SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1113–1140

THE COMPLEXITY OF MONADIC SECOND-ORDER UNIFICATION*

JORDI LEVY†, MANFRED SCHMIDT-SCHAUß‡, AND MATEU VILLARET§

Abstract. Monadic second-order unification 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, where we use the technique of compressing solutions using singleton context-free grammars. We prove that monadic second-order matching is also NP-complete.

Key words. second-order unification, theorem proving, lambda calculus, context-free grammars, rewriting systems

AMS subject classifications. 03B15, 68Q42, 68T15

DOI. 10.1137/050645403

1. Introduction.

The solvability of monadic second-order unification problems considered in this paper is a decision problem with many natural relationships with other well-known decision problems (higher-order unification, word unification, context unification and specializations thereof). For various members of this family, the exact complexity is still not known. In this paper we show that solvability of monadic second-order unification problems is NP-complete, thus adding a new piece to the larger puzzle. The techniques we use in our proof turned out to be relevant for other problems as well. Before we describe the organization of the paper we illuminate in more detail the relationship of monadic second-order unification with the above problems, give a brief survey on the techniques used in our proof, and indicate further contributions obtained from our method.

1.1. Monadic second-order unification from a higher-order perspective.

Higher-order unification (HOU) is unification in the simply typed λ-calculus, i.e., the problem of, given two λ-terms with the same type, deciding if there is a substitution (of free variables by equally typed terms) that, applied to both terms, makes them equivalent w.r.t. $\alpha\beta\eta$-equality (see chapter about HOU in [1]).

Second-order unification (SOU) is a restriction of HOU where all variables are at most second-order typed and constants are at most third-order typed. Some authors restrict constants in SOU to also be second-order typed, and it is also common to consider just one base type. Here, we will also make these assumptions. It is well known that the problem of deciding if an SOU problem has a solution is undecidable [5], even

*Received by the editors November 16, 2005; accepted for publication (in revised form) February 28, 2008; published electronically July 2, 2008. A preliminary version of this work appeared in *Proceedings of the* 15*th Annual International Conference on Rewriting Techniques and Applications (RTA* '04*)*, where it was awarded the best paper prize. This research was partially supported by the CYCIT research projects iDEAS (TIN2004-04343) and Mulog2 (TIN2007-68005-C04-01).

http://www.siam.org/journals/sicomp/38-3/64540.html

†Institut d'Investigació en Intel·ligència Artificial (IIIA), Consejo Superior de Investigaciones Científicas (CSIC), Campus de la UAB, Bellaterra, 08193 Barcelona, Spain (levy@iiia.csic.es, http://www.iiia.csic.es/~levy).

‡Institut für Informatik, Johann Wolfgang Goethe-Universität, Postfach 11 19 32, D-60054 Frankfurt, Germany (schauss@ki.informatik.uni-frankfurt.de, http://www.ki.informatik.uni-frankfurt.de/persons/schauss/schauss.html).

§Informàtica i Matemàtica Aplicada, Universitat de Girona, Campus de Montilivi, 17071 Girona, Spain (villaret@ima.udg.es, http://ima.udg.es/~villaret).

if we impose additional restrictions on the number of second-order variables (just one), their number of occurrences (just four), and their arity (just one) [3, 9, 13]. All these undecidability proofs require a language with at least a binary or higher arity constant. In fact, one single binary constant is enough [3, 14].

Monadic second-order unification (MSOU) is a restriction of SOU where all function constants occurring in the problem are at most unary. Contrary to general SOU, MSOU is decidable [6, 24, 2]. In [23], it is proved that the problem is NP-hard. In this paper, our main contribution is to prove that MSOU is in NP.

Assuming that second-order variables, like constants, are also unary does not affect the decidability of MSOU or its complexity (see Proposition 3.1). Also, the use of a unique first-order constant does not affect the complexity of MSOU. Here we will use a single one called $\flat$. Thus, in instantiations $\lambda y\,.\,t$ for variables X of the problem, the variable y can occur at most once in t. This leads to the specialization where every equation is of the form $f_1(f_2(\dots f_n(\flat)\dots)) \stackrel{?}{=} g_1(g_2(\dots g_n(\flat)\dots))$, where f_i, g_i are unary function symbols or unary variables, and which is rather similar to a word equation.

