Monadic Second-Order Unification is NP-Complete*

Jordi Levy[1], Manfred Schmidt-Schauß[2], and Mateu Villaret[3]

[1] IIIA, CSIC, Campus de la UAB, Barcelona, Spain.
http://www.iiia.csic.es/~levy
[2] Institut für Informatik, Johann Wolfgang Goethe-Universität,
Postfach 11 19 32, D-60054 Frankfurt, Germany.
http://www.ki.informatik.uni-frankfurt.de/persons/schauss/schauss.html
[3] IMA, UdG, Campus de Montilivi, Girona, Spain.
http://ima.udg.es/~villaret

Abstract. Monadic Second-Order Unification (MSOU) is Second-Order Unification where all function constants occurring in the equations are unary. Here we prove that the problem of deciding whether a set of monadic equations has a unifier is NP-complete. We also prove that Monadic Second-Order Matching is also NP-complete.

1 Introduction

Monadic Second-Order Unification (MSOU) is Second-Order Unification (SOU) where all function constants occurring in the problem are at most unary. It is well-known that the problem of deciding if a SOU problem has a solution is undecidable [Gol81,Far91,Lev98,LV00], whereas, in the case of MSOU, the problem is decidable [Hue75,Zhe79,Far88]. It is not a restriction to assume for this discussion that second order variables are also unary. In [SSS98], it is proved that the problem is NP-hard. In this paper, we prove that it is in NP.

MSOU can be decided by first making a guess for every variable, whether it uses its argument or not, and then calling string unification. This shows again that MSOU is decidable since string unification is decidable [Mak77], and also that MSOU is in PSPACE by using the result that string unification is in PSPACE [Pla99]. Since this is the currently known upper bound for string unification, our result that MSOU is NP-complete gives a sharp bound that (currently) cannot be obtained from results on string unification.

MSOU is a specialization of bounded second order unification (BSOU).[4] BSOU is decidable [SS04], which provides another proof of decidability of MSOU, but no tight upper complexity bound. On the other hand, our proof and results

* This research has been partially supported by the CICYT Research Projects CADVIAL (TIC2001-2392-C03-01) and LOGFAC (TIC2001-1577-C03-01).

[4] Accordingly to Property 2 we can restrict variables to be unary, as constants. Then in instantiations $\lambda x.t$ for variables X of the problem, the variable x can occur at most once in t, as in BSOU.

suggest an application to BSOU, which may result in proving a precise upper complexity bound for BSOU.

To prove that MSOU is in NP, first, we show how, for any solvable set of equations, we can represent (at least) one of the solution (unifiers) in polynomial space. Then, we prove that we can check if a substitution (written in such representation) is a solution in polynomial time.

There are two key results to obtain this sharp bound: One is the result on the exponential upper bound on the exponent of periodicity of size-minimal unifiers[Mak77,KP96,SSS98] (see Lemma 3). This upper bound allows us to represent exponents in linear space. The other key is a result of Plandowski [Pla94,Pla95] (see Theorem 1) where he proves that, given two context-free grammars with just one rule for every non-terminal symbol, we can check if they define the same (singleton) language in polynomial time (on the size of the grammars).

This paper proceeds as follows. After some preliminary definitions, in Section 3, we define lazy unifiers and prove that size-minimal unifiers are lazy unifiers. We prove some properties of singleton CFG in Section 4. We use a graph in order to describe the instance of some variable (Section 5). Sometimes, we need to rewrite such graph (Section 6). Based in this graph, we prove that, for any size-minimal lazy unifier, we can represent the value of some variable instance using a polynomial singleton grammar (Theorem 2). In Section 7, we extend this result to the whole unifier, and conclude the NP-ness of the MSOU problem.

2 Preliminary Definitions

Like in general second-order unification, we deal with a signature $\Sigma = \bigcup_{i\geq 0} \Sigma_i$, where constants of Σ_i are i-ary, and a set of variables $\mathcal{X} = \bigcup_{i\geq 0} \mathcal{X}_i$, where variables of $\mathcal{X}_i$ are also i-ary. Variables of $\mathcal{X}_0$ are therefore first-order typed and those of $\mathcal{X}_i$, with $i \geq 1$ are second-order typed. Well-typed terms are built as usual. We notate free variables with capital letters X, Y,... and bound variables and constants with lower-case letters x, y,... and a, b,... Terms are written in $\beta\eta$-normal form, thus arities can be inferred from the context. As far as we do not consider third or higher-order constants, first-order typed terms (in normal form) do not contain λ-abstractions, and second-order typed terms only contain λ-abstractions in topmost positions. The *size* of a term t is noted $|t|$ and defined as its number of symbols. A term is said to be *monadic* if it is built without using constants of arity greater than one, i.e. on a signature with $\Sigma_i = \emptyset$, for any $i \geq 2$. Notice that there is no restriction on the arity of variables.

Second-order substitutions are functions from terms to terms, defined as usual. For any substitution σ, the set of variables X, such that $\sigma(X) \neq X$, is finite and is called the *domain* of the substitution, and noted $\mathtt{Dom}(\sigma)$. A substitution σ can be presented as $[X_1 \to t_1, \ldots, X_n \to t_n]$, where $X_i \in \mathtt{Dom}(\sigma)$, t_i has the same type as X_i, and satisfies $t_i = \sigma(X_i)$. Given two substitutions σ and ρ, their composition is defined by $(\sigma \circ \rho)(t) = \sigma(\rho(t))$, for any term t, and is also a substitution, i.e. $\mathtt{Dom}(\sigma \circ \rho)$ is finite. We say that a substitution σ is *more general*

than another substitution ρ, noted $\sigma \preceq \rho$, if there exists a substitution τ such that $\rho(X) = \tau(\sigma(X))$, for any variable $X \in \mathtt{Dom}(\sigma)$. This defines a preorder relation on substitutions. An equivalence relation can also be defined as $\sigma \approx \rho$ if $\sigma \preceq \rho$ and $\rho \preceq \sigma$. A substitution σ is said to be *ground* if $\sigma(X)$ is a closed term, i.e. it does not contains free occurrences of variables, for any $X \in \mathtt{Dom}(\sigma)$.

