p be a top-position of the variable Y in s, and let $Y(\bar{u}_p)$ be the subterm of s at position p, let $x_1, \ldots, x_n$ be the bound variables of s occurring in $Y(\bar{u}_p)$ then there exists an instance θ of some free variables of s occurring in $Y(\bar{u}_p)$ s.t. $Y(\bar{u}_p)\theta = H(x_1, \ldots, x_n)$.*

Proof By structural induction on the arguments. □

Example For instance $\lambda x, y.b(Y_1(\lambda z.Y_2(y, a), \lambda z.z, Y_3(y, y, Y_4)), Y_2(x, y))$ has two subterms at top-positions, namely $Y_1(\lambda z.Y_2(y, a), \lambda z.z, Y_3(y, y, Y_4))$ and $Y_2(x, y)$ and the bound variables to look at are y for the first subterm and x and y for the second one. This second one does not cause any problem, and the first subterm is equal to $H(y)$ when one uses the substitution $Y_1 \leftarrow \lambda z_1 z_2 z_3.z_3, Y_3 \leftarrow \lambda z_1 z_2 z_3.H(z_1)$. ■

Moreover if s is a restricted top-linear term, substituting a non-linear free variable of s not occurring at a top-position by some term which does not contain a free variable of s, yields a restricted top-linear term and corresponding subterms have the same set of bound variables.

Example Replacing Z' by $\lambda xy.a(H(x), y)$ in $\lambda xy.b(Y(Z(x), Z'(x, y)), Y'(Z'(y, x), y)$ gives $\lambda xy.b(Y(Z(x), a(H(x), y)), Y'(a(H(y), x), y)$. The subterm $Y(Z(x), Z'(x, y))$ corresponds to $Y(Z(x), a(H(x), y)$ and they have the same bound variables x, y. The subterm $Y'(Z'(y, x), y)$ corresponds to $Y'(a(H(y), x), y)$ and they have the same bound variables x, y. ■

In practice, many higher-order specifications satisfy the condition on bound variables. From now on, we consider *restricted* top-linear complement problems i.e. top-linear problem $\exists \bar{X} \forall \bar{Y}$: $t \neq_{\eta\beta} t_1 \wedge \ldots \wedge t \neq t_n$ s.t. each t_i is restricted. The algorithm to decide these problems has three steps:

- Use basic rules to get a conjunction of disequations where the left-hand side contains existential variables only and the right-hand side is a closed term or has a universal variable at root.

- Eliminate universal variables (introducing membership constraints).

- Decide the resulting existential formulae.

Each step is detailed in the following. For clarity, we shall assume that there is one base type only. The extension of our results to several base types is tedious but easy.

4.2 Getting universal variables at root

This step consists of using basic rules and explosion rule with the following strategy: *basic rules are used eagerly and the explosion rule is performed on an existential variable F provided that there is some disequation $s \neq_{\eta\beta} t$ with $root(s) = F$, t contains some universal variable and $root(t) \notin \bar{Y}$.*

Proposition 2 *A top-linear complement problem is equivalent to a disjunction of formulae*

$$\exists \bar{X} \forall \bar{Y} : C_1 \wedge \ldots \wedge C_p$$

where

- *either* C_i *is an equation* $\lambda \bar{x}_n.F(\bar{x}_n) = t$ *s.t.* F *is an existential variable occurring once and* $FV(t) \subseteq \bar{X}$,
- *or* C_i *is a disjunction of disequations* $s \neq_{\eta\beta} t$ *s.t.*
  - $FV(s) \subseteq \bar{X}$,
  - *either* t *is a closed term,*
  - *or* $t = \lambda \bar{x}.Y(\bar{u})$, *where* Y *does not occur elsewhere at a top-position.*

Proof First it is obvious that the application of basic rules or the explosion rule returns a formula $\exists \bar{X} \forall \bar{Y} : (s_1 \neq_{\eta\beta} t_1 \vee \ldots \vee s_n \neq_{\eta\beta} t_n) \wedge F_1 =_{\eta\beta} u_1 \wedge \ldots \wedge F_p = u_p$ where the F_i's occur once, the variables of the u_i's and of the s_i's are existentially quantified, the variables of the t_i are universally quantified. Moreover, since the initial problem is top-linear, a universal variable can occur only once at a top-position.

Then the process terminates: to each disequation $s \neq_{\eta\beta} t$ one associates the triple of integers (d, h, n) where

- if t is a closed term then $d = 0$ else $d = |p_1| + \ldots + |p_n|$ for p_i the top-positions of universal variables,
- h is the height of s,
- n is the number of symbols of $s \neq_{\eta\beta} t$.

To a disjunction of disequations, one associates the multiset of such triples. The application of basic rules decreases this complexity measure, but the case of explosion requires more work.

- if the explosion is $F = \lambda \bar{x}_n.x_i$, then the height of terms containing F decreases (even if there are multiple occurrences of F),

- if the explosion is $F = \lambda\bar{x}_n.a(H_1(\bar{x}_n), \ldots, H_p(\bar{x}_n))$ then

  - the clash rule will eventually apply to each disequation $s \neq_{\eta\beta} t$ with $root(s) = F$ and $root(t) \neq a$,

  - the decomposition rule will eventually apply to each disequation $(s \neq_{\eta\beta} t)\{F \leftarrow \lambda\bar{x}_n.a(H_1(\bar{x}_n), \ldots, H_p(\bar{x}_n))\}$ with $s = \lambda\bar{y}_m.F(u_1, \ldots, u_n)$ and $t = \lambda\bar{y}_m.a(v_1, \ldots, v_p)$ yielding n disequations $\lambda\bar{y}_m.H_i(u'_1, \ldots, u'_n) \neq_{\eta\beta} \lambda\bar{y}_m.v_i$ where $u'_i = u_i\{F \leftarrow \lambda\bar{x}_n.a(H_1(\bar{x}_n), \ldots, H_p(\bar{x}_n))\}$ which are smaller than the initial one since either v_i has no universal variable or these universal variables are higher than the universal variables of t.

Therefore the complexity measure decreases which proves that the transformation stops. Moreover, the final formula has the required form, otherwise some rule would be applicable. □

4.3 Reducing complement problem to existential formulae

Once universal variables occur at root position, the restrictions on bound variables in the t_i's allow one to eliminate these variables using the next three rules. This elimination process introduces membership constraints on bound variables which will be dealt with in the next step.

$(EUV1)$ $\exists\bar{X}\forall\bar{Y} : (\lambda\bar{x}_n.s \neq_{\eta\beta} \lambda\bar{x}_n.Y(\bar{y}_p) \vee P) \rightarrow \exists\bar{X}\forall\bar{Y} : P\{Y \leftarrow \lambda\bar{y}_p.s\}$
if $\bar{y}_p$ are distinct variables of $\bar{x}_n$ and $FV(s) \subseteq \bar{y}_p$

(EUV2) $\exists\bar{X}\forall\bar{Y} : (\lambda\bar{x}_n.s \neq_{\eta\beta} \lambda\bar{x}_n.Y(\bar{u}_p) \vee P) \rightarrow \exists\bar{X}\forall\bar{Y} : \top \parallel z_1 \in s \vee \ldots \vee z_m \in s$
where $z_1, \ldots, z_m$ are the x_i's which are free variables of s but not of $Y(\bar{u}_p)$

$(EUV3)$ $\exists\bar{X}\forall\bar{Y} : (\lambda\bar{x}_n.s \neq_{\eta\beta} \lambda\bar{x}_n.Y(\bar{u}_p) \vee P) \rightarrow \exists H, \bar{X}\forall\bar{Y} : P \parallel z_1 \notin s \wedge \ldots \wedge z_m \notin s$
where $z_1, \ldots, z_m$ are the x_i's which are free variables of s but not of $Y(\bar{u}_p)$
for Y a top-linear or linear universal variable.