Example 1. The equation $X(a(X(\flat))) \stackrel{?}{=} a(Y(Y(\flat)))$ has, among others, the solutions $[X \mapsto \lambda x\,.\,a(a(a(x))), Y \mapsto \lambda x\,.\,a(a(a(x)))]$ and $[X \mapsto \lambda x\,.\,a(a(a(\flat))), Y \mapsto \lambda x\,.\,a(x)]$. In the second solution the instantiation of X does not use its argument.

Since all terms are monadic, we will avoid the use of parenthesis in all cases where this is possible and write the terms as words.

1.2. Monadic second-order unification from a word unification perspective. Word unification (WU) is the problem of solving equations on strings. Given a finite alphabet of constants Σ and variables $\mathcal{X}$, a word equation $s \stackrel{?}{=} t$ is defined by a pair of words $s, t \in (\Sigma \cup \mathcal{X})^+$. A solution of $s \stackrel{?}{=} t$ is a substitution of variables by words in Σ^* such that, after replacing, the words obtained from s and t are equal.

In MSOU, apart from Σ and $\mathcal{X}$, we also have a special symbol denoted by $\flat$. A basic MSOU equation $s \stackrel{?}{=} t$ is defined by a pair of words $s, t \in (\Sigma \cup \mathcal{X})^+\flat$. A solution of $s \stackrel{?}{=} t$ is a substitution of variables by either $\lambda x\,.\,w\,x$ or $\lambda x\,.\,w\,\flat$, where $w \in \Sigma^*$ and where we use β-reduction after the substitution. In the first case we say that the instantiation uses its argument. The substitution of a variable X by $\lambda x\,.\,w\,x$ in $w_1\,X\,w_2$, where w_1, w_2 do not contain X, results in $w_1\,w\,w_2$, as in word unification, whereas the substitution of X by $\lambda x\,.\,w\,\flat$ in $w_1\,X\,w_2$ results in $w_1\,w\,\flat$. Therefore, compared to WU in MSOU some part of the original equation can be removed after instantiation. Moreover, the set of solutions of an MSOU equation is wider than the set of solutions of the corresponding WU equation.

Example 2. All solutions of the word equation $X\,a\,X \stackrel{?}{=} a\,Y\,Y$ have the form $[X \mapsto a^n, Y \mapsto a^n]$. The monadic equation $X\,a\,X\,\flat \stackrel{?}{=} a\,Y\,Y\,\flat$ has, apart from solutions of the form $[X \mapsto \lambda x\,.\,a^n\,x, Y \mapsto \lambda x\,.\,a^n\,x]$, other solutions of the form $[X \mapsto \lambda x\,.\,a\,w\,w\,\flat, Y \mapsto \lambda x\,.\,w\,x]$, $[X \mapsto \lambda x\,.\,a\,w\,x, Y \mapsto \lambda x\,.\,w\,a\,a\,w\,\flat]$, $[X \mapsto \lambda x\,.\,x, Y \mapsto \lambda x\,.\,\flat]$, and $[X \mapsto \lambda x\,.\,a\,w\,\flat, Y \mapsto \lambda x\,.\,w\,\flat]$, for any $w \in \Sigma^*$.

MSOU problems can be decided by guessing for every variable whether it uses its argument or not, modifying the equation by dropping symbols to the right of variables that do not use their arguments, and then calling WU (see also Proposition 2.5).

Example 3. The solutions of the MSOU equation $X\,a\,X\,\flat \stackrel{?}{=} a\,Y\,Y\,\flat$ can be found by solving the disjunction of the word equations $X\,a\,X \stackrel{?}{=} a\,Y\,Y$, $X \stackrel{?}{=} a\,Y\,Y$, $X\,a\,X \stackrel{?}{=} a\,Y$, and $X \stackrel{?}{=} a\,Y$.

This simple reduction shows that MSOU is decidable, since WU is decidable [16],

and also that MSOU is in PSPACE, since the reduction is in NP and by using the result that WU is in PSPACE [20]. Since this is the best currently known upper bound for WU, our result that MSOU is NP-complete gives a sharp bound that (currently) cannot be obtained from results on WU.

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

We combine 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 [16, 8, 23] (see Lemma 2.4). This upper bound allows us to represent exponents in linear space. The other key is a result of Plandowski [17, 18] (see Theorem 4.2). He proves that, given two context-free recursion-free grammars with just one rule for every nonterminal symbol (here called *singleton* grammars), we can check if they define the same (singleton) language in polynomial time (in the size of the grammars). This result is used to check that applying a substitution (represented using this kind of grammar) to both sides of an equation will result in the same term. The successful combination requires us to prove a polynomial upper bound for the size increase of the singleton grammars obtained after a series of extensions (see section 4).