An instance of the MSOU problem is a *set of equations* $\{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\}$ where t_i's and u_i's are monadic terms with the same first-order type, i.e. not containing λ-abstractions. A *solution* or *unifier* is a second-order substitution σ solving all equations: $\sigma(t_i) = \sigma(u_i)$ modulo $\beta\eta$-equality. A unifier σ of E is said to be *ground* if $\sigma(t_i)$ does not contain free occurrences of variables, for any $t_i \stackrel{?}{=} u_i \in E$. (Notice that not all ground substitutions that are unifiers are ground unifiers, because t_i may contain variables not instantiated by the unifier). A ground unifier is said to be *size-minimal* if it minimizes $\sum_{i=1}^{n} |\sigma(t_i)|$ among all ground unifiers. (Notice that w.l.o.g. size-minimal unifiers are required to be ground, because if a problem has a non-ground unifier, then it has also a ground unifier of equal or smaller size). *Most general unifiers* are defined as usual.

Notice that instances of the problem are required to be build from monadic terms, but there are not restrictions on the solutions. However, the following property ensures that most general unifiers instantiate variables by monadic terms.

Property 1. For any set of second-order unification equations E, and most general unifier σ, all constants occurring in σ also occur in E.

This property does not hold for variables. Even if the set of equations is built from unary second-order variables, most general unifiers can introduce fresh n-ary variables with $n \geq 2$. For instance, the set of equations $\{X(a) \stackrel{?}{=} Y(b)\}$ has only a most general unifier $[X \mapsto \lambda x.Z(x,b), Y \mapsto \lambda x.Z(a,x)]$, that introduces a binary second-order variable Z. Fortunately, non-unary variables do not give rise to undecidability, we need non-unary constants. In fact, one single binary constant is enough to generate a class of undecidable second-order unification problems [Far88,LV02].

3 Lazy Unifiers

We can restrict instances of second-order variables to ones that do not use their arguments, *whenever this is possible*. In this way we obtain what we call *lazy unifiers*.

Definition 1. *A substitution σ is said to be a* lazy unifier *if*

1. *it is a ground unifier, and,*
2. *it can be decomposed as $\sigma = \rho \circ \tau$, where τ is a most general unifier and ρ has the form $[X_1 \mapsto \lambda x_1.\cdots.\lambda x_{n_1}.a_1, \ldots, X_m \mapsto \lambda x_1.\cdots.\lambda x_{n_m}.a_m]$, where $a_1, \ldots, a_m \in \Sigma_0$ are first-order constants.*

Lemma 1. *For any solvable set of MSOU equations E containing at least a 0-ary constant, there exists a lazy unifier σ that does not introduce constants not occurring in E.*

Lemma 2. *Any size-minimal unifier is a lazy unifier.*

From now on, when we say a *solution* of a set of equations, we mean a *lazy unifier*. We also assume that any set of equations contains, at least, a 0-ary constant. The following property allows us to go a step further and assume that all variables are unary.

Property 2. Monadic SOU is NP-reducible to monadic SOU where all variable are second-order typed and unary.

From now on, since all symbols are unary or zero-ary, we can avoid parenthesis, and represent the term $a(X(b))$ as the word $a\,X\,b$, and $\lambda x.a(x)$ as $\lambda x.a\,x$.

We know that the size-minimal ground unifiers of a monadic second-order unification problem satisfies the exponent of periodicity lemma [Mak77,KP96,SSS98,SS04]. Therefore, from Lemma 2, we can conclude that it also holds for lazy unifiers:

Lemma 3 ([SS04]). *There exists a constant $\alpha \in \mathbb{R}$ such that, for any solvable monadic second-order unification problem E, there exists a lazy unifier σ such that, for any variable X, and words w_1, w_2 and w_3,*

$$\sigma(X) = \lambda x.w_1\, w_2^n\, w_3\, x \text{ and } w_2 \text{ not empty implies } n \leq 2^{\alpha|E|}.$$

4 Singleton Context Free Grammars

A *context-free grammar (CFG)* is a 4-tuple (Σ, N, P, s), where Σ is an alphabet of *terminal* symbols, N is an alphabet of *non-terminal* symbols (contrarily to the standard conventions, and in order to avoid confusion between free variables and non-terminal symbols, all terminal and non-terminal symbols are denoted by lower-case letters), P is a finite set of rules, and $s \in N$ is the *start symbol*. We will not distinguish a particular start symbol, and we will represent a context free grammars as a 3-tuple (Σ, N, P). Moreover, we will use Chomsky grammars with at most two symbols on the right hand side of the rules.

Definition 2. *We say that a context free grammar $G = (\Sigma, N, P)$* generates *a word $w \in \Sigma^*$ if there exists a non-terminal symbol $a \in N$ such that w belongs to the language defined by (Σ, N, P, a). In such case, we also say that a generates w.*

We say that a context free grammar is a singleton CFG *if it is not recursive and every non-terminal symbol occurs in the left-hand side of exactly one rule. Then, every non-terminal symbol $a \in N$ generates just one word, denoted w_a, and we say that a* defines *w_a. In general, for any sequence $\alpha \in (\Sigma \cup N)^*$, $w_\alpha \in \Sigma^*$ denotes the word generated by α.*

Plandowski [Pla94,Pla95] defines singleton grammars, but he calls them *grammars defining set of words*. He proves the following result.

