Linear Second-Order Unification*

Jordi Levy

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
http://www-lsi.upc.es/~jlevy

Abstract. We present a new class of second-order unification problems, which we have called *linear*. We deal with completely general second-order typed unification problems, but we restrict the set of unifiers under consideration: they instantiate free variables by linear terms, i.e. terms where any λ-abstractions bind one and only one occurrence of a bound variable. Linear second-order unification properly extends *context unification* studied by Comon and Schmidt-Schauß. We describe a sound and complete procedure for this class of unification problems and we prove termination for three different subcases of them. One of these subcases is obtained requiring Comon's condition, another corresponds to Schmidt-Schauß's condition, (both studied previously for the case of context unification, and applied here to a larger class of problems), and the third one is original, namely that free variables occur at most twice.

1 Introduction

It is well-known that second-order unification is undecidable. However, it is still possible to obtain decidability for some restricted, although practical, classes of problems. That is the case of Miller's *higher-order patterns*. More recently, Comon and Schmidt-Schauß have proposed a new class of second-order unification problems, called *context unification*. They have proved decidability of two different subclasses of these problems, although it is not known if the whole class is decidable. In this paper we extend this class of problems to what we call *linear second-order unification*.

Historically, Robinson and Guard were the first who studied the higher-order unification problem in the sixties. In fact, a student of Guard (Gould [4]), was the first who found a complete second-order matching algorithm. Most of the results of second-order and higher-order unification problems were proved during the seventies. Pietrzykowski [15] described a complete second-order unification procedure, that was later extended to the higher-order case [7], and Huet [6] defined *preunification* (lazy unification) and found a non-redundant procedure for it. Most of the negative results were also discovered during this decade. Independently, Lucchesi [11] and Huet [5] showed that third-order unification is not decidable, later Goldfarb [3] showed that second-order unification is not decidable either. In the nineties, Miller [13] found the first class of decidable higher-order

* This work was partially supported by the ESPRIT Basic Research Action CCL and the project DISCOR (TIC 94-0847-C02-01) funded by the CICYT

unification problems, named *higher-order pattern* unification. Such patterns were used by Nipkow [14] to propose a notion of higher-order rewriting, which lead to quite some further research [16, 12, 10].

This paper deals with another class of higher-order unification problems that has caught the attention of some researchers recently, the so called *context unification* problems. This is an extension of first-order term unification where variables may denote not only (first-order) terms, but also contexts —terms with a *hole* or *distinguished position*—. Comon [1] studied these problems to solve membership constraints. He proved that context unification is decidable, when any occurrence of the same context variable is always applied to the same term. Schmidt-Schauß [17] also studies the same problem, however he is interested in reducing the problem of unification modulo distributivity to a subset of such context unification problems. He proves that context unification is decidable when terms are *stratified.* Hence, stratified means that the string of second-order variables we find going from the root of a term to any occurrence of the same variable, is always the same. Finally, our interest on such problem comes from the *extended critical pair* problem. This problem arises when trying to apply rewriting techniques to automated deduction in monotonic order theories, using *bi-rewriting systems* [8, 9].

However, we deal with an extension of the context unification problem, named *linear second-order unification.* The generalization comes from two facts:

(i) we consider second-order terms (instead of first-order terms), thus expressions may contain λ-bindings, and

(ii) second-order context variables are not restricted to be unary, they may denote (second-order) terms with more than a *hole* or distinguished positions. In other words, second-order variables denote linear second-order terms. Hence, a *linear term* is a term whose normal form only contains λ-abstractions where the bound variable occurs once, and only once.

This generalization is motivated by the following problem. The unification problem $F(a) \stackrel{?}{=} G(f(a))$ has two minimum linear second-order unifiers:

$$\sigma_1 = [F \mapsto \lambda x \,.\, G(f(x))]$$
$$\sigma_2 = [F \mapsto \lambda x \,.\, H(x, f(a))][G \mapsto \lambda x \,.\, H(a, x)]$$

However, if we restrict ourselves to unary second-order variables, we can not represent the second minimum unifier. Context unification is infinitary for most of the problems.

This paper is structured as follows. Linear second-order unification problems are defined in section 2. Section 3 describes a linear second-order unification procedure which is proved to be sound and complete in section 4. Then, we prove decidability of such unification problems in three cases. Firstly, when no free variable occurs more than twice (section 5), secondly when a variable is applied to the same term in all its occurrences (section 6), and thirdly when we deal with *stratified* terms (section 7). Some of the proofs are long and quite complex, therefore in some cases we have included only a sketch of them.

2 Linear Second-Order Unification

In this section we define the linear second-order (LSO) typed λ-calculus, LSO-unification, and we prove some of its main properties.

We are concerned with a set of types $\mathcal{T} = \bigcup_{n\geq 1} \mathcal{T}^n$, a set of second-order typed variables $\mathcal{V} = \bigcup_{\tau\in\mathcal{T}^2} \mathcal{V}_\tau$, and a set of third-order typed constants $\mathcal{F} = \bigcup_{\tau\in\mathcal{T}^3} \mathcal{F}_\tau$ The inference rules defining well-typed linear-terms are the following ones.

$$\frac{\{x \in \mathcal{V}_\tau\}}{x:\tau} \qquad \frac{\{c \in \mathcal{F}_\tau\}}{c:\tau} \qquad \frac{x:\tau \qquad t:\tau' \quad \{x \text{ occurs once in } t\}}{\lambda x\,.\,t : \tau \to \tau'} \qquad \frac{t:\tau\to\tau' \qquad u:\tau}{t(u):\tau'}$$

We say that t is a *linear second-order (LSO) term* if $t : \tau$ may be inferred from these rules and $\tau \in \mathcal{T}^2$. Other concepts commonly used in λ-calculus, as free variables $\mathcal{FV}$, bound variables $\mathcal{BV}$, etc. will be used throughout without previous definition. Like in the simply second-order typed λ-calculus, we also consider the β and η equations $(\lambda x\,.\,t)(u) =_\beta t[x \mapsto u]$ and $\lambda x\,.\,t(x) =_\eta t$. Notice that the side condition $x \notin \mathcal{FV}(t)$ is not necessary in the η-equation because, if $\lambda x\,.\,t(x)$ is well-typed, then this condition is ensured. Notice also that these equations, used as rewriting rules, transform well-typed linear terms into linear terms with the same type. The η-long β-normal form of a term t is denoted by $t|_{\beta\eta}$ and has the form $\lambda x_1 \ldots x_n\,.\,a(t_1,\ldots,t_m)$, where a is called the *head* of the term and can be either a free variable, a bound variable or a constant, $a(t_1,\ldots,t_m)$ is a first-order typed term, and $t_1 \ldots t_m$ are also second-order normal terms. Moreover, if a is a bound variable then $m = 0$. We require linearity to prove the following lemma.