The semantics of a membership constraints is obvious: σ validates $x \in s$ iff x is a free variable of $s\sigma$. This is extended to boolean combinations of such constraints. A substitution is a solution of $P \parallel \mathcal{M}$ iff σ is a solution of P which validates $\mathcal{M}$. The $\parallel$ operator is an alias for $\wedge$ but it is used in order to separate two different kinds of contraints: equality constraints and membership constraints which are not solved in the same way.

By definition, if $EUV2$ (resp. $EUV3$) is applicable, then $EUV3$ (resp. $EUV2$) is also applicable and the proof tree must contain the two possible cases for completeness sake. Usually $EUV1$ will be applied on first-order universal variables ($p = 0$). Moreover $EUV2$ and $EUV3$ are meaningful even if $m = 0$ i.e. each variable x_i of s occurs in $Y(\bar{u}_p)$ and in this case the membership constraint $z_1 \in s \vee \ldots \vee z_m \in s$ becomes $\bot$ and the constraint $z_1 \notin s \wedge \ldots \wedge z_m \notin s$ becomes $\top$.

Proposition 3 *$EUV1$ is consistent and if P_1 and P_2 are computed from P by the application of $EUV2$ and $EUV3$, then the set of solutions of P is the union of the solutions of P_1 and P_2.*

Proof The consistency of $EUV1$ is obvious since $\lambda\bar{x}_n.s\sigma =_{\eta\beta} \lambda\bar{x}_n.Y(\bar{y}_p)\sigma$ for each σ iff $Y = \lambda\bar{y}_p.s$ (which is meaningful since no x_i different from the y_j's occurs in s). The rule $EUV2$ is correct since the membership constraints proves that the disequation is always true. Accordingly, the proof is straighforward for $EUV3$ since each solution of the conclusion validates the disequation because our restriction on bound variables allows one to use lemma 1 and to find instances of linear or top-linear universal variables s.t. the two members of the disequation are equal. If Y is non-linear, we should replace the occurrences of Y in P by the values of Y falsifying the disequation. However, this is useless since Y is necessarily a top-linear variable, therefore Y cannot be the root of any other disequation and replacing Y by the values s.t. the equation $\lambda\bar{x}_n.s =_{\eta\beta} \lambda\bar{x}_n.Y(\bar{u}_p)$ is true in P does not change the way the other rules are applied to P (remember that replacing non top-linear variables by closed terms in a restricted term yields a restricted term). □

Remark 3 *The reader should notice that $EUV1$ is consistent and that $EUV2$ is correct even when no restriction is done on bound or free variables of the terms. On the other hand the correctness of $EUV3$ is based on our condition on bound variables, more precisely on the fact that each subterm occurring at a top-position in the t_i's can be made equal to any $H(x_{i_1}, \ldots, x_{i_k})$ for $x_{i_1}, \ldots, x_{i_k}$ the relevant bound variables.*

Since the equations occurring in the computed formulae are used only for defining new variable we shall not take them into account and the next step is to show how membership constraints and existential conjunction of disequations are solved.

4.4 Deciding existential formulae and membership constraints

4.4.1 Solving membership constraints

New reduction rules are introduced for these new contraints. We write $F = \lambda\bar{x}_n.C[x_{i_1}, \ldots, x_{i_p}]$ to say that the F depends on arguments $i_1, \ldots, i_p$ and $F \neq \lambda\bar{x}_n.C[x_{i_1}, \ldots, x_{i_p}]$ to say that F does not depend on arguments $i_1, \ldots, i_p$ (we assume that the variables occurring in the contexts are bound by the λ). As usual boolean rules are used to get disjunctive forms.

$x \in x(t_1, \ldots, t_m) \rightarrow \top \quad (m \geq 0)$
$x \notin x(t_1, \ldots, t_m) \rightarrow \bot \quad (m \geq 0)$
$x \in s \rightarrow \bot$ if s is a closed term not containing x
$x \notin s \rightarrow \top$ if s is a closed term not containing x
$x \in \lambda \bar{y}_m.a(u_1, \ldots, u_n) \rightarrow x \in u_1 \vee \ldots \vee x \in u_n$
$x \notin \lambda \bar{y}_m.a(u_1, \ldots, u_n) \rightarrow x \notin u_1 \wedge \ldots \wedge x \notin u_n$
$x \in \lambda \bar{y}_n.F(u_1, \ldots, u_n) \rightarrow \bigvee_{i=1,\ldots,n}(F = \lambda \bar{x}_n.C[x_i] \wedge x \in u_i)$
$x \notin \lambda \bar{y}_m.F(u_1, \ldots, u_n) \rightarrow \bigvee_{I \subseteq \{1,\ldots,n\}}(\bigwedge_{i \in I} x \notin u_i \wedge F \neq \lambda x_n.C[x_{j_1}, \ldots, x_{j_q}])$
where $x_{j_1} \ldots, x_{j_q}$ is the complement of $\{x_i \mid i \in I\}$ in $\{x_1, \ldots, x_n\}$.

For simplicity we write $C[\bar{x}_I]$ for $C[x_1, \ldots, x_p]$ if $I = \{1, \ldots, p\}$. Since we introduced new contraints expressing the dependence or the independence of some function with respect to its arguments, we show how to solve them:

$$F = \lambda \bar{x}_n.C[\bar{x}_I] \wedge F = \lambda \bar{x}_n.C[\bar{x}_J] \rightarrow F = \lambda \bar{x}_n.C[\bar{x}_{I \cup J}]$$
$$F \neq \lambda \bar{x}_n.C[\bar{x}_I] \wedge F \neq \lambda \bar{x}_n.C[\bar{x}_J] \rightarrow F \neq \lambda \bar{x}_n.C[\bar{x}_{I \cup J}]$$
$$F \neq \lambda \bar{x}_n.C[\bar{x}_I] \wedge F = \lambda \bar{x}_n.C[\bar{x}_J] \rightarrow \bot \text{ if } I \cap J \neq \emptyset$$

The rules of both sets are consistent and compute the solutions of membership constraints.

Proposition 4 *Any boolean combination of membership constraints is equivalent to* $\top$, $\bot$ *or a finite disjunction of expressions* $F_1 = \lambda \bar{x}_n.C[\bar{x}_{I_1}] \wedge F_1 \neq \lambda \bar{x}_n.C[\bar{x}_{J_1}] \wedge \ldots \wedge F_p = \lambda \bar{x}_n.C[\bar{x}_{I_p}] \wedge F_p \neq \lambda \bar{x}_n.C[\bar{x}_{J_p}]$ *with* $I_l \cap J_l = \emptyset$ *for* $l = 1, \ldots, p$.

Proof First use the rules on the membership relation which obviously terminate, and then the rules combining dependence or independence of arguments. □

4.4.2 Solving existential conjunctions of disequations

First the basic rules are employed to get rid of Rigid-Rigid disequations, and we have to deal with four different kinds of disequations:

- $\lambda \bar{x}_n.F(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_n.G(\bar{v})$
- $\lambda \bar{x}_n.F(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_n.C[\textit{some free variables}]$ with C some fixed context.
- $\lambda \bar{x}_n.F(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_n.t$ with t a closed term.
- $\lambda \bar{x}_n.F(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_n.F(\bar{v})$ with $\bar{u} \neq \bar{v}$.

The last kind of disequations complicates the resolution process since it forbids solutions like $F \leftarrow \lambda \bar{x}_n.C$ where $FV(C) \cap \bar{x}_n = \emptyset$. Therefore we show first how to solve conjunctions which do not contain this kind of disequations without using such solutions. Two cases are distinguished depending on the signature. Either $\mathcal{C}$ contains only constants of arity 0 and one constant of arity 1, or not.

In the first case each term looks like $\lambda \bar{x}_n.x_i$ or $\lambda \bar{x}_n.a_i$ or $\lambda \bar{x}_n.f^n(a_i \text{ or } x_i)$. Therefore solving disequations amounts to solving linear diophantine (dis)equations on integers (the unknowns are the exponents of f) which is decidable. One should remark that constrained disequations may have no solution in this setting, like $F(G(a)) \neq_{\eta\beta} G(F(a)) \parallel F = \lambda x.C[x] \wedge G = \lambda x.C[x]$.