Theorem 1 ([Pla95], Theorem 33). *The word equivalence problem for singleton context-free grammars is defined as follows: Given a grammar and two non-terminal symbols a and b, to decide whether $w_a = w_b$. This problem can be solved in polynomial worst-case time on the size of the grammar.*

Definition 3. *Let $G = (\Sigma, N, P)$ be a singleton CFG.*
For any terminal symbol $a \in \Sigma$, we define $\mathtt{depth}(a) = 0$, and for any non-terminal symbol $a \in N$ we define

$$\mathtt{depth}(a) = \max\{\mathtt{depth}(b_i) + 1 \mid a \to b_1\, b_2 \in P \ \wedge \ i = 1, 2\}$$

We define the depth of G as $\mathtt{depth}(G) = \max\{\mathtt{depth}(a) \mid a \in N\}$.
We define the size of G as its number of rules. (Notice that this definition is for Chomsky grammars).

We can enlarge a grammar in order to define concatenation, exponentiation and prefixes of words already defined by the grammar. We use these operation in the next sections to build the grammar defining some lazy unifier of the unification problem. The following three lemmas state how the size and the depth of the grammar are increased with these transformations.

Lemma 4. *Let G be a singleton grammar defining the words $w_1, \ldots, w_n$. There exists a singleton grammar $G' \supseteq G$ that defines the word $w = w_1 \ldots w_n$ and satisfies*

$$\begin{aligned} |G'| \ &\leq \ |G| + n - 1 \\ \mathtt{depth}(G') \ &\leq \ \mathtt{depth}(G) + \lceil \log n \rceil \end{aligned}$$

Lemma 5. *Let G be a singleton grammar defining the word w. For any n, there exists a singleton grammar $G' \supseteq G$ that defines the word w^n and satisfies*

$$\begin{aligned} |G'| \ &\leq \ |G| + 2\,\lfloor \log n \rfloor \\ \mathtt{depth}(G') \ &\leq \ \mathtt{depth}(G) + \lceil \log n \rceil \end{aligned}$$

Lemma 6. *Let G be a singleton grammar defining the word w. For any prefix or suffix w' of w, there exists a singleton grammar $G' \supseteq G$ that defines w' and satisfies*

$$\begin{aligned} |G'| \ &\leq \ |G| + \mathtt{depth}(G) \\ \mathtt{depth}(G') \ &= \ \mathtt{depth}(G) \end{aligned}$$

Definition 4. *Consider a signature composed by a nonempty set Σ_0 of first-oder constants, a set Σ_1 of second-order and unary constants, a set N of non-terminal symbols, and a set $\mathcal{X}_1$ of second-order and unary variables.*

A generalized set of equations *is a pair $\langle E, G\rangle$, where E is a set of equations of the form $\{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\}$ where terms t_i and u_i, for $i = 1, \ldots, n$, are sequences of $(\Sigma_1 \cup \mathcal{X}_1 \cup N)^*\, \Sigma_0$, and $G = \langle \Sigma_1, N, P\rangle$ is a singleton context free grammar with just one production for every non-terminal symbol of N occurring in E.*

A generalized unifier *of* $\langle E, G\rangle$ *is a pair* $\langle \sigma, G'\rangle$*, where* σ *is a mapping that assigns either* $[X \mapsto \lambda x.a\,x]$ *or* $[X \mapsto \lambda x.a\,b]$ *or* $[X \mapsto \lambda x.x]$ *or* $[X \mapsto \lambda x.b]$ *to each variable* X*, for some* $a \in N \cup \Sigma_1$ *and* $b \in \Sigma_0$*, and* $G' \supseteq G$ *is a singleton grammar that contains a production for every non-terminal symbol of* E *or* σ*, such that, replacing every variable* X *by its instance in* E*,* $\beta\eta$*-normalizing both sides of each equation, and then replacing every nonterminal symbol* a *by the word* w_a *that it defines, all the equations of* E *are satisfied as equalities.*

Notice that non-terminal symbols derive into sequences of second-order constants, and that we do not consider first-order variables.

Example 1. Consider the generalized set of equations $\langle E, G\rangle$, defined by

$$E = \{b\,X\,X\,f\,a \stackrel{?}{=} Y\,Y\,Y\,a\}$$
$$G = \{b \to c\,c,\ c \to f\,f\}$$

Then, the pairs $\langle \sigma, G'\rangle$, defined by

$$\sigma = [X \mapsto \lambda x.c\,x,\ Y \mapsto \lambda x.d\,x] \qquad \sigma = [X \mapsto \lambda x.a,\ Y \mapsto \lambda x.b\,a]$$
$$G' = \{b \to c\,c,\ c \to f\,f,\ d \to c\,f\} \quad \text{and} \quad G' = \{b \to c\,c,\ c \to f\,f\}$$

are generalized unifiers. In fact, these would be the only two lazy unifiers found by our algorithm. Notice that the second one is a lazy unifier corresponding to the most general unifier $[X \mapsto \lambda x.Z,\ Y \mapsto \lambda x.f\,f\,f\,f\,Z]$.

From now on, an instance of a MSOU problem will be a generalized set of equations. Let σ assign $[X \mapsto \lambda x.a\,x]$. We will use $\sigma(X)$ to denote indistinctly the functions $\lambda x.a\,x$ or $\lambda x.w_a\,x$, or the word w_a, being its meaning clear from the context.

Notice that any monadic set of equations E is equivalent to the generalized set of equations $\langle E, \emptyset\rangle$, and vice versa, any generalized set of equations is equivalent to the monadic set of equations that we obtain by replacing every non-terminal symbol by the word that it defines. Therefore, solvability of monadic set of equations and of generalized set of equations are, with respect to decidability, equivalent problems. With respect to their complexity, we will prove that solvability of generalized sets of equations can be decided in NP-time. This implies that solvability MSOU is also in NP.