Lemma 1. *For any pair of LSO-terms t and u, if $t =_{\beta\eta} u$ then $\mathcal{FV}(t) = \mathcal{FV}(u)$.*

We consider any kind of second-order unification problem, and we only restrict the set of possible unifiers.

Definition 2. A *second-order unification* (SOU) *problem* is a finite set of pairs $\{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\}$, where t_i and u_i are —not necessarily linear— second-order typed terms, and have the same type, for any $i \in [1..n]$.

A *LSO-substitution* σ is an idempotent and type preserving morphism between terms such that its *domain*, defined by $Dom(\sigma) \stackrel{def}{=} \{X \mid X \neq \sigma(X)\}$, is finite and for any $X \in Dom(\sigma)$, $\sigma(X)$ is a $\beta\eta$-normal LSO-term.

As usual, given a pair of substitutions σ, ρ, we say that $\sigma \preceq \rho$ if there exists a substitution τ such that $\rho = \tau\circ\sigma$. We say that σ is a *minimum substitution* (w.r.t. a set of them) if for any other substitution ρ of the set we have $\sigma \preceq \rho$ or $\rho \not\preceq \sigma$. The set of *free variables of a substitution* is defined as $\mathcal{FV}(\sigma) = \bigcup_{X\in Dom(\sigma)} \mathcal{FV}(\sigma(X))$.

Lemma 3. *If the composition of two LSO-substitutions is idempotent, then it is also an LSO-substitution. (Composition preserves linearity).*
Given two terms t and u, there are finite many LSO-substitutions σ satisfying $t =_{\beta\eta} \sigma(u)$ and $Dom(\sigma) \subseteq \mathcal{FV}(u)$.[2]

[2] Notice that this is not true for non-linear substitutions: there are infinitely many substitutions $\sigma = [F \mapsto \lambda x\,.\,a, X \mapsto t]$ satisfying $\sigma(F(X)) = a$.

Second part of this lemma suggests us that LSO-substitutions are specially adequate for basing on it a definition of second-order rewriting systems.

3 A Linear Second-Order Unification Procedure

In this section we describe a sound and complete procedure for the LSO-unification problem. This procedure is non-terminating in general, but remind that decidability of LSO-unification and context unification are open problems. In following sections we will use modified versions of this procedure to prove decidability of the LSO-unification problem in some restricted cases. Thus, those modified versions are terminating, although they are complete only for some subset of problems.

In the description of the procedure we use a compact notation based on lists of indexes and indexed lists of indexes. We denote lists of indexes by capital letters P, Q, R,... For any list of indexes $P = \{p_1, \ldots, p_n\}$, the expression $a(\overline{b_P})$ denotes $a(b_{p_1}, \ldots, b_{p_n})$, and for any P-indexed list of indexes $Q_P = \{Q_{p_1}, \ldots, Q_{p_n}\} = \{\{q_1^1, \ldots, q_1^{m_1}\}, \ldots, \{q_n^1, \ldots, q_n^{m_n}\}\}$ the expression $a(\overline{b_P(\overline{c_{Q_P}})})$ denotes $a(b_{p_1}(c_{q_1^1} \ldots c_{q_1^{m_1}}), \ldots, b_{p_n}(c_{q_n^1} \ldots c_{q_n^{m_n}}))$. As usual, small letters like p denotes correlative lists of indexes $[1..p]$, so $a(\overline{b_p})$ denotes $a(b_1, \ldots, b_p)$.

We also use the notation on transformations introduced by Gallier and Snyder [2] for describing unification processes. Any state of the process is represented by a pair $\langle S, \sigma \rangle$ where $S = \{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\}$ is the set of unification problems still to be solved and σ is the substitution computed until that moment, i.e. the substitution leading from the initial problem to the actual one. The initial state is $\langle S_0, Id \rangle$. The procedure is described by means of transformation rules on states $\langle S, \sigma \rangle \Rightarrow \langle S', \sigma' \rangle$. If the initial state can be transformed into a normal form where the unification problem is empty $\langle S_0, Id \rangle \Rightarrow^* \langle \emptyset, \sigma \rangle$ then σ is a solution –minimal unifier– of the original unification problem S_0.

We suppose that any term of the initial state is in $\beta\eta$-normal form, and that after applying any transformation rule the new unification problem and the new substitution are also $\beta\eta$-normalized. Therefore, we can suppose that any pair $t \stackrel{?}{=} u \in S$ has the form $\lambda\overline{x_N}.\, a(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}.\, b(\overline{u_Q})$ because, if t and u have the same type, then they have the same number of more external λ-bindings. We give the same name to these bound variables using α-conversion.

Definition 4. All *problem transformation rules* have the form:

$$\langle S \cup \{t \stackrel{?}{=} u\}, \sigma \rangle \Rightarrow \langle \rho(S \cup R), \rho \circ \sigma \rangle$$

where, for each rule, the transformation $t \stackrel{?}{=} u \Rightarrow R$ and the LSO-substitution ρ are defined as follows.

1. *Rigid-rigid rule (or Simplification rule).* If a is a constant, or a bound variable then
$$\lambda\overline{x_N}.\, a(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}.\, a(\overline{u_P}) \;\Rightarrow\; \bigcup_{i \in P} \{\lambda\overline{x_N}.\, t_i \stackrel{?}{=} \lambda\overline{x_N}.\, u_i\}$$
$$\rho = Id$$

2. *Imitation rule.* If a is a constant[3] and F is a free variable, and $\{R_i\}_{i \in P}$ is a

[3] Notice that if a is a bound variable and $\lambda\overline{x_N}.\, a(\overline{t_P})$ is a LSO term, then $P = \emptyset$ and this situation is captured by the projection rule.