In the second case, either $\mathcal{C}$ contains only constants of arity 0 and the problem is trivial, or the following proposition is true:

Proposition 5 *For each n, there is some h s.t. there are at least n contexts of height h*

From now on, we suppose that the signature satisfies the given property. The key proposition is the following one:

Proposition 6 *Let $Conj$ be an existentially quantified conjunction of disequations of the first three kinds, then*

- *either $Conj$ contains a disequation $t \neq_{\eta\beta} t$ (hence has no solution),*

- *or there are infinitely many solutions of the form*

  $F_1 \leftarrow \lambda \bar{x}_{n_1}.C_1[X_1(\bar{x}_{n_1})]$

  .

  .

  $F_p \leftarrow \lambda \bar{x}_{n_1}.C_p[X_p(\bar{x}_{n_p})]$

  where $C_1, \ldots, C_p$ are fixed contexts, $F_1, \ldots, F_p$ are the existential variables of $Conj$, where $X_i(\bar{x}_{n_i})$ stands for any term built on $\mathcal{C}$ and the x_i's.

Proof The proof is by induction on the number of disequations. In the following the reasoning on contexts strongly relies on the assumption made on the signature $\mathcal{C}$.

Base case: one disequation.

- $\lambda \bar{x}_{n_1}.F_1(\bar{u}) \neq_{\eta\beta} t$ with t a closed term. Then any substitution of the required form is a solution provided that the height of C_1 is greater than the height of t.

- $\lambda \bar{x}_{n_1}.F_1(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_{n_1}.C[F_{i_1}(\bar{v_1}), \ldots, F_{i_k}(\bar{v_p})]$

- F_1 does not occur in $F_{i_1}, \ldots, F_{i_k}$. Then there exist contexts $C_1 \ldots, C_p$ s.t $C_1[\ ] \neq C[C_{i_1}[\ ], \ldots, C_{i_k}[\ ]]$.

- F_1 occurs in $F_{i_1}, \ldots, F_{i_k}$. Firstly, one instantiates the $F_i \neq F_1$ by terms of the required form, yielding some disequation of the form $\lambda \bar{x}_{n_1}.F_1(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_{n_1}.C[F_1(\bar{u}_1), \ldots, F_1(\bar{u}_k)]$. This last term can be written $\lambda \bar{x}_{n_1}.a(t_1, \ldots, t_n)$ with $a \in \mathcal{C}$ and either some t_i, say t_1, is a constant of arity 0 and $F = \lambda \bar{x}_{n_1}.a(a[\ldots], \ldots)$ satisfies the disequation, or none of the t_i is a constant of arity 0 and $F = \lambda \bar{x}_{n_1}.a(\textit{some constant of arity } 0, \ldots)$ validates the disequation.

- $\lambda \bar{x}_{n_1}.F_1(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_{n_2}.F_2(\bar{v})$. Then there exists two contexts $C_1[\ ]$ and $C_2[\ ]$ s.t. $C_1[\ ] \neq C_2[\ ]$ therefore $F_1 \leftarrow \lambda \bar{x}_{n_1}.C_1[X_1(\bar{x}_{n_1})]$ and $F_2 \leftarrow \lambda \bar{x}_{n_2}.C_2[X_2(\bar{x}_{n_2})]$ validates the disequation whatever the other F_i's are.

Induction step: the property is true for $m-1$ disequations, and we add a new one. The solution depends on how the new disequation looks like:

- $\lambda \bar{x}_{n_1}.F_1(\bar{u}) \neq_{\eta\beta} t$ with t a closed term. Either F_1 is a new variable and we proceed as in the base case or F_1 is subject to the constraint $F_1 = \lambda \bar{x}_{n_1}.C_1[X(\bar{x}_{n_1})]$. In this case we replace X by $C'[X(\bar{x}_{n_i})]$ s.t. the height of $C_1[C'[\ ]]$ is greater than the height of t which yields solutions of the required form for the new set of disequations.

- $\lambda \bar{x}_{n_1}.F_1(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_{n_1}.C[F_{i_1}(\bar{v}_1), \ldots, F_{i_k}(\bar{v}_p)]$. To any new variable, we associate an arbitrary constraint $F_i = \lambda \bar{x}_{n_i}.C_i[X(\bar{x}_{n_i})]$. To conclude, we distinguish two cases:

  - F_1 does not occur in $F_{i_1}, \ldots, F_{i_k}$. In this case there exist contexts $C'_1, C'_{i_1}, \ldots, C'_{i_k}$ s.t. $C_1[C'_1[\ ]] \neq C[C'_{n_1}[\ ], \ldots, C'_{i_k}[\ ]]$ and we are done.

  - F_1 is one of the $F_{i_1}, \ldots, F_{i_k}$. For simplicity, we assume that no variable different from F_1 occurs. Let p be a position of $[\ ]$ in $C_1[\ ]$, then either p is not a position of $C[C_1[\ ], \ldots, C_1[\ ]]$ and we are done or p is a position of this context. Let a be the symbol at position p in this last context and let X be the variable occurring at position p in $\lambda \bar{x}_{n_1}.C_1[\bar{X}_i(\bar{x}_{n_i})]$.

    * either there is a constant b of arity greater than 0 and different from a, then replacing X by $b[\bar{X}(\bar{x}_{n_i})]$ yields a solution,

    * or a is the only function of arity greater than 1 in $\mathcal{C}$. Let $a(t_1, \ldots, t_r)$ be the subterm occurring at position p in $\lambda \bar{x}_{n_1}.C[C_1[\ ], \ldots, C_1[\ ]]$ then either all the t_i have root a and an instance of X by $a(\ldots, b, \ldots)$ yields a solution, or there is some t_{i_0} with

$root(t_i) \neq a$ then one instantiates X by $a(\ldots, a(\ldots), \ldots)$ with $a(\ldots)$ as i_0^{th} argument.

When a variable F_i different from F_1 occurs, replace F_i by $\lambda \bar{x}_{n_i}.C_i[X(\bar{x}_{n_i})]$ and the proof proceeds like above except small changes.

- $\lambda \bar{x}_{n_1}.F_1(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_{n_2}.F_2(\bar{v})$. If there is no constraint on F_1 or F_2 the problem is easily solved, otherwise $F_1 = \lambda \bar{x}_{n_1}.C_1[X(\bar{x_{n_1}})]$ and $F_2 = \lambda \bar{x}_{n_2}.C_2[X(\bar{x_{n_2}})]$. In this case one can find two contexts $C_1'[\ ]$ and $C_2'[\ ]$ s.t. $C_1[C_1'[\ ]] \neq C_2[C_2'[\ ]]$ and we are done.

□

The last thing to do is to use this result in order to get a decision method for the general case, i.e. in presence of equations $\lambda \bar{x}_n.F(\bar{u}) \neq_{\eta\beta} \lambda \bar{x}_n.F(\bar{v})$. The intuition supporting the last proposition is that if one applies the same function to some values, the results are distinct iff some arguments are distinct *and* the function depends on one of these arguments.

Proposition 7 *Let $\exists \bar{F} : \lambda \bar{x}_n.F(\bar{u}_p) \neq_{\eta\beta} \lambda \bar{x}_n.F(\bar{v}_p)$ be a disequation, then the disequation has a solution iff $\bigvee_{i=1,\ldots,p}(\lambda \bar{x}_{n_1}.u_i \neq_{\eta\beta} \lambda \bar{x}_{n_1}.v_i \parallel F = \lambda \bar{y}_p.C[y_i])$ has a solution.*

Proof We prove each implication:

- *Necessary condition*
  Let θ be a solution of the disequation, then $F\theta = \lambda \bar{x}_p.C[x_{i_1}, \ldots, x_{i_m}]$ with $m > 0$ since constant function are not solution of the disequation. Since $C[u_{i_1}\theta, \ldots, u_{i_m}\theta] = F(\bar{u}_p)\theta \neq_{\eta\beta} F(\bar{v}_p)\theta = C[v_{i_1}\theta, \ldots, v_{i_m}\theta]$, there is some i_j of $i_1, \ldots, i_p$ s.t. $u_{i_j}\theta \neq_{\eta\beta} v_{i_j}\theta$

- *Sufficient condition*
  Let θ be a solution of $\lambda \bar{x}_{n_1}.u_i \neq_{\eta\beta} \lambda \bar{x}_{n_1}.v_i \parallel F = \lambda \bar{y}_p.C[y_i]$, then $F\theta = \lambda \bar{x}_p.C[x_i]$ for some context C. Therefore $F(\bar{u}_p)\theta = C[u_i\theta]$ and $F(\bar{v}_p)\theta = C[v_i\theta]$ with $u_i\theta \neq_{\eta\beta} vi\theta$, i.e. θ is a solution of the initial disequation.