5 The Graph of Surface Dependencies

In this Section we define the graph of surface dependencies. The purpose of this graph is to describe, for a given lazy unifier σ, the instance $\sigma(X)$ of some variable X of the problem. In some cases, the ones not covered by Lemmas 8, 9 and 10, the graph is not able to describe such instances, and it becomes necessary to rewrite it to obtain a new graph with this capability. This graph rewriting process will be described in Section 6.

The graph of surface dependencies is defined only for *simplified* equations (not containing rigid-rigid pairs, i.e. pairs with constants in the head of both sides of the equation). In case we have such kind of equations we can simplify them. This can increase the size of the associated grammar as the following Lemma states. After this Lemma, if nothing is said, we will assume that all sets of equations are simplified.

Lemma 7. *Given a generalized set of equations $\langle E, G\rangle$, where $E = \{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\}$, we can get an equivalent* simplified *problem $\langle E', G'\rangle$, where $E' = \{t'_1 \stackrel{?}{=} u'_1, \ldots, t'_m \stackrel{?}{=} u'_m\}$, i.e. a problem with exactly the same set of solutions, where, for every equation $t'_i \stackrel{?}{=} u'_i$, either t'_i or u'_i has a variable in the head (all rigid-rigid pairs have been removed), and*

$$\begin{array}{ll} |E'| \leq |E| & |G'| \leq |G| + n\,\mathtt{depth}(G) \\ m \leq n & \mathtt{depth}(G') = \mathtt{depth}(G) \end{array}$$

Definition 5. *Let $\langle E, G\rangle$ be a simplified generalized set of equations, the graph of surface dependencies of $\langle E, G\rangle$ is defined as follows.*

Let $\approx$ be the minimal equivalence relation defined by: if E contains an equation of the form $X\,w_1 \stackrel{?}{=} Y\,w_2$, then $X \approx Y$. This defines a partition on the variables of E.

Every node of the graph is labeled by an $\approx$-equivalence class of variables, the empty set, or a first-order constant, and every edge is labeled by either a terminal or a second-order constant. Then:

- *We add just one node for every $\approx$-equivalence class of variables.*
- *For every equation of the form $X\,w_1 \stackrel{?}{=} a_1 \cdots a_n\,Y\,w_2$, where $a_1 \cdots a_n \in (N \cup \Sigma_1)^*$, we add a sequence of nodes with the empty set as labels, and a sequence of labeled edges of the form*

  where $X \in L_1$ and $Y \in L_2$.
- *For every equation of the form $X\,w_1 \stackrel{?}{=} a_1 \cdots a_n\,b$, where $a_1 \cdots a_n \in (N \cup \Sigma_1)^*$ and $b \in \Sigma_0$, we add a sequence of nodes with the empty set as label, and a sequence of labeled edges of the form*

  where $X \in L$.

Notice that, for every variable X, there is just one node with label L satisfying $X \in L$. This is called its *corresponding* node.

The cycles of this graph describe the base of *some* exponentiation occurring in the instance of some variables. For instance, the solutions of the equation $X\,f\,a \stackrel{?}{=} f\,X\,a$ have the form $[X \mapsto \lambda x.f^n\,x]$, for some $n \geq 0$. The base of this power is described by a cycle in its graph of dependencies:

We prove that, if one of the following conditions holds:

1. the graph contains a cycle (Lemma 8),
2. there is a node with two exiting edges with distinct and *divergent* labels (Lemma 9), or
3. there is at most one exiting edge, for every node (Lemma 10),

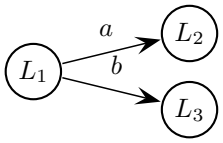
then the graph of surface dependencies describes the instance of some variable. In the rest of cases it may be necessary to rewrite the graph in order to obtain the desired description.

Lemma 8. *Given a generalized set of equations $\langle E, G\rangle$, let D be its graph of surface dependencies. If D contains a cycle, then, for every lazy unifier σ, whose exponent of periodicity does not exceed k, there exists a variable X such that its corresponding node is inside the cycle, and, if $\alpha \in (N \cup \Sigma_1)^*$ is the sequence of transitions completing the cycle from this node, for some $0 \leq n \leq k$, and some prefix w' of w_α, we have $\sigma(X) = \lambda x.(w_\alpha)^n\, w'\, x$.*

Moreover, there exists a singleton context-free grammar $G' \supseteq G$ that generates $(w_\alpha)^n\, w'$ and satisfies

$$|G'| \leq |G| + \mathtt{depth}(G) + |D| + \lceil \log |D| \rceil + 2\,\lfloor \log k \rfloor$$
$$\mathtt{depth}(G') \leq \mathtt{depth}(G) + \lceil \log |D| \rceil + \lceil \log k \rceil + 1$$

Definition 6. *A dependence graph D is said to contain a* divergence $L_2 \overset{a}{\leftarrow} L_1 \overset{b}{\rightarrow} L_3$, *if it contains a subgraph of the form:*



where neither w_a is a prefix of w_b, nor w_b a prefix of w_a.

Lemma 9. *Given a generalized set of equations $\langle E, G\rangle$, let D be its graph of surface dependencies. If D contains a divergence $L_2 \overset{a}{\leftarrow} L_1 \overset{b}{\rightarrow} L_3$, then, for any lazy unifier σ of E, there exists a variable $X \in L_1$ and some common prefix w' of w_a and w_b, such that $\sigma(X) = \lambda x.w'\, x$.*

Moreover, there exists a singleton context-free grammar $G' \supseteq G$ that generates w', has the same depth as G, and satisfies $|G'| \leq |G| + \mathtt{depth}(G)$.