P-indexed family of disjoint lists of indexes satisfying[4] $\bigcup_{i\in P} R_i = Q$, then

$$\lambda\overline{x_N}\,.\,a(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,F(\overline{u_Q}) \;\Rightarrow\; \bigcup_{i\in P}\{\lambda\overline{x_N}\,.\,t_i \stackrel{?}{=} \lambda\overline{x_N}\,.\,F'_i(\overline{u_{R_i}})\}$$
$$\rho = [F \mapsto \lambda\overline{y_Q}\,.\,a(\overline{F'_P(\overline{y_{R_P}})})]$$

where $\{F'_j\}_{j\in Q}$ are fresh free variables of the appropriate type that can be inferred from the context.

3. *Projection rule.* If a is a constant or a bound variable and F is a free variable, and $a(\overline{t_p})$ and u have the same type, then

$$\lambda\overline{x_N}\,.\,a(\overline{t_p}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,F(u) \;\Rightarrow\; \{\lambda\overline{x_N}\,.\,a(\overline{t_p}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,u\}$$
$$\rho = [F \mapsto \lambda y\,.\,y]$$

4. *Flexible-flexible rule with equal heads (or Simplification rule).* If F is a free variable, then

$$\lambda\overline{x_N}\,.\,F(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,F(\overline{u_P}) \;\Rightarrow\; \bigcup_{i\in P}\{\lambda\overline{x_N}\,.\,t_i \stackrel{?}{=} \lambda\overline{x_N}\,.\,u_i\}$$
$$\rho = Id$$

5. *Flexible-flexible rule with distinct-heads (or Distinct-heads rule).* If F and G are free distinct variables, $P' \subseteq P$ and $Q' \subseteq Q$ are two lists of indexes, and $\{R_j\}_{j\in Q'}$ is a Q'-indexed and $\{S_i\}_{i\in P'}$ a P'-indexed family of disjoint lists of indexes satisfying R_j and P' are disjoint, S_i and Q' are also disjoint, $\left(\bigcup_{j\in Q'} R_j\right) \cup P' = P$ and $\left(\bigcup_{i\in P'} S_i\right) \cup Q' = Q$, then

$$\lambda\overline{x_N}\,.\,F(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,G(\overline{u_Q}) \;\Rightarrow\; \bigcup_{j\in Q'}\{\lambda\overline{x_N}\,.\,F'_j(\overline{t_{R_j}}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,u_j\} \cup \bigcup_{i\in P'}\{\lambda\overline{x_N}\,.\,t_i \stackrel{?}{=} \lambda\overline{x_N}\,.\,G'_i(\overline{u_{S_i}})\}$$
$$\rho = [F \mapsto \lambda\overline{y_P}\,.\,H'(\overline{F'_{Q'}(\overline{y_{R_{Q'}}})}, \overline{y_{P'}})][G \mapsto \lambda\overline{z_Q}\,.\,H'(\overline{z_{Q'}}, \overline{G'_{P'}(\overline{z_{S_{P'}}})})]$$

where H', $\{G'_i\}_{i\in P'}$ and $\{F'_j\}_{j\in Q'}$ are fresh free variables of the appropriate types. Notice that if $R_j = \emptyset$ then F'_j is a first-order typed variable (resp. for S_i and G'_i).

Compared with the general second-order unification procedure [7], we avoid the use of the prolific *elimination* and *iteration* rules, which always compromise the termination of their procedure. It does not mean that our procedure terminates, but it avoids many redundant states. Moreover, in contrast to Jensen and Pietrzykowski's procedure, our procedure only computes minimal unifiers.

To ensure that the final substitution is linear, we only instantiate free variables by linear terms. This means that when we apply imitation rule like $[F \mapsto \lambda\overline{y_Q}\,.\,a(\overline{F'_P(\overline{y_{R_P}})})]$, we have to make sure that the Q arguments of F are distributed among the F'_P new variables. This imposes the restrictions $\{R_i\}_{i\in Q}$ are disjoint lists of indexes and $\bigcup_{i\in P} R_i = Q$. Notice that this condition ensures that $\lambda\overline{y_Q}\,.\,a(\overline{F'_P(\overline{y_{R_P}})})$ is a linear term.

[4] Union and comparison of lists of indexes is computed without considering their order.

4 Soundness and Completeness

In this section we schematize the proofs for soundness and completeness of our procedure.

Theorem 5. Soundness. *For any second-order unification problem S_0, if there exists a derivation $\langle S_0, Id\rangle \Rightarrow^* \langle \emptyset, \sigma\rangle$, then σ is a minimal linear second-order unifier of S_0.*

Simplification steps preserve the set of minimum unifiers of an unification problem. Imitation, projection and distinct-heads steps may be considered as the composition of an instantiation and a simplification step, where the substitution instantiates a free variable by a linear term. For instance, the imitation rule may be decomposed as follows.

$$\begin{array}{rll} \lambda\overline{x_N}\,.\,a(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,F(\overline{u_Q}) \Rightarrow & \lambda\overline{x_N}\,.\,a(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,a(\overline{F'_P(\overline{u_{R_P}})}) & \text{Instantiation} \\ \Rightarrow & \bigcup_{i\in P}\left\{t_i \stackrel{?}{=} F'_i(\overline{u_{R_i}})\right\} & \text{Simplification} \end{array}$$

Now, taking into account that composition of linear substitutions is also a linear substitution (see lemma 3, taking into account that we only introduce fresh variables, which ensures idempotence), it is not difficult to prove that the final substitution is really a unifier.

Theorem 6. Completeness. *If σ is a minimum linear second-order unifier of the second-order unification problem S_0, then there exists a transformation sequence $\langle S_0, Id\rangle \Rightarrow^* \langle \emptyset, \sigma\rangle$.*

We can prove easily that whenever exists a minimum unifier σ of a unification problem S_0, we can generate a sequence of transformations:

$$\langle S_0, Id\rangle \Rightarrow \langle S_1, \sigma_1\rangle \Rightarrow \cdots \Rightarrow \langle S_n, \sigma_n\rangle \Rightarrow \cdots$$

satisfying $\sigma_n \preceq \sigma$ for any $n \geq 0$. There are two possibilities, either this sequence is infinite or its last state is $\langle \emptyset, \sigma\rangle$.

Now, we can prove that no such infinite sequences exist. In first-order we can define the *size of a substitution* as $size(\sigma) = \sum_{X\in Dom(\sigma)} \sigma(X)$, where the size of a term is the number of applications it contains. Then, we can prove that each new instantiation increases this size, i.e. if $\sigma \preceq \rho$ then $size(\sigma) \leq size(\rho)$. This is not true in second-order because we can instantiate a second-order variable by a function which disregards its parameters[5]. This is not true in linear second-order unification either, due to the projection rule. Fortunately, in our case we can avoid this problem defining the size of a substitution w.r.t. another substitution.