1.4. Further contributions. The method presented in this paper appears to be new and powerful for obtaining sharp complexity bounds and efficient algorithms for unification problems.

WU can be seen as a restriction of MSOU with the extra condition that every variable must be instantiated with a λ-term $\lambda x\,.\,t$, where t has exactly one occurrence of the bound variable x. Our results suggests that MSOU is an easier problem—from the complexity point of view—than WU. Moreover, all naive attempts to encode WU as MSOU have failed. The direct application of our method to WU fails, since Lemma 6.8 does not hold for WU; i.e., if all equations are of the form $X \ldots \stackrel{?}{=} Y \ldots$, then there is a trivial solution as MSOU equations, but this syntactic form does not imply any easy solution method for the equations interpreted as word equations.

Context unification (CU) is a variant of SOU where instantiations of second-order variables use their arguments exactly once. Hence, WU is monadic CU. Decidability of CU is currently unknown. During revision of this paper we were able to apply variants of this technique to determine the complexity of a fragment of CU: stratified context unification [12].

Bounded second-order unification (BSOU) is another variant of SOU where, given a positive integer k, instantiations of second-order variables can use their arguments at most k times. Hence, MSOU is also a subproblem of BSOU, because in MSOU instantiations of variables can use their arguments at most once. It is also known that BSOU is decidable [22], which provides another proof of decidability of MSOU, but no tight upper complexity bound. On the other hand, our proof and results suggest an application to BSOU, which has recently (during revision of this paper) resulted in proving a precise upper complexity bound for BSOU [11].

1.5. Paper organization. This paper proceeds as follows. In section 2 we define a *basic* version of the MSOU problem and give some complexity bounds. We will prove in the rest of the paper that this problem is in NP. Then, in section 3 we define the MSOU problem in its most general form as a specialization of SOU, and we prove that it can be NP-reduced to the basic version. We prove some properties of singleton

context-free grammars in section 4, and in section 5 we use them to compact the representation of equations and solutions. We use a graph in order to describe the instantiation of some variable w.r.t. a given solution (section 6). Sometimes, we need to rewrite such graphs (section 7). Based on this graph, we prove that for any size-minimal solution we can represent the values of all variable instantiations using a polynomial-sized singleton grammar (Theorem 7.7). In section 8, we conclude the NP-completeness of MSOU and also of the corresponding matching problem.

2. Basic monadic second-order unification. In this section we present a simplification of the MSOU problem, called the *basic* MSOU problem. As we will see in section 3 (Proposition 3.1), general MSOU can be NP-reduced to this basic case; therefore this will not cause a loss of generality. The simplification consists in (1) considering only unary free variables (notice that, in the general case, the "monadic" restriction affects constants only, not variables) and (2) considering only the function symbols occurring in the equations and a unique (zero-ary) constant, called $\flat$. In this presentation we will limit the use of concepts and notation of the λ-calculus as much as possible. Moreover, we will use words to represent monadic second-order terms. This will make the use of context-free grammars in section 4 more comprehensible.

Let Σ be a finite set of unary function symbols, denoted by lowercase letters $f, g, \ldots$, and let $\mathcal{X}$ be an infinite and denumerable set of unary variables, denoted by uppercase letters $X, Y, \ldots$. Apart from these sets, we also consider a unique constant $\flat$.

Words of $(\Sigma \cup \mathcal{X})^*$ are denoted by lowercase letters $w, u, v, \ldots$, and ε is the empty word. The length of a word w is denoted by $|w|$. Concatenation is juxtaposition. The notation $v \preceq w$ means that the word v is a prefix of the word w.

Monadic terms, denoted by letters $s, t, \ldots$, are defined by the grammar $t ::= \flat \mid f(t) \mid X(t)$, where $f \in \Sigma$ and $X \in \mathcal{X}$. We remove parentheses and represent monadic terms, e.g., $f(X(g(Y(\flat))))$, as words, e.g., $f\,X\,g\,Y\,\flat$. Therefore, terms are redefined as words of $(\Sigma \cup \mathcal{X})^*\,\flat$. The size of a term $t = w\,\flat$, noted $|t|$, is defined by $|t| = |w|$.