□

Therefore, we can state:

Proposition 8 *Existential constrained conjunction of disequations are decidable.*

Proof First we get rid of disequations with the same root F by the rule:

$$\exists \bar{H}_m : (\lambda \bar{x}_n.F(\bar{u}_p) \neq \lambda \bar{x}_n.F(\bar{v}_p) \wedge P) \parallel M \rightarrow \bigvee_{i=1,\ldots,p} \exists \bar{H}_m : (\lambda \bar{x}_n.(u_i \neq \lambda \bar{x}_n.v_i \wedge P) \parallel F = \lambda \bar{x}_p.C[x_i] \wedge M$$

This rule is correct because of proposition 7 and we can use it until there is no disequation of the fourth type, getting either $\perp$ or constrained systems which have solution (use proposition 6). □

4.4.3 The decidability result

Putting all previous results together, one can state the theorem:

Theorem 4 *The restricted second-order top-linear complement problem is decidable.*

Proof First reduce problems to simpler ones as described in section 4.2, then eliminate universal variables as in section 4.3 and finally decide the resulting existential formulae as in section 4.4. □

In some sense our result is optimal since it states a decidability result when non-linear first-order variables are allowed but section 2.2 gives an undecidability result when second-order non-linear variables occurs in the t_i's.

4.5 The *map* example (continued)

In section 3.5, the simplification of the formula associated to the completeness of the definition of the *map* function ended with two formulae which still contained universal variables. We show how to proceed with these formulae by eliminating universal variables.

The first formula was:

$$\exists F, L \forall Z_1, Z_2, X, L_1 : \ \lambda x.F(x) \neq_{\eta\beta} \lambda x.Z_1(x) \wedge L = nil$$

then applying $EUV1$ yields $\bot$ (the rule is applicable since the bound variable x occurs in $Z_1(x)$).

The second formula was:

$$\exists X', L', F, L \forall Z_1, Z_2, X, L_1 : \ (\lambda x.F(x) \neq_{\eta\beta} \lambda z.Z_2(z) \vee X' \neq_{\eta\beta} X \vee L \neq_{\eta\beta} L_1) \wedge L = cons(X', L')$$

again $EUV1$ can be applied and one gets $\bot$.

Therefore each case of the explosion rule yields $\bot$, which means that the initial formula is not valid and that the proposed definition of *map* is complete.

5 Decidability of equational problems on patterns

5.1 Constrained equational problems on patterns

In this section we discuss how to solve equational problems when each term is a *pattern*. This class of lambda-terms has been introduced by Dale Miller [Mil91] who proved that unification of patterns is decidable and unitary. Patterns are now widely used in current implementation of higher-order logical languages like lambda-prolog and Elf and appear to be good candidates for extending first-order languages. Let us recall the definition of a pattern.

Definition 3 *A simply typed lambda-term t is a pattern iff the arguments of any free variable of t are ($\eta\beta$ equal to) distinct bound variables.*

Example $\lambda x, y.F(x, y)$, $\lambda x, y.G(\lambda z.x(z), y)$ are patterns but $\lambda x, y.F(x, a)$, $\lambda x, y.F(G(x), y)$ are not. ■

A key remark is that the replacement of a free variable in a pattern by another pattern yields a pattern after reduction to long $\eta\beta$ normal form. From now on, we consider equational problems where each term occurring in an equation or a disequation is a pattern. For example, the *map* example given in section 1.2 and dealt with in section 3.5 and 4, is also an equational problem on patterns.

As noticed in section 4, a equation or a disequation may have or have not solutions if a functional variable F of type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$ depends or not of its arguments. For example $\lambda xy.F(x) =_{\eta\beta} \lambda xy.G(y)$ has no solution if F or G depend of their argument but has a solution if F and G are constant functions. Since we are interested in the decidability of any formulae on patterns and not equational problems only[3], we introduce a new framework expressing dependency constraints for variables which is more precise than the membership constraints introduced in section 4. We write $F[I]$ where $I = \emptyset$ or $I = \{i_1, \ldots, i_p\} \subseteq \{1, \ldots, n\}$ to express that F is a function which depends of arguments $i_1, \ldots, i_p$ but does not depend of arguments j if $j \notin I$. This means that F can be instantiated only by terms $\lambda \bar{x}_n.s$ with $FV(s) = \{x_i \mid i \in I\}$. If F is a first-order variable, we shall write $F[\emptyset]$.

Example If F has type $Nat \times Nat \rightarrow Nat$ then $F[\{1\}]$ can be instantiated by terms $\lambda xy.s$ with $x \in FV(s)$ but it cannot be instantiated by terms $\lambda xy.s$ with $FV(s) = \emptyset$ or s.t. $y \in FV(s)$. ■

Therefore equational problems are replaced by constrained equational problems ($\overline{F[I]}$ denotes $F_1[I_1], \ldots, F_m[I_m]$):

Definition 4 *A constrained equational problem on patterns is a formula*

$$\overline{X[I]}\ \exists \overline{Z[J]}\ \forall \overline{Y[K]} : P$$

where P is a conjunction of disjunction of equations or disequations s.t. each term occurring in P is a pattern and $FV(P) = \bar{X} \cup \bar{Y} \cup \bar{Z}$. A variable appears once in $\overline{X[I]}\ \exists \overline{Z[J]}\ \forall \overline{Y[K]}$ which is called the prefix part of the problem.

Solutions of constrained equational problems are defined as for equational problems except that $F[I]$ states that the free existential or universal variable F ranges over closed terms $\lambda \bar{x}_n.s$ s.t. $FV(s) = \{x_i \mid i \in I\}$.

There is an easy way to get constrained problems from unconstrained ones:

[3] a simpler and more efficient procedure for equational problems is described in [Lug94a]

φ	$\rightarrow$	$\bigvee_{I \subseteq \{1,\ldots,n\}} X[I]\ \varphi$ if X is a free variable of φ of type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$.
$\exists Z\ \varphi$	$\rightarrow$	$\bigvee_{I \subseteq \{1,\ldots,n\}} \exists Z[I]\ \varphi$ if Z has type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$.
$\forall Y\ \varphi$	$\rightarrow$	$\bigwedge_{I \subseteq \{1,\ldots,n\}} \forall Y[I]\ \varphi$ if Y has type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$.

Since any closed instance of a variable F is a closed instance of some $F[I]$, this transformation preserves the set of solutions.

Remark 5 *Constrained formulae have the usual behavior under negation, i.e.* $\neg(\overline{X[I]}\ \exists \overline{Z[J]} : \varphi)$ *is equivalent to* $\overline{X[I]}\ \forall \overline{Z[J]} : \neg\varphi$ *and a similar result holds for negation of universal formulae.*

Proof $\exists \overline{Z[I]} : \varphi$ is equivalent to $\exists Z : Z \in D_I \wedge \varphi$ where D_I is the set of closed terms related to the constraint I, therefore the negation is equivalent to $\forall Z : (Z \notin D_I \vee \neg\varphi)$. The disjunction is true for the $Z \notin D_I$, therefore the formula is equivalent to $\forall \overline{Z[I]} : \neg\varphi$. The reasoning holds also when free variables occur and works for universal formulae too. □

For simplicity, we shall assume that for each constraint $F[I]$ where F has type τ occurring in the prefix, there are infinitely many closed terms of type τ which satisfy the constraint, otherwise F is replaced by all its possible instances (yielding a disjunction if F is free or existential, a conjunction if it is universal). The finiteness or infiniteness of this set of closed terms can be decided by looking at all possible closed terms of height less than some bound (computed from τ and $\mathcal{C}$).