Definition 7. The *size of a term t* w.r.t. a substitution ρ is defined as follows

$$size(\lambda\overline{x_P}\,.\,a(\overline{t_Q}), \rho) = \sum_{q\in Q} size(t_q, \rho) + \begin{cases} 0 & \text{if } a \text{ is a free variable and } \rho(a) = \lambda x\,.\,x \\ \#Q & \text{otherwise} \end{cases}$$

[5] In this case we could define the size of a term as the number of applications contained in its $\beta\eta$-normal form.

where $\#Q$ is the cardinality of Q, and the *size of a LSO-substitutions* is defined as follows

$$size(\sigma, \rho) \stackrel{def}{=} \sum_{X \in Dom(\sigma)} size(\sigma(X), \rho)$$

Lemma 8. *For any LSO-term t and LSO-substitutions σ, ρ and τ we have*

$$size(t, \tau \circ \rho) \leq size(\rho(t), \tau)$$
$$size(\sigma, \tau \circ \rho) \leq size(\rho \circ \sigma, \tau)$$

Although projection steps do not increase the size of the unifier computed till that moment, they decrease the free arity of the problem, defined as $arity(S) = \sum_{X \in \mathcal{FV}(S)} arity(X)$ where as usual the arity of a variable X is the maximal number of parameters it admits.[6]

Finally, we can define the size of a state $\langle S_n, \sigma_n \rangle$, of our particular sequence of transformations, w.r.t. the unifier σ as a triplet where first and second component are integers and the third component is a multiset of integers:

$$size(\langle S_n, \sigma_n \rangle) = \min_{\{\tau \mid \sigma = \tau \circ \sigma_n\}} \langle arity(S_n), size(\sigma, Id) - size(\sigma_n, \tau), \bigcup_{t \stackrel{?}{=} u \in S_n} \{size(t, \tau), size(u, \tau)\} \rangle$$

Now we have to prove that if $\langle S_n, \sigma_n \rangle \Rightarrow \langle S_{n+1}, \sigma_{n+1} \rangle$ then $size(\langle S_n, \sigma_n \rangle) > size(\langle S_{n+1}, \sigma_{n+1} \rangle)$, where $>$ is a lexicographic ordering, and third components of the triplet are compared using the usual multiset ordering. We may check that any projection step strictly reduces the free arity of the problem, whereas any other transformation does not change it. For imitation steps and flexible-flexible steps the result depends on the size of the variable (or variables) being instantiated before and after applying the transformation step. If it does not increase, then the size of the substitution remains unchanged but the size of the problem decreases. If it increases, although the size of the problem increases, the size of the substitution also increases (the difference $size(\sigma, Id) - size(\sigma_n, \tau_n)$ decreases). For simplification steps, $\sigma_n = \sigma_{n+1}$, the substitution does not change but the size of the problem decreases (maybe its free arity, too).

Notice that this result proves the completeness of the unification procedure, but not its termination and, therefore, not the decidability of the unification problem. The function *size* could be used to prove the termination of the procedure if we would be able to fix an upper bound $size(\sigma, Id) \leq MAX$ for the size of one of the minimum unifier of a unification problem, if it has any.

5 A Termination Result

In the following we will prove that our procedure always finishes for a very useful case: if no free variable occurs more than twice in an unification problem. This fact is related with the termination of the *naive* string unification procedure when variables occurs at most twice [18].

Theorem 9. Termination. *If no free variable occurs more than twice in a linear second-order unification problem, then this unification problem is decidable.*

[6] If $X : \tau_1 \to \ldots \to \tau_n \to \tau$ and τ is a first-order type, then $arity(X) = n$.

To prove this theorem we define the following *size* function, where we suppose that any term is normalized previously to compute its size.

$$size(\lambda x_1 \ldots x_n \,.\, a(t_1, \ldots, t_p)) = p + \sum_{i=1}^{p} size(t_i)$$
$$size(\{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\}) = \sum_{i=1}^{n} size(u_i) + size(t_i)$$

We prove now that if $\langle S, \sigma\rangle \Rightarrow \langle S', \rho \circ \sigma\rangle$ and any free variable appears at most twice in S then $size(S') \leq size(S)$ and any free variable also appears at most twice in S'. There are five cases. Here we only show the case of flexible-flexible steps with distinct-heads.

$$\lambda\overline{x_N} \,.\, F(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N} \,.\, G(\overline{u_Q}) \;\Rightarrow\; \bigcup_{j \in Q'} \{\lambda\overline{x_N} \,.\, F'_j(\overline{t_{R_j}}) \stackrel{?}{=} \lambda\overline{x_N} \,.\, u_j\} \cup \bigcup_{i \in P'} \{\lambda\overline{x_N} \,.\, t_i \stackrel{?}{=} \lambda\overline{x_N} \,.\, G'_i(\overline{u_{S_i}})\}$$
$$\rho = [F \mapsto \lambda\overline{y_P} \,.\, H'(\overline{F'_{Q'}(y_{R_{Q'}})}, \overline{y_{P'}})][G \mapsto \lambda\overline{z_Q} \,.\, H'(\overline{z_{Q'}}, \overline{G'_{P'}(z_{S_{P'}})})]$$

The size of the problem decreases in $\#P + \#Q - \sum_{i \in P'} \#S_i - \sum_{j \in Q'} \#R_j = \#P' + \#Q'$ and is increased in $\#Q'$ for any instantiation of F and in $\#P'$ for any instantiation of G. Therefore, in the worst case, the size of the problem remains equal. It is also easy to see that in the worst case we introduce two occurrences of each one of the fresh variables H', $\{F'_i\}$ and $\{G'_j\}$.

Finally, taking into account that for any finite signature and given size there are finitely many unification problems, it easy to see that it is enough to control loops to ensure the termination of the procedure described in the previous sections.

6 Extension of Comon's Decidability Result

To extend Comon's condition [1] to linear second-order unification we have to consider non-unary free variables and λ-bindings. The later makes necessary to introduce λ-equivalent terms.