Monadic functions, denoted by Greek letters $\varphi, \ldots$, may be of the form $\lambda x\,.\,w\,\flat$ or $\lambda x\,.\,w\,x$, where $w \in (\Sigma \cup \mathcal{X})^*$. In the first case we say that the function *does not use* the argument. In both cases, the size of the function is $|w|$, and x is said to be a *bound variable*, i.e., not *free*.

A *monadic substitution*, denoted by Greek letters $\sigma, \rho, \tau, \ldots$, is a mapping from a finite subset of variables to monadic functions. We represent these mappings as $\sigma = [X_1 \mapsto \varphi_1, \ldots, X_n \mapsto \varphi_n]$, where $Dom(\sigma) = \{X_1, \ldots, X_n\}$ is the *domain* of the substitution. We extend substitutions to functions from monadic terms to monadic terms recursively as follows:

$$\begin{aligned}
&\sigma(\flat) = \flat,\\
&\sigma(f\,w_1) = f\,\sigma(w_1),\\
&\sigma(X\,w_1) = w_2\,\flat \qquad \text{if } \sigma(X) = \lambda x\,.\,w_2\,\flat,\\
&\sigma(X\,w_1) = w_2\,\sigma(w_1) \quad \text{if } \sigma(X) = \lambda x\,.\,w_2\,x.
\end{aligned}$$

The size of a substitution is defined as $|\sigma| = \sum_{X \in Dom(\sigma)} |\sigma(X)|$. 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; hence $Dom(\sigma \circ \rho)$ is finite. Given a set of variables V and a substitution σ, the restriction of σ to the domain V is denoted by $\sigma|_V$. Given a set of variables V, we say that a substitution σ is *more general* w.r.t. V than another substitution ρ, denoted $\sigma \preceq_V \rho$, if there exists a substitution τ such that $\rho(X) = \tau(\sigma(X))$

for all variables $X \in V$, i.e., $\rho = (\tau \circ \sigma)|_V$. (Usually, V will be the set of variables occurring in a set of equations; in this case, we do not mention V if it is clear from the context). This defines a preorder relation on substitutions. An equivalence relation can also be defined as $\sigma \approx_V \rho$ if $\sigma \preceq_V \rho$ and $\rho \preceq_V \sigma$. A substitution σ is said to be *ground* if $\sigma(X)$ does not contain (free occurrences of) variables for all $X \in Dom(\sigma)$. Notice that if σ and τ are ground, then $\sigma \approx_V \tau$ is equivalent to $\sigma|_V = \tau|_V$ (otherwise they are only equivalent modulo variable renaming). We say that a substitution σ *introduces* a constant a (or a variable X) if, for some $Y \in Dom(\sigma)$, $\sigma(Y)$ has an occurrence of a (or X).

DEFINITION 2.1. *A* basic monadic second-order unification problem *(basic MSOU problem) E is a finite set of pairs of monadic terms a.k.a.* monadic equations*, represented as $E = \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\}$.*

The set of variables occurring in E is denoted as $FV(E)$. The *size* of E is the sum of the sizes of its terms s_i and t_i and is denoted as $|E|$. We denote the number of equations of E as $\#Eq(E)$.

DEFINITION 2.2. *A* unifier *of E is a monadic substitution σ, mapping variables of $FV(E)$ to monadic functions and solving all equations: $\sigma(s_i) = \sigma(t_i)$ for $i = 1, \ldots, n$. It is said to be* ground *if $\sigma(s_i)$ and $\sigma(t_i)$ do not contain free occurrences of variables for $i = 1, \ldots, n$.* Most general unifiers *are unifiers that are minimal w.r.t. $\preceq_{FV(E)}$.*

A solution *of E is a ground unifier. A solution σ of E is said to be* size-minimal *if $Dom(\sigma) = FV(E)$ and has minimal size among all solutions of E; i.e., it minimizes $\sum_{X \in FV(E)} |\sigma(X)|$.*

The problem E is said to be unifiable *if it has a unifier, and* solvable *if it has a ground unifier or solution.*

Example 4. Let f and g be unary function symbols and X and Y unary variables. Consider the following basic MSOU problem:

$$\{f\, g\, Y\, X\, \flat \stackrel{?}{=} X\, f\, g\, Y\, \flat\}.$$

It has infinitely many solutions, for instance, $\sigma_1 = [X \mapsto \lambda x\,.\,(f\, g)^n\, x, \quad Y \mapsto \lambda x\,.\,(f\, g)^m x]$ for any $n, m \geq 0$ or $\sigma_2 = [X \mapsto \lambda x\,.\,(f\, g)^{n+1}\, \flat, \quad Y \mapsto \lambda x\,.\,(f\, g)^n\, \flat]$ for any $n \geq 0$. Obviously σ_1 with $n = m = 0$ is a size-minimal solution. Observe also that, interpreting the problem as a WU equation, we can use σ_1 to get the corresponding solution for the word equation because the monadic functions of the solutions use their argument, whereas this is not the case for σ_2.