Lemma 10. *Given a generalized set of equations $\langle E, G\rangle$, let D be its graph of surface dependencies. If D contains at most one exiting edge, for every node, and D does not contain cycles, then, for any size-minimal lazy unifier σ of E, one of the following properties holds:*

1. *for some node L without exiting edges, some variable $X \in L$, and some first-order constant $b \in \Sigma_0$, we have $\sigma(X) = \lambda x.b$, or*

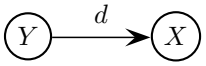
2. *for some node L, let α be the unique path starting at L and finishing in a node without exiting edges, then, either*
   (a) *the sequence α ends in a node labeled with a first-order constant $b \in \Sigma_0$, and $\sigma(X) = \lambda x.w_\alpha\, b$, or*
   (b) *for some proper prefix w' of w_α, we have $\sigma(X) = \lambda x.w'\, x$.*

Moreover, there exists a singleton context-free grammar $G' \supseteq G$ that, in each case, defines w_α or w', and satisfies

$$|G'| \leq |G| + \mathtt{depth}(G) + |D| + \lceil \log |D| \rceil - 1$$
$$\mathtt{depth}(G') \leq \mathtt{depth}(G) + \lceil \log |D| \rceil$$

Remark 1. Notice that this Lemma, contrarily to Lemmas 8 and 9, only applies to *size-minimal* lazy unifiers. Notice also that in case 1, it forces some variable to *forget* its argument, and therefore, applies to lazy unifiers, but not to most general unifiers. In this point is where the search of a size-minimal lazy unifier differs from the search of a most-general unifier, and in fact, where our algorithm looses its completeness and soundness when applied to word unification. (Otherwise this paper would prove NP-completeness of word unification!!).

For instance, in Example 1, the graph of surface dependencies is



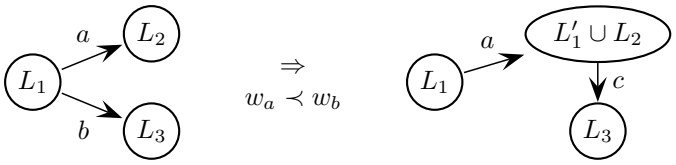
Therefore, we are in the conditions of Lemma 10. Applying sub-case 1, we find $\sigma(X) = \lambda x.a$ (and later, $\sigma(Y) = \lambda x.d\, a$). Applying sub-case 2b, we find, among others, $\sigma(Y) = \lambda x.f\ f\ f\ x$ (and later, $\sigma(X) = \lambda x.f\ f\ x$). The first one is a lazy, but not a most general unifier, whereas the second one is a ground and most general (and therefore lazy) unifier. Notice that there are other most general unifiers of the form $[X \mapsto \lambda x.f^{2+3n}\, x,\ Y \mapsto \lambda x.f^{3+2n}\, x]$ that are not found by our algorithm for $n \geq 1$.

6 Rewriting the Graph of Dependencies

There are graphs not satisfying any of the conditions of Lemmas 8, 9 and 10. These graphs contain a node with two *compatible* exiting edges. In other words, these graphs contain a subgraph, used as redex in our transformation rules, of the form $L_2 \overset{a}{\leftarrow} L_1 \overset{b}{\rightarrow} L_3$, where w_a is a prefix of w_b, or w_b is a prefix of w_a. An example of such kind of graphs is shown in Example 2. In these cases, in order to obtain a description of some variable instantiation, it can be necessary to transform the graph of dependencies using the following graph rewriting system. These rules transform the redexes described above.

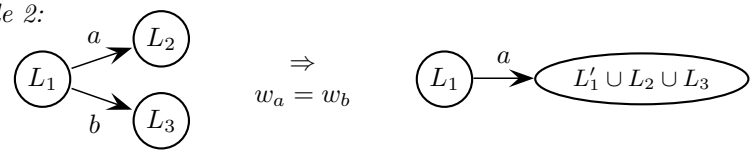
Definition 7. *We consider a transformation system described by rules that work on pairs of the form $\langle D, G\rangle$, where D is a dependence graph and G is a singleton grammar. The transformation on the dependence graph is interpreted as a graph rewriting system.*

Rule 1:



where c is a fresh non-terminal symbol, and $L_1' = \{X' \mid X \in L_1\}$ is the set of labels of L_1 where we have added a quote to every variable name.

Rule 2:



In the first rule, the grammar G is transformed in order to be able to define the word w_c satisfying $w_b = w_a\, w_c$. According to Lemma 6, we can obtain such grammar G' satisfying

$$|G'| \leq |G| + \mathtt{depth}(G)$$
$$\mathtt{depth}(G') = \mathtt{depth}(G)$$

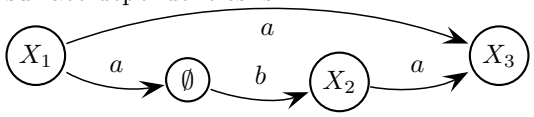
In the second rule, the grammar is not modified.

These rules can only be applied if the graph has no cycles and there are no divergences.

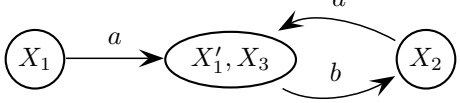
Example 2. Consider the following set of equations, and their set of unifiers, for $n \geq 0$.

$$\begin{array}{ll} X_1\, c_1 \stackrel{?}{=} a\, b\, X_2\, c_1 & X_1 \mapsto \lambda x.(a\, b)^{n+2}\, x \\ X_1\, c_2 \stackrel{?}{=} a\, X_3\, a\, b\, c_2 & X_2 \mapsto \lambda x.(a\, b)^{n+1}\, x \\ X_2\, c_3 \stackrel{?}{=} a\, X_3\, c_3 & X_3 \mapsto \lambda x.b\,(a\, b)^{n}\, x \end{array}$$

The graph of surface dependencies is



Using the second rule of the graph rewriting system we get



Lemma 11. *Any graph rewriting sequence $D_1 \Rightarrow^* D_n$ has length at most $n \leq |D_1|^2$, where $|D_1|$ is the number of edges of D_1.*

As we have said, if the graph of surface dependencies does not contain redexes, then it describes a variable instance. Moreover, depending on the lazy unifier, even if the graph contains redexes, it can also describe some variable instance. We distinguish, according to the lazy unifier, between *incompatible* and *compatible* redexes. If the graph contains an incompatible redex, it already describes a variable instance, and must not be rewritten. Thus, we only rewrite a graph if all redexes are compatible.