Definition 10. Given two LSO-terms $\lambda x_1, \ldots, x_n \,.\, t$ and $\lambda y_1, \ldots, y_m \,.\, u$, we say that they are *$\lambda$-equivalent*, noted $\lambda x_1, \ldots, x_n \,.\, t =_\lambda \lambda y_1, \ldots, y_m \,.\, u$, if $\lambda x_1, \ldots, x_n \,.\, t|_{\beta\eta} =_\alpha \lambda z_1, \ldots, z_{n-m}, y_1, \ldots, y_m \,.\, u|_{\beta\eta}$, where we suppose that $n \geq m$ and $\{z_1, \ldots, z_n\} \not\subseteq \mathcal{FV}(u)$.

Given a normalized unification problem S, we say that it satisfies the *extended Comon's condition* if, for any pair of occurrences of a free variable F, they are in two subterms $F(t_1, \ldots, t_n)$ and $F(u_1, \ldots, u_n)$ satisfying $\lambda\overline{x_N} \,.\, F(t_1, \ldots, t_n) =_\lambda \lambda\overline{y_M} \,.\, F(u_1, \ldots, u_n)$, where $\overline{x_N}$ and $\overline{y_M}$ are respectively the sequence of λ-bindings above these subterms.

Notice that λ-equivalence is an extension of α-conversion between $\beta\eta$-normal terms. We can prove that Comon's condition is also enough to ensures decidability in the more general linear second-order unification setting.

Theorem 11. Termination II. *Any linear second-order unification problem satisfying the extended Comon's condition is decidable and finitary.*

To ensure the termination of our unification procedure we have to apply a kind of problem normalization before any transformation. This is equivalent to working with *generalized equations* like it is done in A-unification [18]. We deal

with sequences of equalities $t_1 \stackrel{?}{=} \cdots \stackrel{?}{=} t_n$ and we concatenate any pair of sequence sharing two λ-equivalent terms. Thus, apart from the normalization procedure described in section 2, we will apply the following *concatenation rule*:

$\{t_1 \stackrel{?}{=} \cdots \stackrel{?}{=} t_n\ ,\ u_1 \stackrel{?}{=} \cdots \stackrel{?}{=} u_m\} \cup S \Rightarrow \{t_1 \stackrel{?}{=} \cdots t_n \stackrel{?}{=} \lambda\overline{x_N}\,.\,u_2 \stackrel{?}{=} \cdots \stackrel{?}{=} \lambda\overline{x_N}\,.\,u_m\} \cup S$

whenever $t_1 =_\alpha \lambda\overline{x_N}\,.\,u_1$, i.e. whenever $t_1 =_\lambda u_1$.

This normalization procedure and Comon's condition ensures that no free variable F occurs in the outermost head of more than one term, although F may occur below the head of other terms.

Definition 12. Given a unification problem S, let $\approx_S$ be the reflexive-transitive closure of the relation defined by: if $t =_\lambda u$ or $t \stackrel{?}{=} u \in S$ then $t \approx_S u$, for any pair of terms t and u. Consider the finite set of $\approx_S$-equivalence classes of terms containing a term of S or a subterms of one of them.

Let $\succ_S$ be the relation, on this set of classes of terms, defined by the transitive closure of the strict subterm relation: $[\lambda\overline{x_N}\,.\,a(\overline{t_P})] \succ_S [\lambda\overline{x_N}\,.\,t_i]$ for any $i \in P$ and any term $\lambda\overline{x_N}\,.\,a(\overline{t_P})$ of S.

Since there are finitely many classes of terms, $\succ_S$ is either cycling or well-founded. Definitions of $\approx_S$ and $\succ_S$ are invariants for concatenation of generalized equations (normalization of unification problems), moreover any two terms belonging to the same generalized equation are $\approx$-equivalent. Finally, if σ is a unifier of S and $t \approx_S u$ then $\sigma(t)$ and $\sigma(u)$ have the same size (number of applications), and, if $t \succ_S u$ then the size of $\sigma(t)$ is greater or equal to the size of $\sigma(u)$. This fact allows us to prove the following lemma, which characterizes two possible situations we have to consider.

Lemma 13. *For any unification problem S, either a) the relation $\succ_S$ is irreflexive and therefore well-founded, or b) any unifier σ of S satisfies $\sigma(F) = \lambda x\,.\,x$ for some free variable F.*

In case b) of the lemma the only chance to get a solution is applying —after maybe some simplification steps— the projection rule to a free variable involved in a $\succ$ cycle, i.e. if we have $[\lambda\overline{x_N}\,.\,F(t)] \succ_S [\lambda\overline{x_N}\,.\,t] \succ_S \cdots \succ_S [\lambda\overline{x_N}\,.\,F(t)]$, then any unifier σ of S, if there is any[7], satisfies $\sigma(F) = \lambda x\,.\,x$. As we know, projection rule strictly decreases the free arity of the problem, whereas any other transformation rule preserves it. This allows us to define a well-founded lexicographic ordering on unification problems where first component is free arity.

In case a) if S satisfies Comon's condition and $\succ_S$ is not cycling, we can prove that —after applying some simplification steps— there exists a free variable occurring in the head of a term and no occurring elsewhere. We will try to find an instantiation for this variable. Suppose we decide to apply imitation rule, and such variable is F. This imitation rule may be extended to deal with generalized

[7] Some unification problems, like $\{\lambda x\,.\,F(x) \stackrel{?}{=} \lambda x\,.\,a(G(x)), \lambda y\,.\,G(y) \stackrel{?}{=} \lambda y\,.\,b(F(y))\}$, defining a cycling $\succ_S$ relation, have no solutions, even if we try instantiate F or G by $\lambda x\,.\,x$.

equations as follows.

$$\lambda\overline{x_N}\,.\,a(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}\,.\,F(\overline{u_Q}) \stackrel{?}{=} v_1 \stackrel{?}{=} \ldots \stackrel{?}{=} v_r$$
$$\Downarrow$$
$$\lambda\overline{x_N}\,.\,a(\overline{t_P}) \stackrel{?}{=} v_1 \stackrel{?}{=} \ldots \stackrel{?}{=} v_r \;\cup\; \bigcup_{i\in P}\left\{\lambda\overline{x_N}\,.\,t_i \stackrel{?}{=} \lambda\overline{x_N}\,.\,F_i'(\overline{u_{R_i}})\right\}$$

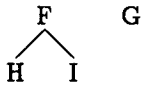
Most important thing is noticing that no instantiation is necessary on such *transformation step* because F occurs only once. After normalizing the new problem, new equations $\lambda\overline{x_N}\,.\,t_i \stackrel{?}{=} \lambda\overline{x_N}\,.\,F_i'(\overline{u_{R_i}})$ may be appended to other generalized equations, however this is not a problem because $\succ_S$ is invariant for such process. Moreover, although we have modified the unification problem, the (finite) set of $\approx$-equivalence classes of terms does not change (or it has decreassed if $v_1 \stackrel{?}{=} \cdots v_r$ is empty). We have introduced two different kinds of new terms: $\lambda\overline{x_N}\,.\,t_i$ and $\lambda\overline{x_N}\,.\,F_i'(\overline{u_{R_i}})$, but both of them belong to already taken into account classes of equivalences. Furthermore, subterms of these new terms are also subterms of other old terms. However, the relation $\succ_S$ has changed, for any transformation $S \Rightarrow S'$ we have $\succ_S \subseteq \succ_{S'}$. (Notice that now $[\lambda\overline{x_N}\,.\,t_i] = [\lambda\overline{x_N}\,.\,F_i'(\overline{u_{R_i}})] \succ_{S'} [\lambda\overline{x_N}\,.\,u_{R_i}]$).