5.2 The transformation rules

The transformation rules are designed for equational problems, and we shall see how to solve any quantified formulae later on. The first rules are the basic rules as defined in section 3. For simplicity we may write $H(\bar{x}_n)$ instead of its long eta-beta normal form $\lambda \bar{z}_q.H(\bar{x}_n, \bar{z}_q)$ for some new variables H.

Explosion rules for free or existential variables

The explosion rule for existential or free variables is rephrased as:

(EX) $X[I]\ \exists Z[J]\ \forall Y[K] : P \rightarrow$ $X[I]\ \exists Z[J]\ \exists H_1[J_1], \ldots, H_p[J_m] \forall Y[K] :$
$F = \lambda \bar{x}_n.a(H_1(\bar{x}_n), \ldots, H_m(\bar{x}_n)) \wedge$
$P\{F \leftarrow \lambda \bar{x}_n.a(H_1(\bar{x}_n), \ldots, H_m(\bar{x}_n))\}$

for any $a \in \mathcal{C} \cup \{x_i \mid i \in I\}$ and any $J_1, \ldots, J_m$ s.t. $(J_1 \cup \ldots \cup J_m) \cap \{1, \ldots, n\} = I$
if F is a free or existential variable s.t. $F[I]$ belongs to the prefix.

Explosion of universal variables

(EXUV) $\forall Y[I] : (\lambda \bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.a(\bar{u}_m) \vee P)$
$\rightarrow$
$\bigwedge_{(J_1 \cup \ldots \cup J_m) \cap \{1,\ldots,p\} = I} \forall H_1[J_1], \ldots, H_m[J_m] :$
$(\lambda \bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.a(\bar{u}_m) \vee P\{Y \leftarrow \lambda \bar{y}_m.a(H_1(\bar{y}_m), \ldots, H_p(\bar{y}_m))\})$
if $\bar{u}_m$ contains some existential, free or universal variable of the equational problem.

Occurrence test rules

$(OC1)$ $\lambda \bar{x}_n.F(\bar{y}_n) =_{\eta\beta} \lambda \bar{x}_n.t \rightarrow \bot$ $(OC2)$ $\lambda \bar{x}_n.F(\bar{y}_n) \neq_{\eta\beta} \lambda \bar{x}_n.t \rightarrow \top$
if $root(t)$ is not F and $F \in FV(t)$.

Compatibility rules for Flexible-Flexible cases

F and G are two free, existential or universal variables of the equational problem s.t. $F[I]$ and $G[J]$ occur in the prefix. These rules takes into account the dependence constraints stated in the prefix part of the equational problem.

(CO1) $\lambda \bar{x}_n.F(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.G(\bar{z}_q) \rightarrow \top$ (CO2) $\lambda \bar{x}_n.F(\bar{y}_p) =_{\eta\beta} \lambda \bar{x}_n.G(\bar{z}_q) \rightarrow \bot$
if $\{y_i \mid i \in I\} \neq \{z_j \mid j \in J\}$.

(CO3) $\lambda \bar{x}_n.F(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.F(\bar{z}_p) \rightarrow \top$ (CO4) $\lambda \bar{x}_n.F(\bar{y}_p) =_{\eta\beta} \lambda \bar{x}_n.F(\bar{z}_p) \rightarrow \bot$
if $\exists i \in I$ s.t $y_i \neq z_i$.

(CO5) $\lambda \bar{x}_n.F(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.F(\bar{z}_p) \rightarrow \bot$ (CO6) $\lambda \bar{x}_n.F(\bar{y}_p) =_{\eta\beta} \lambda \bar{x}_n.F(\bar{z}_p) \rightarrow \top$
if $\forall i \in I, y_i = z_i$.

Flexible-Rigid disequation: universal-constant or bound variable case

(EUVD1) $\forall Y[I] : (\lambda \bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.s \vee P) \rightarrow P\{Y \leftarrow \lambda \bar{y}_p.s\}$
if $\lambda \bar{x}_n.s$ is a closed term s.t. $FV(s) = \{y_i \mid i \in I\}$

(EUVD2) $\forall Y[I] : (\lambda \bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.s \vee P) \rightarrow \top$
if $\lambda \bar{x}_n.s$ is a closed term s.t. $FV(s) \neq \{y_i \mid i \in I\}$

Flexible-Flexible disequation: universal-universal case

(EUVD3) $\forall Y[I] : (\lambda \bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.Y(\bar{z}_p) \vee P) \rightarrow \forall H[L] : P\{Y \leftarrow \lambda \bar{y}_p.H(\bar{w}_l)\}$
if $\forall i \in I \;\; y_i = z_i$, $\bar{w}_l = \{y_i \mid i \in I\}$ and $L = \{1, \ldots, l\}$

(EUVD4) $\forall Y_1[I]\; Y_2[J] : (\lambda \bar{x}_n.Y_1(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.Y_2(\bar{z}_q) \vee P)$
$\rightarrow$
$\forall H[K] : P \left\{ \begin{array}{l} Y_1 \leftarrow \lambda \bar{y}_p.H(\bar{w}_k), \\ Y_2 \leftarrow \lambda \bar{y}_q.H(\bar{w}_k) \end{array} \right\}$
if $\{y_i \mid i \in I\} = \{z_j \mid j \in J\} = \{w_l \mid l \in K\}$ and $K = \{1, \ldots, k\}$.

Flexible-Flexible disequation: universal-free or existential case

(EUVD5) $\exists Z[I]\; \forall Y[J] : (\lambda \bar{x}_n.Z(\bar{y}_p) \neq_{\eta\beta} \lambda \bar{x}_n.Y(\bar{z}_q) \vee P)$
$\rightarrow$
$\exists H[K]\; \forall Y[J] : P \left\{ \begin{array}{l} Z \leftarrow \lambda \bar{y}_p.H(\bar{w}_k), \\ Y \leftarrow \lambda \bar{z}_q.H(\bar{w}_k) \end{array} \right\}$
if $\{y_i \mid i \in I\} = \{z_j \mid j \in J\} = \{w_l \mid l \in K\}$ and $K = \{1, \ldots, k\}$.

A similar rule exists for free variables.

Universal variables in equations

(EUVE) $X[I]\; \exists Z[J]\; \forall Y[K] : (s_1 =_{\eta\beta} t_1 \vee \ldots \vee s_n =_{\eta\beta} t_n \vee P) \rightarrow X[I]\; \exists Z[J]\; \forall Y[K] : P$
if each $s_i =_{\eta\beta} t_i$ contains a universal variable, P contains no universal variable, no rule (except explosion rules) is applicable to any $s_i =_{\eta\beta} t_i$.

5.3 An example

Before giving consistency and correctness results on the above set of rules, we show how to use these rules. Let us consider the (unconstrained) equational problem:

$$\forall Y : \; (\lambda xyz.Y(x,y,z) \neq_{\eta\beta} \lambda xyz.Y(x,z,y) \vee \lambda xyz.Y(x,y,z) \neq_{\eta\beta} \lambda xyz.Y(y,x,z) \vee \lambda xyz.Z(x) \neq_{\eta\beta} \lambda xyz.Y(x,y,z))$$

The first step is to transform this problem into constrained equational problems. There are two possible constraints for Z, i.e. $Z[\emptyset]$ and $Z[\{1\}]$, and 8 possible constraints for Y, leading to 16 constrained problems and we describe the transformation process on some of them only. For simplicity we write $F[i]$ instead of $F[\{i\}]$ for integers i.