2.1. The exponent of periodicity bound. The following lemma will provide us with an upper bound on the number of iterations of subwords within solutions.

DEFINITION 2.3. *For a ground substitution σ, its* exponent of periodicity*, denoted as* $\mathrm{eop}(\sigma)$*, is the maximal number $n \in \mathbb{N}$, such that for words u, v, and w over Σ^*, where v is not empty, $\sigma(X) = \lambda y\,.\,u\, v^n\, w\, y$ or $\sigma(X) = \lambda y\,.\,u\, v^n\, w\, \flat$ for some $X \in Dom(\sigma)$.*

We know that any size-minimal ground unifier (i.e., solution) of a set of MSOU equations satisfies the following exponent of periodicity lemma [16, 8, 23, 22].

LEMMA 2.4 (see [22, Lemma 4.1]). *There exists a constant $\alpha \in \mathbb{R}^+$ such that for every basic MSOU problem $\langle \Sigma, E \rangle$ and every size-minimal solution σ we have $\mathrm{eop}(\sigma) \leq 2^{\alpha |E|}$.*

2.2. Some upper and lower complexity bounds. We show the relation of MSOU problems to WU. The NP-reduction of MSOU to WU (and its NP-hardness) allows us to translate any upper bound from WU to MSOU. It does not appear to be

possible to encode WU as an MSOU problem, which provides evidence that MSOU may be an easier problem than WU.

PROPOSITION 2.5. *MSOU is in PSPACE.*

Proof. MSOU is NP-reducible to solvability of basic MSOU problems (Proposition 3.1), and this in turn to WU: Given a basic MSOU problem E, we solve it using WU as follows. It is only necessary to search solutions where $\sigma(X)$, for $X \in FV(E)$, is of the form $\lambda x \,.\, a_1 \dots a_n \, x$ or of the form $\lambda x \,.\, a_1 \dots a_n \, \flat$. Thus, the first step is guessing, for every variable occurring in E, whether it uses its argument or not, i.e., whether it is of the first or second form. Then, we translate E into a set of word equations by first replacing every occurrence of $X \, s$ by $X \, \flat$, when $\sigma(X)$ does not use its argument, and then removing $\flat$ at the end of the terms and interpreting them as words. Now, we can apply an algorithm solving WU.

This nondeterministic reduction is correct, since if E is solvable as a basic MSOU problem, then the resulting word equations are solvable (for the convenient guessing). It is easy to see that the converse is also true.

A theorem of Plandowski [20] showing that WU is in PSPACE now implies that MSOU is in PSPACE. ∎

It is well known that MSOU is NP-hard [23]. We show that this also holds for monadic second-order matching. The proof gives a good feeling of what one can express in MSOU.

THEOREM 2.6. *Basic monadic second-order matching is NP-hard.*

Proof. We use the ONE-IN-THREE-SAT problem, which is known to be NP-complete [4]. An instance of the ONE-IN-THREE-SAT problem consists of the following: a set of propositional variables $p_1, \dots, p_n$ and m clauses $C_i = \{q_{i,1}, q_{i,2}, q_{i,3}\}$, where $q_{i,j} \in \{p_1, \dots, p_n\}$ for every $i = 1, \dots, m$ and $j = 1, 2, 3$. A solution is an assignment of the truth values *true* and *false* to the propositional variables, such that in every clause exactly one variable is assigned the value *true*.