Summing up, term $\lambda\overline{x_N}\,.\,F(\overline{u_Q})$ has been replaced (from a generalized equation) by several $\succ_S$-strictly smaller terms (introduced in other generalized equations), and $\succ_S$ has been replaced by a bigger relation $\succ_{S'}$ over the same set of classes of terms. Something similar applies to the distinct-heads rule. This fact and the reduction of free arity due to projection rule may be used to define a well-founded ordering on unification problems, and conclude that linear second-order unification under Comon's restriction is finitary.

7 Decidability Result for Stratified Terms

Schmidt-Schauß [17] describes another class of decidable context unification problems, called *stratified second-order unification*. Hence, a stratified second-order term is a second-order typed term where the *string* or sequence of second-order variables we find going from the root of the term to an occurrence of a second-order variable is always the same, for any occurrence of this variable. We can represent these stratification of second-order variables as a set of trees (i.e. as a *forest*). This is an example of stratified term and its corresponding forest:

$$f\big(F(H(b))\;,\;G(a)\;,\;F(g(H(b),I(a)))\big)$$



The restriction on the *string* of each variable is extended to all the occurrences of such variable in a unification problem. This introduces a new problem, because such restriction neither is stable for instantiations nor for simplifications. For instance, term $F(G(a))$ becomes non-stratified after applying substitution $[F \mapsto \lambda x\,.\,G(x)]$, and the stratified unification problem $\left\{F(G(a)) \stackrel{?}{=} F(H(a)), F(G(a)) \stackrel{?}{=} f(a)\right\}$ is transformed into the non-stratified unification problem $\left\{G(a) \stackrel{?}{=} H(a), F(G(a)) \stackrel{?}{=} f(a)\right\}$ after simplification. The second problem suggests us to generalize the definition of stratified problem. Moreover, we have also to generalize it to consider non-unary free variables.

Definition 14. Given a set of variables $\mathcal{V}$, a *stratified forest* is a set of trees W

where, for any n-ary variable $F \in \mathcal{V}$, we have n distinct nodes $F^1, \ldots, F^n$ such that, either all them are roots of some tree, or all them are sons of the same father. A $\beta\eta$-normal term t is a *stratified term* w.r.t. (a position $p = [\,]$ of) a stratified forest W if for any free variable F of t we have:

(i) if the corresponding nodes $F^1, \ldots, F^n$ are roots of trees of W, then no free occurrence of F is below any other free variable occurrence of t,

(ii) if the corresponding nodes $F^1, \ldots, F^n$ are sons of another node G^i of W, then any free occurrence of F is in the ith argument of an occurrence of G and there is no other free variable occurrence between these occurrences of F and G.

We define a *position p of a forest W*, denoted by $W|_p$, recursively as follows.

(i) $\{t_1, \ldots, t_n\}|_{[\,]} \stackrel{def}{=} \{t_1, \ldots, t_n\}$

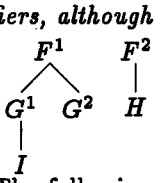
(ii) $\{t_1, \ldots, t_n\}|_{[p_1, \ldots, p_m]} \stackrel{def}{=} \{u_1, \ldots, u_p\}|_{[p_2, \ldots, p_m]}$ where $p_1 \in [1..n]$ and $u_1, \ldots, u_p$ is the list of subtrees of t_{p_1}.

(Notice that, in order to define positions in a forest, we have to suppose a certain ordering on trees and subtrees).

A $\beta\eta$-normal term t is a *stratified term* w.r.t. a position p of a stratified forest W, if t is a stratified term w.r.t. the forest $W|_p$.

Given a unification problem S, we say it is a *stratified unification problem* if there exists a forest W such that for any $t \stackrel{?}{=} u \in S$, terms t and u are both stratified w.r.t. the same position p of W. For any stratified problem S there exists a unique minimum —the one which has less nodes— forest W_S, called the *associated* forest of S.

Theorem 15. Termination III. *It is decidable whether a stratified unification problem has a solution or not. Moreover, we can find the complete set of minimum unifiers, although the unification problem may be infinitary.*



On the left there is an example of stratified forest of $\mathcal{V} = \{F, G, H, I\}$, where F and G are binary variables and H and I are unary variables.

The following are examples of stratified terms w.r.t. (the position $p = [\,]$ of) this stratified forest.

$$F\big(f(G(a,b)\,,\, G(I(a),b))\,,\, H(a)\big) \qquad\qquad F(a, H(b))$$

The following are examples of stratified terms

$$G(I(a), b) \qquad\qquad f(H(a)\,,\, g(H(b))) \qquad\qquad g(I(a))$$

w.r.t. positions [1], [2] and [1, 1] respectively, of the same forest. The unification problem $\{G(I(a), b) \stackrel{?}{=} g(I(a))\}$ is not stratified because, although both terms are stratified, they are stratified w.r.t. two different positions.

Notice that any term without free variables is a stratified term w.r.t. any position of any stratified forest. However, such terms may be suppressed from any unification problem using a second-order matching algorithm.

Since for any n-ary variable F there are only n nodes $F^1, \ldots, F^n$ in the stratified forest and all them are sons of the same father or roots, we can not have a

stratified term with an free occurrence of F below any other free occurrence of the same variable F.

We will give now some results which will allow us to prove termination of a modified version of our unification procedure when we deal with stratified unification problems.