Constraints $Z[\emptyset]$

Let us see what happens with the problem:

$$Z[\emptyset]\ \forall Y[\emptyset]: (\lambda xyz.Y(x,y,z) \neq_{\eta\beta} \lambda xyz.Y(x,z,y) \vee \lambda xyz.Y(x,y,z) \neq_{\eta\beta} \lambda xyz.Y(y,x,z) \vee \lambda xyz.Z(x) \neq_{\eta\beta} \lambda xyz.Y(x,y,z))$$

$$\rightarrow_{EUVD3}$$

$$Z[\emptyset]\ \forall H[\emptyset]: (\vee\lambda xyz.Y(x,y,z) \neq_{\eta\beta} \lambda xyz.Y(y,x,z) \vee \lambda xyz.Z(x) \neq_{\eta\beta} \lambda xyz.Y(x,y,z))\{Y \leftarrow \lambda xyz.H\}$$

which is simplified into:

$$Z[\emptyset]\ \forall H[\emptyset]: (\lambda xyz.H \neq_{\eta\beta} \lambda xyz.H \vee \lambda xyz.Z(x) \neq_{\eta\beta} \lambda xyz.H)$$

$$\rightarrow_{ET2}$$

$$Z[\emptyset]\ \forall H[\emptyset]: \lambda xyz.Z(x) \neq_{\eta\beta} \lambda xyz.H)$$

$$\rightarrow_{EUVD5}$$

$$\perp$$

Therefore there is no solution for the constraint $Z[\emptyset]$ (even if the other problems with the constraint $Z[\emptyset]$ do not return $\perp$).

Constraint $Z[1]$

Let us see what happens with the problem:

$$Z[1]\ \forall Y[2]: (\lambda xyz.Y(x,y,z) \neq_{\eta\beta} \lambda xyz.Y(x,z,y) \vee \lambda xyz.Y(x,y,z) \neq_{\eta\beta} \lambda xyz.Y(y,x,z) \vee \lambda xyz.Z(x) \neq_{\eta\beta} \lambda xyz.Y(x,y,z))$$

$$\rightarrow_{CO3}$$

$$[Z[1]\ \forall Y[2]: (\lambda xyz.Y(x,y,z) \neq_{\eta\beta} \lambda xyz.Y(y,x,z) \vee \lambda xyz.Z(x) \neq_{\eta\beta} \lambda xyz.Y(x,y,z))$$

$$\rightarrow_{CO3}$$

$$Z[1]\ \forall Y[2] : \lambda xyz.Z(x) \neq_{\eta\beta} \lambda xyz.Y(x,y,z)$$

$$\rightarrow_{CO1}$$

$$Z[1] : \top$$

The reader may check that all other constrained problems with constraint $Z[1]$ gives $\top$, which means that the solutions of the initial unconstrained problem are the $Z = \lambda x.Z(x)$ s.t. Z depends on its argument x.

5.4 Decidability of Equational problems

5.4.1 Consistency of rules

Proposition 9 *The explosion rule for free or existential variable is correct and the other rules are consistent.*

Proof This statement is a straightforward consequence of Miller's result on pattern unification except for $EUVE$. Miller's result for Flexible-Flexible pairs states that the unification problems:

$$\lambda\bar{x}_n.U(\bar{y}_p) =_{\eta\beta} \lambda\bar{x}_n.V(\bar{z}_q)$$

has a most general unifier $U \leftarrow \lambda\bar{y}_p.H(\bar{w}_l), V = \lambda\bar{z}_q.H(\bar{w}_l)$ where $\{\bar{w}_l\} = \{\bar{y}_p\} \cap \{\bar{z}_q\}$. Moreover the unification problem

$$\lambda\bar{x}_n.U(\bar{y}_p) =_{\eta\beta} \lambda\bar{x}_n.U(\bar{z}_q)$$

has a most general unifier $Y \leftarrow \lambda\bar{y}_p.H(\bar{w}_l)$ where $w_l = \{y_i \mid y_i = z_i\}$. Combining these results and the dependence constraints stated in the prefix part yields the consistency of the compatibility rules. Rules $CO1$ to $CO4$ states that two flexible terms can not be unifiable if one of them must contain a bound variable which can not appear in the other one. The two last rules states that bound variables which do not appear actually in the terms can be dropped. The proof of correctness is similar for the $EUVDi$'s rules. For example $EUVD4$ relies on the fact that the equation $\lambda\bar{x}_n.Y_1(\bar{y}_p) =_{\eta\beta} \lambda\bar{x}_n.Y_2(\bar{z}_q)$ is true only for instances $Y_1 = \lambda\bar{y}_p.H(\bar{w}_l), Y_2 = \lambda\bar{z}_q.H(\bar{w}_l)$ for all H s.t. $H[L]$ is true when Y_1 and Y_2 satisfy the constraints $Y_1[I]$ and $Y_2[J]$.

The correctness of $EUVE$ requires the following lemma which is used also for the decidability of the solved forms computed by the transformation process:

Lemma 2 *A disjunction* $\exists\overline{W[I]}\ \forall\overline{Y[J]} : (s_1 =_{\eta\beta} t_1 \vee \ldots \vee s_m = t_m)$ *where* s_i *and* t_i *are distinct patterns and s.t. no basic occurrence test or compatibility rule is applicable, has no solution when each equation contains an occurrence of some* Y_i.

Proof The proof of this lemma is by induction on the number of universal variables.

Base Case: instantiate free and existential variables, then one gets equations $\lambda\bar{x}_n.Y(\bar{y}_p) =_{\eta\beta} r$ with r a closed term which are falsified by any instance of $Y \leftarrow \lambda\bar{x}_m.C[\bar{x}_I]$ if $Y[I]$ is the constraint on Y occurring in the prefix. No other equation can happen otherwise occurrence test or compatibility rules are applicable. Therefore any instance $Y \leftarrow \lambda\bar{x}_p.C[\bar{x}_I]$ for any context C high enough falsifies the disjunction and we are done.

Induction case: Given an instantiation of free and existential variables, by induction hypothesis there exists an instantiation of $Y_2, \ldots, Y_{n-1}$ which falsifies all equations which contain only these universal variables. Then the situation is similar to the base case, and one can choose a suitable instance of Y_n s.t. the disjunction is false. □

The consistency of $EXUV$ rule is also easy: since Y occurs in a disequation $s \neq_{\eta\beta} t$ with $root(s) = Y$ and $root(t) = a$, each instance of Y a term with root $b \neq a$ yields $\top$. Therefore the only instances of Y to consider are the instances with root a. Since Y is subject to dependence constraints, the term $a(H_1(\ldots), \ldots, H_m(\ldots))$ is subject to the same constraints which gives the relevant constraints on the H_i's.

The correctness of the EX rules works like in the unconstrained case, and we also have to consider all possible cases (for a and the possible dependency constraints) for completeness sake. □

5.4.2 Termination proof

The first point is that no universal variable remain when no rule is applicable and that one gets an expression which can be rewritten as a disjunction of existentially quantified conjunction of equations or disequations. The second point is to have some strategy for the application of rules:

- the basic rules, occurrence test rules, compatibility rules and elimination of universal variable rules have the highest priority,

- a universal variable Y is exploded if none of the previous rule is applicable and if Y occurs in a disequation $\lambda\bar{x}_n.Y(\ldots) \neq_{\eta\beta} t$ where $root(t)$ is not a variable and t contains some free, existential or universal variable. Moreover we set a priority on universal variables in the following way: the initial variable are ordered arbitrarily $Y_1 \succ Y_2 \succ \ldots$. When several variables can be exploded, the explosion takes place on the variable with the highest priority and if $H_1, \ldots, H_p$ are introduced by exploding some universal variable the ordering becomes $H_1 \succ \ldots \succ H_p \succ$ *the previous ordering.*

- the explosion of an existential or free variable F is performed if no other rule is applicable and if F occurs in a equation (resp. disequation) $\lambda\bar{x}_n.F(\ldots) =_{\eta\beta} t$ (resp. $\neq_{\eta\beta}$) where $root(t)$ is not a variable and s contains some universal variable.