Definition 8. *Given a generalized set of equations $\langle E, G\rangle$, its graph of dependencies D, and a lazy unifier σ, we say that a graph rewriting step with redex $L_2 \overset{a}{\leftarrow} L_1 \overset{b}{\rightarrow} L_3$ is* incompatible with σ *if, for some variable $X \in L_1$, we have $\sigma(X) = \lambda x.w'\, x$, where w' is a proper prefix of w_a and w_b. Otherwise, it is said to be* compatible.

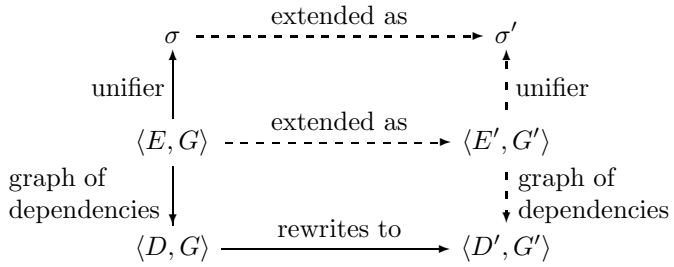
Lemma 12. *Given a generalized set of equations $\langle E, G\rangle$, its graph of dependencies D, and a lazy unifier σ, if there exists an incompatible with σ redex $L_2 \overset{a}{\leftarrow} L_1 \overset{b}{\rightarrow} L_3$ in D, i.e. there exists a variable $X \in L_1$ with $\sigma(X) = \lambda x.w'\, x$, where w' is a proper prefix of w_a and w_b, then there exist a singleton context-free grammar $G' \supseteq G$ that defines w' and satisfies $|G'| \leq |G| + \mathtt{depth}(G)$ and $\mathtt{depth}(G') = \mathtt{depth}(G)$.*

When we rewrite a graph, the new graph does not describe *exactly* a variable instance of the lazy unifier, but an instance of a *modification* of this unifier. This new unifier is also a solution of a *modification* of the set of equations. Therefore, when we rewrite the graph of surface dependencies, apart from the associated grammar, we have to transform the set of equations and its lazy unifier. The next Lemma describes how we have to make such modifications.

Lemma 13. *Given a generalized set of equations $\langle E, G\rangle$, its graph of dependencies D, a lazy unifier σ, and a compatible rewriting step $\langle D, G\rangle \Rightarrow \langle D', G'\rangle$, there exist a generalized set of equations $\langle E', G'\rangle$ and a substitution σ' such that*

1. *σ' is a lazy unifier of $\langle E', G'\rangle$,*
2. *D' is the graph of dependencies of E', and*
3. *σ' extends σ as $\sigma'(X) = \sigma(X)$, for any variable occurring in E, and satisfies $\sigma(X) = \sigma'(X) = \lambda x.w_a\, \sigma'(X')\, x$, for any variable $X \in L_1$ occurring in the redex of the rewriting step.*

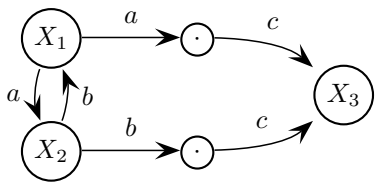
Graphically we can represent this Lemma as a category-like commutative diagram:



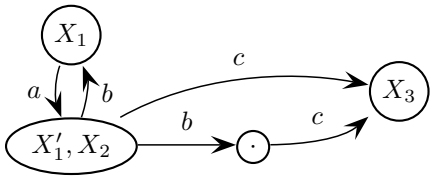
Example 3. Consider the following set of equations

$$X_1\, Y_1\, d \stackrel{?}{=} a\, c\, X_3\, d \quad X_1\, Y_2\, d \stackrel{?}{=} a\, X_2\, Y_3\, d$$
$$X_2\, Y_6\, d \stackrel{?}{=} b\, c\, X_3\, d \quad X_2\, Y_4\, d \stackrel{?}{=} b\, X_1\, Y_5\, d$$

The graph of surface dependencies is



Now either $\sigma(X_1) = \lambda x.x$, and we already have the description of a variable instance, or we can apply a compatible rewriting step, to the redex $\emptyset \overset{a}{\leftarrow} \{X_1\} \overset{a}{\rightarrow} \{X_2\}$, to obtain the following graph:



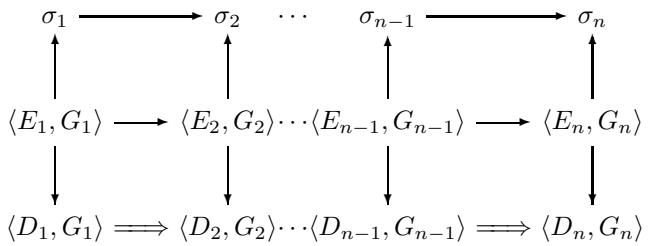
Now, we have an inconsistency, and we can apply Lemma 9 to ensure that either $\sigma'(X_1') = \lambda x.x$ or $\sigma'(X_2) = \lambda x.x$. Applying Lemma 13, in the first case, we obtain $\sigma(X_1) = \lambda x.a\, x$, and in the second case $\sigma(X_2) = \lambda x.x$. Now, we have the instance of some variable that can be instantiated in the equations, and we can repeat the process to obtain instances of other variables.