Lemma 16. *Any substitution applied by the unification procedure transforms a stratified unification problem S w.r.t. a forest W_S, into another stratified unification problem S' w.r.t. a new forest $W_{S'}$, such that $W_{S'}$ has less or as many nodes as W_S.*

Let us see how imitation rule transforms these stratified forests. We take $P = [1..p]$ and $Q = [1..q]$ for simplicity. Given a family of disjoint lists of indexes $\{R_i\}_{i\in[1..p]}$ satisfying $\bigcup_{i\in[1..p]} R_i = [1..q]$. The substitution applied by this rule is written as follows.

$$\rho = \left[F \mapsto \lambda \overline{y_q}\,.\, a\big(G_1(y_{r_1^1}, \ldots, y_{r_{n_1}^1}), \ldots, G_p(y_{r_1^p}, \ldots, y_{r_{n_p}^p})\big)\right]$$

where $R_i \stackrel{def}{=} \{r_1^i, \ldots, r_{n_i}^i\}$ and $\{G_j\}_{j\in[1..p]}$ are fresh free variables. Nodes corresponding to F have to be substituted by nodes corresponding to the G_i, in the new forest, as it is shown in figure 1. In the same figure we show also how projection rule and distinct-heads rule transform stratified forests.

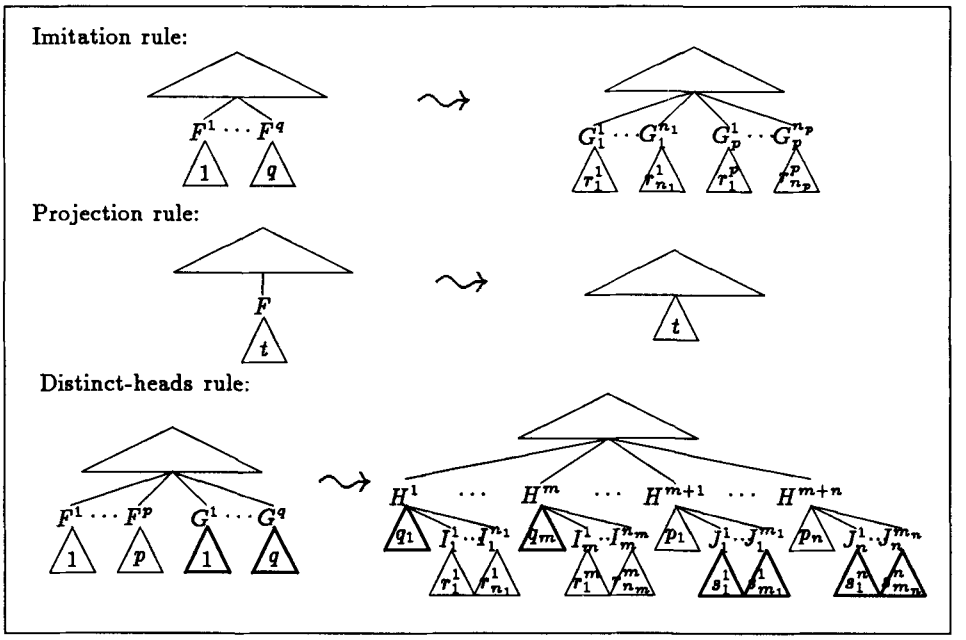


Fig. 1. Forest transformation obtained for each one of the transformation rules.

Notice that not all new G_i appear in the new forest, only the second-order typed ones, i.e. those which satisfy $R_i \neq \emptyset$.

The restriction on the family of indexes R_i ensures that $\sum_{i=1}^{p} n_i = q$ and $\bigcup_{i\in[1..p]} R_i = \{r_1^1 \ldots r_{n_1}^1 \ldots r_1^p \ldots r_{n_p}^p\}$ is a permutation of $[1..q]$, therefore the number of nodes, even the shape of the tree, do not change. Nodes $F^1, \ldots, F^q$ have been replaced by nodes G_i^j, and subtrees below them have been permuted and reallocated below new nodes.

Notice that when a subtree of W is reallocated in W' then, except in the case of projection rule, it either goes to a deeper position or remains in the same forest level. This allows to prove the following lemma.

Lemma 17. *If S is a stratified unification problem w.r.t. a position p of W, then $\rho(S)$ is a stratified unification problem w.r.t. a position p' of W' such that either*

(i) *$p = p'$, or*

(ii) *p' is shorter than p, if ρ is the substitution corresponding to a projection step,*

(iii) *p and p' have the same length, if ρ corresponds to an imitation step,*

(iv) *p' is longer or has the same length than p, if ρ corresponds to a distinct-heads step.*

This lemma suggests us to define a notion of depth level of a stratified term.

Definition 18. Term t is said to be a *n-level stratified term* w.r.t. a forest W, noted $\mathit{depth}(t, W) = n$, if it is a stratified term w.r.t. a position p of W and $\mathtt{length}(p) = n$.

We associate a multiset of integers to each stratified unification problem S w.r.t. a forest W, defined as follows

$$\mathit{depth}(S, W) = \{\mathit{Nod} - \mathit{depth}(t, W) \mid t \stackrel{?}{=} u \in S \wedge u \stackrel{?}{=} t \in S\}$$

where Nod is the number of nodes of W (i.e. the maximum depth W can reach).

After showing how imitation, projection and distinct-heads rules transform a stratified forest, let us analyze what happens with simplification rules.

Lemma 19. *If $\lambda\overline{x_N}.a(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}.a(\overline{u_P})$ are both stratified term w.r.t. a position p of a forest W, then so they are $\lambda\overline{x_N}.t_i \stackrel{?}{=} \lambda\overline{x_N}.u_i$, for any $i \in P$.*

If $\lambda\overline{x_N}.F(\overline{t_P}) \stackrel{?}{=} \lambda\overline{x_N}.F(\overline{u_P})$ are both stratified term w.r.t. a position p of a forest W, then $\lambda\overline{x_N}.t_i \stackrel{?}{=} \lambda\overline{x_N}.u_i$ are both stratified terms w.r.t. a position p' of W, where $\mathtt{length}(p') = \mathtt{length}(p) + 1$, for any $i \in P$.

If we consider each problem transformation as an instantiation followed by a simplification, then this two lemmas allow us to define an ordering on unification problems taking into account the following facts.

(i) Each projection step strictly decreases the size (number of nodes) of the associated stratified forest of a problem, whereas the rest of rules preserve this size.

(ii) Each flexible-flexible step with equal heads or distinct-heads strictly decrease the depth level of any unification problem S, whereas the rest of rules also decrease or preserve this depth level.