To prove termination, we define a complexity measure on equational problems and we show that the application of rules decreases this measure. In fact the complexity may increase after the application of some rule, but we show that the next applications decrease the complexity to some level less than the initial one. The complexity measure of a conjunction is the multiset of complexity measure of each conjunct, and the complexity measure of a disjunction is the triple $(NUV, PUV, NSYM)$ where NUV is the number of universal variables occurring in the disjunction, $PUV = 0$ if there is no universal variable or $\Sigma|p|$ for p top-position of universal variables, $NSYM$ is the number of symbols (excepting bound variables and the λ symbol. Triples are ordered lexicographically.

Proposition 10 *The application of rules according to the previous strategy terminates.*

Proof We show that each application of rule leads to a smaller complexity either immediately or later on.

- Basic rules, occurrence test rules and compatibility rules do not increase NUV nor PUV but decrease $NSYM$.
- The rules for the elimination of universal universal variables in disequation decrease NUV except for $EUVD3$ which does not increase NUV but decrease PUV since one equation with some universal variable has vanished and since the variable H replacing Y appears at the same positions in the other equations or disequations.
- $EUVE$ decreases NUV.
- The case of explosion rules requires more work since explosion rules increase the complexity measure first and one gets a smaller complexity than the initial one only after a sequence of applications of rules.

  - First, we prove that the universal variables introduced by an explosion rule are eventually eliminated. The proof is by induction on $N = Max\{height(t) \mid \lambda\bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} t \text{ occurring in the problem}\}$.

    Base case: $N = 1$ then no explosion can take place and all disequations $\lambda\bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} t$ are eliminated by the $EUVDi$'s rules.

    Induction step: the induction hypothesis is
    each variable occurring in disequations $\lambda\bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} t$ with $height(t) < N$ can be eliminated as well as each new introduced universal variable.

    Suppose that no rule is applicable and let Y be some exploded variable, then let us see how disequations $\lambda\bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} t$ with $t = \lambda\bar{x}_n.a(t_1, \ldots, t_m)$ and $height(t) = N$ are

transformed. The instances $Y \leftarrow \lambda \bar{x}_p.b(H_1(\bar{x}_n), \ldots, H_p(\bar{x}_n))$ yield $\top$ (and the relevant disjunctions vanish) and the instances $Y \leftarrow \lambda \bar{x}_p.a(H_1(\bar{x}_n), \ldots, H_m(\bar{x}_n))$ yields the disjunction
$\bigvee_{i=1,\ldots,m} \lambda \bar{x}_n.H_i(\bar{y}_n) \neq_{\eta\beta} \lambda \bar{x}_n.t_i$. Some of the H_i can be eliminated directly by the basic or $EUVDi$'s rules, and the induction hypothesis can be applied to each of the remaining ones (remember that in patterns no free variable occurs in the scope of another one and that disequations $\lambda \bar{x}_n.H_i(\bar{y}_n) \neq_{\eta\beta} s$ necessarily come from disequations $\lambda \bar{x}_n.Y(\bar{y}_p) \neq_{\eta\beta} t$).

- Then we show that the explosion of a free or existential variables is followed by rules which decrease NUV. Let F be the exploded variable, then F occurs in equations (resp.disequations) $\lambda \bar{x}_m.F(\bar{y}_n) \neq_{\eta\beta} \lambda \bar{x}_m.a(t_1, \ldots, t_p)$ (resp. $\neq_{\eta\beta}$) where at least one t_i contains a universal variable.The explosion is either $F \leftarrow \lambda \bar{x}_n.a(H_1(\bar{x}_n), \ldots, H_l(\bar{x}_n))$ or $F \leftarrow \lambda \bar{x}_n.b(H_1(\bar{x}_n), \ldots, H_l(\bar{x}_n))$ with $a \neq b$, therefore the clash or the decomposition rules eventually apply to this equation (resp. disequation) since no other rule is applicable when the explosion rule is applied. This yields either $\bot$ (resp. $\top$) or n smaller equations (resp. disequations) $\lambda \bar{x}_m.H() =_{\eta\beta} \lambda \bar{x}_m.t_i$ (resp. $\neq_{\eta\beta}$) since either t_i has no universal variable or they occur at smaller positions. Therefore explosion of existential or free variables eventually decrease PUV.

□

The last step is to transform existentially quantified into constrained solved forms i.e. $\bot$, $\overline{X[I]} : \top$ or formulae:

$$\overline{X[I]}\ \exists \overline{Z[J]} : \xi_1 =_{\eta\beta} t_1 \wedge \ldots \wedge \xi_n =_{\eta\beta} t_n \wedge z_1 \neq_{\eta\beta} s_1 \ldots \wedge z_m \neq_{\eta\beta} s_m$$

where

- each term is a pattern,
- for all i, ξ_i and t_i have the same type, $\xi_i = \lambda \bar{u}_{n_i}.X_i(\bar{u}_{n_i})$, and X_i is a variable which occurs once,
- for all j, z_j and s_j have the same type and $z_i = \lambda \bar{u}_{n_i}.Z_i(\bar{v}_{n_i})$ with Z_i a variable and $\bar{v}_{n_i}$ a permutation of a subset of $\bar{u}_{n_i}$

Example $X[1]\ \exists Z_1[\emptyset], Z_2[\emptyset] : \lambda x.X(x) =_{\eta\beta} \lambda x.f(x, Z_1, Z_2) \wedge Z_1 \neq_{\eta\beta} Z_2$ is a constrained solved form. ■

Solved forms are computed using a adaptation of the classical rules for pattern unification, i.e. we use basic rules, occurrence test rules, the explosion rule, compatibility rules and the new rules:

Flexible-Rigid unification pairs

F is a free or existential variable subject to the constraint $F[I]$.

(FR1) $\lambda\bar{x}_n.F(\bar{y}_p) =_{\eta\beta} \lambda\bar{x}_n.s \wedge P \rightarrow \lambda\bar{y}_p.F(\bar{y}_p) =_{\eta\beta} \lambda\bar{y}_p.s \wedge P\{F \leftarrow \lambda\bar{y}_p.s\}$
if $\lambda\bar{x}_n.s$ is a closed term s.t. $FV(s) = \{y_i \mid i \in I\}$

(FR2) $\lambda\bar{x}_n.F(\bar{y}_p) =_{\eta\beta} \lambda\bar{x}_n.s \wedge P \rightarrow \bot$
if $\lambda\bar{x}_n.s$ is a closed term s.t. $FV(s) \neq \{y_i \mid i \in I\}$

Flexible-Flexible unification pairs

F and G and free or existential variables subject to the constraints $F[I]$ and $G[J]$.

(FF1) $\lambda\bar{x}_n.F(\bar{y}_p) =_{\eta\beta} \lambda\bar{x}_n.G(\bar{z}_q) \wedge P$
$\rightarrow$
$$\exists H[L] : \lambda\bar{y}_p.F(\bar{y}_p) = \lambda\bar{y}_p.H(\bar{w}_l) \wedge \lambda\bar{z}_q.G(\bar{z}_q) = \lambda\bar{z}_q.H(\bar{w}_l) \wedge P \left\{ \begin{array}{l} F \leftarrow \lambda\bar{y}_p.H(\bar{w}_l), \\ G \leftarrow \lambda\bar{z}_q.H(\bar{w}_l)\} \end{array} \right\}$$
if $\{y_i \mid i \in I\} = \{z_j \mid j \in J\} = \{w_k \mid k \in L\}$ et $L = \{1, \ldots, l\}$

(FF2) $\lambda\bar{x}_n.F(\bar{y}_p) =_{\eta\beta} \lambda\bar{x}_n.F(\bar{z}_p) \wedge P$
$\rightarrow$
$$\exists H[L] : \lambda\bar{y}_p.F(\bar{y}_p = \lambda\bar{y}_p.H(\bar{w}_l) \wedge P\{F \leftarrow \lambda\bar{y}_p.H(\bar{w}_l)\}$$
if $\forall i \in I$, $y_i = z_i$ and $\bar{w}_l = \{y_i \mid i \in I\}$, $L = \{1, \ldots, l\}$.