Theorem 2. *Let σ be a size-minimal lazy unifier of $\langle E, G\rangle$ with exponent of periodicity not exceeding k. Then there exist a variable X in E, and a singleton grammar G', deriving $\sigma(X)$ and such that:*

$$|G'| \leq |G| + \mathcal{O}(|E|^2\, \mathtt{depth}(G) + \log k)$$
$$\mathtt{depth}(G') \leq \mathtt{depth}(G) + \mathcal{O}(\log k + \log |E|)$$

Proof. If the set of equations E is not simplified, we can apply Lemma 7 in order to obtain an equivalent set of simplified equations. This transformation implies a worst-case increase of order $\mathcal{O}(|E|\, \mathtt{depth}(G))$ on the size of G, which is compensated by the increase of order $\mathcal{O}(|E|^2\, \mathtt{depth}(G) + \log k)$ stated on the Theorem.

Let $\langle E_1, G_1\rangle = \langle E, G\rangle$, $\sigma_1 = \sigma$, and D_1 be the graph of dependencies of E_1. If Lemmas 8, 9 and 10 are not applicable, then there exists a redex in the graph D_1. Then either there exists a redex incompatible with σ_1, or all redexes are compatible. In the second case, we can rewrite $D_1 \Rightarrow D_2$, and use Lemma 13 to find a new substitution σ_2, and set of generalized equations $\langle E_2, G_2\rangle$. Repeating this argument, we can obtain a diagram of the form:



where D_n either satisfies Lemmas 8, 9 or 10, or contains a redex incompatible with σ_n. Now, either using Lemmas 8, 9 or 10, if D_n does not contain redexes, or using Lemma 12, if D_n contains an incompatible redex, we can find the instance $\sigma_n(X)$ of some variable X of E_n. Using the bounds of Lemmas 8, 9, 10 and 12, there exists a singleton context-free grammar G'_n that generates $\sigma_n(X)$ and satisfies:

$$|G'_n| \leq |G_n| + \mathtt{depth}(G_n) + |D_n| + \lceil \log |D_n| \rceil + 2 \lfloor \log k \rfloor$$
$$\mathtt{depth}(G'_n) \leq \mathtt{depth}(G_n) + \lceil \log k \rceil + \lceil \log |D_n| \rceil + 1$$

Notice that the worst bounds are given by Lemma 8.

By Lemma 11 we have $n \leq |D_1|^2$, and using Lemma 6 (see Definition 7), we have

$$|G_n| \leq |G_1| + |D_1|^2 \, \mathtt{depth}(G_1)$$
$$\mathtt{depth}(G_n) = \mathtt{depth}(G_1)$$

Moreover, the size of the dependence graph does not increase during the rewriting steps, therefore $|D_n| \leq |D_1|$. Notice that it is possible that X would not be a variable of E_1. In this case, X will be a variable with primes, say $X^{(m)}$ with $m \leq n$. Then it is possible to construct the instance $\sigma_1(X)$ from $\sigma_n(X^{(m)})$ as $\sigma_1(X) = \lambda x. w_{a_1} \cdots w_{a_m} \sigma_n(X^{(m)})\, x$, where w_{a_i} is the word generated by a_i, and this is a non-terminal symbol of $G_i \subseteq G_n$. Therefore, if we already have a grammar generating $\sigma_n(X^{(m)})$, we can construct a grammar generating $\sigma_1(X)$ by simply adding new rules. In the worst case, this increases the depth of the grammar by $\lceil \log(m+1) \rceil$, and its size by m.

Summarizing, we can find a grammar $G' \supseteq G$ generating $\sigma_1(X)$, for some variable X of E, and satisfying:

$$|G'| \leq |G| + |D|^2 \, \mathtt{depth}(G) + |D| + \mathtt{depth}(G) + \lceil \log |D| \rceil + 2 \lfloor \log k \rfloor + |D|^2$$
$$\mathtt{depth}(G') \leq \mathtt{depth}(G) + \lceil \log k \rceil + \lceil \log |D| \rceil + \lceil \log(|D|^2 + 1) \rceil$$

Using orders:

$$|G'| \leq |G| + \mathcal{O}(|D|^2 \, \mathtt{depth}(G) + \log k)$$
$$\mathtt{depth}(G') \leq \mathtt{depth}(G) + \mathcal{O}(\log k + \log |D|)$$

Since $|D|$ is the number of edges in the graph of dependencies, and $|E|$ the number of symbols in the equations, by construction of the graph of dependencies from the equations, we have $|D| \leq |E|$. □

7 Main Results and Some Remarks

Theorem 2 states that, given a generalized set of equations $\langle E, G \rangle$, we can build a new grammar defining the instance of some variable of E. Then we can instantiate this variable in the equations in order to obtain a new set of equations with one variable less. This process does not increase the size of the equations, since we use just one non-terminal symbol on the grammar to describe the instance of the variable. We can repeat N times this process, being N the number of variables, bounded by the original size of the problem $|E|$. The increase on the depth of the grammar is $N\,\mathcal{O}(\log k)$ (being $k = 2^{\mathcal{O}(|E|)}$ the bound on the exponent of periodicity), thus $\mathcal{O}(|E|^2)$. The increase on the size of the grammar is

$N\,\mathcal{O}(|E|^2\,\mathtt{depth}(G) + \log k)$. Although it depends on the depth of the grammar (see Remark 2), it has order $\mathcal{O}(|E|^5)$. This allows us to conclude:

Theorem 3. *For any solvable generalized set of equations $\langle E, \emptyset\rangle$, there exists a lazy unifier $\langle \sigma, G\rangle$ such that the size of $\langle \sigma, G\rangle$ is polynomially bounded on the size of E, in fact $|\sigma| = \mathcal{O}(|E|)$, $|G| = \mathcal{O}(|E|^5)$, and $\mathtt{depth}(G) = \mathcal{O}(|E|^2)$.*

Theorem 3 proves the existence of a polynomially bounded solution for every solvable MSOU problem. Now we have to prove that checking if a substitution is a solution can be performed in polynomial time. Given a substitution, we instantiate the equations. This will remove all variable occurrences, and it will not increase their sizes, because every variable is replaced by just one symbol of the grammar (in some cases, their argument are removed, but this decreases the size). With a small increase of $|E|$ (according to Lemma 4) on the size of the grammar, we can obtain a new grammar defining both sides of every equation, and use Plandowski's Theorem 1 to check their syntactic equality.