(iii) Each rigid-rigid step strictly decreases the size of a unification problem.

We can conclude that any non-terminating transformation sequence of a stratified unification problem contains infinitely many imitation steps. We can go a bit farther and state the following theorem.

Theorem 20. *Any infinite sequence of transformations of stratified unification problems contains an infinite subsequence of imitation steps of the form:*

$$\lambda\overline{x_N}.\,a(\overline{t_p}) \stackrel{?}{=} \lambda\overline{x_N}.\,F(\overline{u_q}) \;\Rightarrow\; \{\lambda\overline{x_N}.\,t_i \stackrel{?}{=} \lambda\overline{x_N}.\,F_i'(u_1,\ldots,u_q)\} \cup \bigcup_{j\in P\wedge j\neq i}\{\lambda\overline{x_N}.\,t_j \stackrel{?}{=} \lambda\overline{x_N}.\,F_j\}$$
$$\rho = [F \mapsto \lambda\overline{y_Q}.\,a(F_1',\cdots,F_{i-1}',F_i'(y_1,...,y_q),F_{i+1}',\cdots,F_p')]$$

for some $i \in [1..p]$ such that F occurs free in t_i, and no other free variable occurs between the root of t_i and this occurrence of F.

Once we have characterized the only type of infinite transformation sequences we can have, we substitute them by a single transformation rule without losing completeness property of the procedure. This rule will be the only source of infinitary solutions of this class of problems. Notice that all pairs involved in such infinite sequences must be treated together, so this new rule is a bit complex and we have no room here to describe it in detail. We will do that in the case of A-unification, using the parallelism existing between A-unification and linear second-order unification.

We have, in general, a set of unification pairs

$$\{F\cdot\alpha_1 \stackrel{?}{=} \beta_1\cdot F\cdot\delta_1\ ,\cdots,\ F\cdot\alpha_n \stackrel{?}{=} \beta_n\cdot F\cdot\delta_n\}$$

where greek letters represent sequences of symbols and F does not occur in them. Moreover, β_i does not contain any variable occurrence. There are two kinds of solution for F.

If there exists a decomposition of $\beta_i = \gamma_i\cdot\eta_i$ and there exists an exponent n_i such that $(\beta_i)^{n_i}\cdot\gamma_i = \omega_1$ for any $i\in[1..n]$ then substitution $[F\mapsto\omega_1]$ transforms the previous problem into

$$\{\alpha_1 \stackrel{?}{=} \eta_1\cdot\gamma_1\cdot\delta_1\ ,\cdots,\ \alpha_n \stackrel{?}{=} \eta_n\cdot\gamma_n\cdot\delta_n\}$$

If additionally there exists another exponent m_i such that $(\beta_i)^{m_i} = \omega_2$ for any $i\in[1..n]$ then $[F\mapsto(\omega_2)^p\cdot\omega_1]$ transforms the original problem into the same new one, for any value of p. Therefore, this second kind of solutions is infinitary because the value of p is not fixed.

Finally, it can be proved that the existence of such decompositions and exponents is decidable, even in the more complex case of linear second-order unification.

8 Conclusions

In this paper we have extended decidability results of Comon [1] and Schmidt-Schauß [17] for context unification to a proper extension of this unification problem, that we have called *linear second-order unification*, where n-ary free variables and λ-bindings are allowed. We have also described a complete unification procedure for this problem on which we have based all our decidability proofs. These proofs are completely independent from those given by Comon and Schmidt-Schauß.

This class of unification problems has good chances to become the basis of a notion of second-order rewriting. Then it would be interesting to find termination orderings and a critical pairs lemma for them. Decidability of general linear second-order unification problems, like decidability of general context unification problems, still remain as open questions.

References

1. H. Comon. Completion of rewrite systems with membership constraints, part I: Deduction rules and part II: Constraint solving. Technical report, CNRS and LRI, Université de Paris Sud, 1993. (To appear in J. of Symbolic Computation).
2. J. H. Gallier and W. Snyder. Designing unification procedures using transformations: A survey. *Bulletin of the EATCS*, 40:273–326, 1990.
3. W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
4. W. E. Gould. *A Matching Procedure for ω-Order Logic.* PhD thesis, Princeton Univ., 1966.
5. G. Huet. The undecidability of unification in third-order logic. *Information and Control*, 22(3):257–267, 1973.
6. G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
7. D. C. Jensen and T. Pietrzykowski. Mechanizing ω-order type theory through unification. *Theoretical Computer Science*, 3:123–171, 1976.
8. J. Levy and J. Agustí. Bi-rewriting, a term rewriting technique for monotonic order relations. In *4th Int. Conf. on Rewriting Techniques and Applications, RTA'93*, volume 690 of *LNCS*, pages 17–31, Montreal, Canada, 1993.
9. J. Levy and J. Agustí. Bi-rewriting systems. *J. of Symbolic Computation*, 1995. (To be published).
10. C. Loría-Sáenz. *A Theoretical Framework for Reasoning about Program Construction based on Extensions of Rewrite Systems.* PhD thesis, Univ. Kaiserslautern, 1993.
11. C. L. Lucchesi. The undecidability of the unification problem for third-order languages. Technical Report CSRR 2059, Dept. of Applied Analysis and Computer Science, Univ. of Waterloo, 1972.
12. O. Lysne and J. Piris. A termination ordering for higher-order rewrite systems. In *6th Int. Conf on Rewriting Techniques and Applications, RTA'95*, volume 914 of *LNCS*, Kaiserslautern, Germany, 1995.
13. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. of Logic and Computation*, 1:497–536, 1991.
14. T. Nipkow. Functional unification of higher-order patterns. In *8th IEEE Symp. on Logic in Computer Science, LICS'93*, pages 64–74, Montreal, Canada, 1993.
15. T. Pietrzykowski. A complete mechanization of second-order logic. *J. of the ACM*, 20(2):333–364, 1973.
16. C. Prehofer. *Solving Higher-Order Equations: From Logic to Programming.* PhD thesis, Technische Universität München, 1995.
17. M. Schmidt-Schauß. Unification of stratified second-order terms. Technical Report 12/94, Johan Wolfgang-Goethe-Universität, Frankfurt, Germany, 1995.
18. K. U. Schulz. Makanin's algorithm, two improvements and a generalization. Technical Report CIS-Bericht-91-39, Centrum für Informations und Sprachverarbeitung, Universität München, 1991.