These rules terminate when the explosion rule is restricted to the classical imitation and projection cases, i.e. exploded variables F must occur in equations $\lambda\bar{x}_n.F(\ldots) =_{\eta\beta} s$ s.t $root(s)$ is neither a free nor existential variable but s contains one existential or free variable. The proof is the same as for pattern unification, see [Nip91, Mil91].

Our last preliminary result is that solved forms are decidable, which is achieved by the next lemma:

Lemma 3 *It is decidable when a constrained solved form has a solution.*

Proof The case with $\bot$ and $\top$ is obvious, let us deal with the third kind of solved forms.

A straightforward adaptation of the proof of lemma 2 (use negation) shows that an existentially quantified conjunction of disequations always has a solution $\theta = \{Z_1 \leftarrow \lambda\bar{w}_{n_1}.C_1[\bar{w}_{I_1}], \ldots, Z_l \leftarrow \lambda\bar{w}_{n_l}.C_l[\bar{w}_{I_l}]\}$ when the constraints are $Z[I_1], \ldots, Z[I_l]$. Now let $\lambda\bar{x}_p.X(\bar{x}_p) = \lambda\bar{x}_p.t$ be an equation where the variable X occurs once in the equational part and is subject to the constraint $X[I]$, then the constraint is equivalent to $x_i \in t$ for each $i \in I$. This is easily decided by the rules:

$x_i \in \lambda\bar{y}_m.x_i(\ldots) \rightarrow \top$
$x_i \in \lambda\bar{y}_m.a(\bar{u}_n) \rightarrow x_i \in u_1 \vee \ldots \vee x_i \in u_m$
$x_i \in \lambda\bar{y}_m.F(\bar{u}_n) \rightarrow \bigvee_{j \in J} x_i \in u_j$ if $F[J]$ is the constraint on F.

□

Therefore one can state the main decidability result:

Theorem 6 *Constrained equational problems on pattern are decidable.*

This results has two consequences:

Theorem 7 *Equational problems on patterns are decidable.*

Proof Obvious from the transformation of an equational problem into constrained equational problems. □

and

Theorem 8 *Any quantified equational formulae on patterns is decidable.*

Proof First, we transform formulae into constrained formulae. Then the previous results shows that $X[I]\ \exists Z[J]\ \forall Y[K] : \varphi$ can be transformed into $X[I]\ \exists W[L] : \psi$ formulae, which means that -using repeated negation- $X[I]\ \forall Z[J]\ \exists Y[K] : \varphi$ formulae are equivalent to $X[I]\ \forall W[L] : \psi'$ formulae therefore to $X[I]\ \exists U[M] : \varphi'$ formulae. This process can be iterated in order to get rid of any alternation of quantifier, yielding the decidability result. □

Conclusion

Before sketching some possible improvements of our work, we discuss previous approaches to similar works. First, there have been several (recent) works [ACS94, Pre94a], inspired by the same idea of extending what is done for solving equations on first-order terms in a higher-order algebraic framework and their solution is to consider only terms which belong to some subclass with good properties. Unfortunately resolution processes like narrowing usually build terms which do not stay in the given subclass and most unification problems arising during the resolution process are undecidable unless when very strong syntactical restrictions are imposed on the terms which are used. Moreover no quantification beyond existential quantification is allowed.

Another interesting approach to unification is Miller's work on prefix unification [Mil92] which allows quantification in front of unification problems. This allows him to gain expressive power (constants can be simulated by variables) and to see bound variables as universal variables. A main point in his work is his skolemization technique which is proved correct for unification problems. However, there are some fundamental differences with our results, mainly due to different motivations. Quantified unification comes from higher order proof schemes when equational problems were initiated by the study of specification languages where the set of constants is given. Therefore using quantification for defining constants is not relevant for us, and we are interested in decidable fragments when Miller's approach deals with unrestricted unification which is undecidable. On the other hand we are dealing with formulae which are much more complex than unification

problems since we may have boolean connectives and negation. For example, it is not yet clear if the skolemization process used in Miller's work can be used in our approach (do we need to assume explicitely the choice axiom?). Even the very particular formulae involved by completeness of definitions are usually more complicated than unification problems. For example, in the first order case, unification problems can be decidable when complement problems are undecidable if some axioms are involved (see the associativity axiom for example).

A simple improvement of our work is to extend our definition of patterns in order to allow repeated bound variables in the arguments of a free variable, as in Prehofer's work [Pre94b]. In this case, pattern's unification is finitary instead of unitary, and technical changes only are required to adapt our method to this case. Another possible extension of our work is to have an infinite set of constants, since we know that first-order disunification is also decidable in this case. This extension is easy since the constant which are relevant are the constant which actually occur in the formula being solved. The explosion rule can be restricted to these constants and a new unknown one. We don't need to know explicitely what this constant is since it relevant only when it clashes with other constants which can be handled by new constraints on the roots of terms. Therefore our approach will work also in this case which we do not detail any more.

A more interesting extension is to use polymorphic types, since this is the usual framework of functional languages. Since type inference is decidable in this framework provided that no *letrec* construct is allowed, one could think that our framework still holds in this case. Unfortunately, this is not the case because the notion of long eta-beta normal form no longer exists: for example X of type α with α a type variable is in normal form, but if α is instantiated by $\beta \to \beta$ then its normal form is $\lambda x : \beta.X(x)$. Therefore a straightforward rephrasing of our method will not be enough to deal with polymorphic types, and we are currently working on this subject. The decidability results that we have presented are likely to hold also in this framework since types constraints are first-order constraints which are solvable, and since only a finite number of type variable's instantiations is required.

References

[ACS94] J. Avenhaus and C.Loria-Saenz. Higher-order conditional rewriting and narrowing. In J.P. Jouannaud, editor, *Proceedings of the 1st International conference Constraints in Computational Logics*, number 845 in Lecture Notes in Computer Science, pages 269–284. Springer-Verlag, 1994.

[BT88] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 82–90, 1988.

[CL89] H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7:371–425, 1989.

[Gol81] D. Goldfarb. The undecidability of the second order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[HS86] J. Roger Hindley and Johnathan P. Seldin. *Introduction to Combinators and Lambda-calculus.* Cambridge University, 1986.

[JO91] J.P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings 6th IEEE Symposium on Logic in Computer Science, Amsterdam (The Netherlands)*, pages 350–361, 1991.

[Lug94a] Denis Lugiez. Decidable and undecidable case of higher-order disunification. Technical report, number 267, CRIN, 1994.

[Lug94b] Denis Lugiez. Higher-order disunification: some decidable cases. In J.P. Jouannaud, editor, *Proceedings of the* 1^{st} *International conference Constraints in Computational Logics*, number 845 in Lecture Notes in Computer Science, pages 121–135. Springer-Verlag, 1994.

[Mil91] D. Miller. A logic programming language with lambda abstraction, function variables and simple unification. In P.Schroeder-Heister, editor, *Extension of Logic Programming*, volume 475 of *Lecture Notes in Computer Science*, pages 253–281. Springer-Verlag, 1991.

[Mil92] D. Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.

[Nip91] T. Nipkow. Higher-order critical pairs. In *Proceedings 6th IEEE Symposium on Logic in Computer Science, Amsterdam (The Netherlands)*, pages 342–349, 1991.

[NQ91] T. Nipkow and Z. Qian. Modular higher-order E-unification. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 200–214. Springer-Verlag, 1991.

[Pre94a] C. Prehofer. Solving higher-order equations. In *Proceedings of the* 9^{th} *Symp. Logic in Computer Science*, pages 507–516. IEEE, 1994.

[Pre94b] Christian Prehofer. Decidable higher-order unification problems. In *Automated Deduction: CADE-12 - Proc. of the 12th International Conference on Automated Deduction*, 1994. To appear.

[Wol91] D. Wolfram. Rewriting and equational unification: the higher-order case. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 25–36. Springer-Verlag, April 1991.

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)

Unité de recherche INRIA Rennes - IRISA, Campus universitaire de Beaulieu 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes - 46, avenue Félix Viallet - 38031 Grenoble Cedex 1 (France)
Unité de recherche INRIA Rocquencourt - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis - 2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

ISSN 0249 - 6399