Corollary 1. *Monadic Second-Order Unification is NP-complete.*

Theorem 4. *Monadic Second-Order Matching is NP-complete.*

Note that the proof of NP-hardness of MSOU in [SS04] is not a MSO-matching problem, so the proof of Theorem 4 requires a different encoding. We also use the ONE-IN-THREE-SAT problem, which is known to be NP-complete. We associate a pair of second-order variables X_p, Y_p to every propositional variable p, and use a pair of equations $X_p\,Y_p\,b \stackrel{?}{=} a\,b$ and $X_p\,Y_p\,c \stackrel{?}{=} a\,c$, to ensure that their values are $\lambda x.a\,x$, interpreted as true, or $\lambda x.x$, interpreted as false. Then, we encode every clause $p \vee q \vee r$ as $X_p\,X_q\,X_r\,b \stackrel{?}{=} a\,b$.

Remark 2. Theorem 3 clarifies the increase of the size of the grammar representing the solution of a set of equations, after instantiating N variables, according to Theorem 2. This Theorem fixes this increase with respect to the size of the equations, the logarithm of the upper bound on the exponent of periodicity, and the *depth* of the grammar. The question is then: Could we avoid the use of the depth of the grammar? The answer is no. For instance, Lemma 6 says that, if we want to define a prefix of some word defined by a grammar G, in the worst case, we can keep the depth, but we may need to increase the size of G' as $|G'| \leq |G| + \mathtt{depth}(G)$. If we only use the size of the grammar to characterize it, then in the worst case we may be forced to duplicate the size of the grammar $|G'| \leq 2\,|G|$. Each time that we instantiate a variable, it can be necessary to define a new prefix, therefore, in the worst case, the size of the resulting grammar would be 2^N, being $N \leq |E|$ the number of variables. This would result in an exponential upper bound on the size of the grammar.

8 Conclusions

In this paper we prove that Monadic Second-Order Unification (MSOU) is in NP using a result of Plandowski about context-free grammars [Pla94,Pla95].

This result, together the NP-hardness of the problem [SS04] proves its NP-completeness. As we mention in the introduction, MSOU is a specialization of Bounded Second-Order Unification (BSOU). This suggests us that some of the ideas contained in this paper could be used to try to prove that BSOU is in NP.

References

[Far88] W. M. Farmer. A unification algorithm for second-order monadic terms. *Annals of Pure and Applied Logic*, 39:131–174, 1988.

[Far91] W. M. Farmer. Simple second-order languages for wich unification is undecidable. *Theoretical Computer Science*, 87:173–214, 1991.

[GJ79] M. Garey and D. Johnson. *"Computers and Intractability": A guide to the theory of NP-completeness*. W.H. Freeman and Co., San Francisco, 1979.

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

[Hue75] G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[KP96] A. Kościelski and L. Pacholski. Complexity of Makanin's algorithm. *Journal of the ACM*, 43(4):670–684, 1996.

[Lev98] J. Levy. Decidable and undecidable second-order unification problems. In *Proceedings of the 9th Int. Conf. on Rewriting Techniques and Applications (RTA'98)*, volume 1379 of *LNCS*, pages 47–60, Tsukuba, Japan, 1998.

[LV00] J. Levy and M. Veanes. On the undecidability of second-order unification. *Information and Computation*, 159:125–150, 2000.

[LV02] J. Levy and M. Villaret. Currying second-order unification problems. In *Proc. of the 13th Int. Conf. on Rewriting Techniques and Applications (RTA'02)*, volume 2378 of *LNCS*, pages 326–339, Copenhagen, Denmark, 2002.

[Mak77] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, 32(2):129–198, 1977.

[Pla94] W. Plandowski. Testing equivalence of morphisms in context-free languages. In J. van Leeuwen, editor, *Proc. of the 2nd Annual European Symposium on Algorithms (ESA'94)*, volume 855 of *LNCS*, pages 460–470, 1994.

[Pla95] W. Plandowski. *The Complexity of the Morphism Equivalence Problem for Context-Free Languages*. PhD thesis, Department of Mathematics, Informatics and Mechanics, Warsaw University, 1995.

[Pla99] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proc. of the 40th IEEE Annual Symposium on Foundations of Computer Science (FOCS'99)*, pages 495–500, 1999.

[SS01] M. Schmidt-Schauß. Stratified context unification is in PSPACE. In L. Fribourg, editor, *Proc. of the 15th Int. Workshop in Computer Science Logic (CSL'01)*, volume 2142 of *LNCS*, pages 498–512, 2001.

[SS04] M. Schmidt-Schauß. Decidability of bounded second order unification. *Information and Computation*, 188(2):143–178, 2004.

[SSS98] M. Schmidt-Schauß and K. U. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *Proc. of the 9th Int. Conf. on Rewriting Techniques and Applications (RTA'98)*, volume 1379 of *LNCS*, pages 61–75, Tsukuba, Japan, 1998.

[Zhe79] A. P. Zhezherun. Decidability of the unification problem for second order languages with unary function symbols. *Kibernetika (Kiev)*, 5:120–125, 1979. Translated as *Cybernetics* 15(5):735–741, 1980.