We construct a basic MSOU problem where equations have ground right-hand sides and where $\Sigma = \{a, b, c\}$. For every $i = 1, \dots, n$ let X_i, Y_i be unary second-order variables. For $i = 1, \dots, n$, we use the equations

$$\begin{aligned} X_i \, Y_i \, b \flat &\stackrel{?}{=} a \, b \flat, \\ X_i \, Y_i \, c \flat &\stackrel{?}{=} a \, c \flat. \end{aligned}$$

These equations enforce that for all i either X_i is instantiated by $\lambda x \,.\, x$ or by $\lambda x \,.\, a \, x$, and similarly for Y_i, and that there are at most two possibilities for the instantiation of the pair X_i, Y_i for every i: The assignment $[X_i \mapsto \lambda x \,.\, a \, x, Y_i \mapsto \lambda x \,.\, x]$ is interpreted as *true*, and the assignment $[X_i \mapsto \lambda x \,.\, x, Y_i \mapsto \lambda x \,.\, a \, x]$ as *false*. Every clause $C = \{p_i, p_j, p_k\}$ is encoded as an equation

$$X_i \, X_j \, X_k \, b \flat \stackrel{?}{=} a \, b \flat.$$

Now it is obvious that the set of constructed equations has a unifier, if and only if the instance of ONE-IN-THREE-SAT is solvable. The equations form a monadic second-order matching problem and can be generated in linear time. Hence, the claim follows. ∎

3. General monadic second-order unification. In the rest of this paper we will prove the complexity estimation for *basic* MSOU problems. In this section we will argue that the restriction to "basic" does not compromise generality. The main claim is that there is a nondeterministic reduction from (general) MSOU problems to

basic MSOU problems that can be done in nondeterministic polynomial time. As a subcase of HOU, the definition of the problem in all its generality requires the use of the λ-calculus. However, we will limit its use to this section, which can be skipped by those readers not familiar with the λ-calculus.

We will use the standard notation and definitions of the simply typed λ-calculus, and we inherit the definitions of the previous section, unless we explicitly overwrite them here.

We consider only one (first-order) *base type* o and all the *second-order types* constructed from it, i.e., the ones described by the syntax $\tau ::= o \to o \mid o \to \tau$, with the usual convention that $\to$ is associative to the right. Hence, every type is o which has order one, or it is of the form $o \to o \to \cdots \to o$ and has order two.[1]

We consider a *signature* $\Sigma = \Sigma_0 \cup \Sigma_1$ of constants, denoted by $a, b, c, \ldots$, where constants of Σ_0 have type o and constants of Σ_1 have type $o \to o$. There is also a *set of variables* $\mathcal{X} = \bigcup_{i \geq 0} \mathcal{X}_i$, denoted by $x, y, z, \ldots$, where every set $\mathcal{X}_i$ contains infinitely and denumerable many variables with type $\underbrace{o \to \cdots \to o}_{i+1}$. Constants of Σ_0 and variables of $\mathcal{X}_0$ are first-order typed and are said to have arity zero, whereas those of Σ_i and $\mathcal{X}_i$, for $i > 0$, are second-order typed and have arity i.

Well-typed *terms* over the signature Σ and the set of variables $\mathcal{X}$ are built as usual in the simply typed λ-calculus:

(i) any constant $a \in \Sigma_i$ and any variable $x \in \mathcal{X}_i$ is a well-typed term of type $o \to^{i+1} \cdots \to o$,

(ii) if t is a well-typed term of type τ, and $x \in \mathcal{X}_0$, then $(\lambda x \,.\, t)$ is also a well-typed term of type $o \to \tau$, and

(iii) if s of type $o \to \tau$ and t of type o are well-typed terms, then $(s\, t)$ is also a well-typed term of type τ.

General second-order terms are defined using a signature $\Sigma = \bigcup_{i \geq 0} \Sigma_i$ and a set of variables $\mathcal{X} = \bigcup_{i \geq 0} \mathcal{X}_i$, where constants of Σ_i and variables of $\mathcal{X}_i$ have arity i. Therefore, monadic terms are second-order terms built without using constants of arity greater than one. Notice that there is no restriction on the arity of variables, whereas in basic MSOU we consider only unary variables.

Any term of type $\underbrace{o \to \cdots \to o}_{n+1}$ is said to have *arity* n. It is called of *first-order type* when $n = 0$, and of *second-order type* when $n > 0$. Hence, the arity of a term or of a symbol determines its type, and we will usually specify the arity instead of the type. When we say *normal form* we mean η-long β-reduced normal form, defined as usual. Since 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 contain λ-abstractions only in outermost positions. The set of *free variables* of a term t is denoted by $FV(t)$. A term without occurrences of free variables is said to be *closed*. The *size* of a term t is denoted $|t|$ and defined as its number of symbols (variables and constants), when written in normal form.

Second-order substitutions are functions from terms to terms, defined as usual. The application of a substitution σ to a term t is written as $\sigma(t)$, where we implicitly assume that $\sigma(t)$ (after some β-reductions) is written in normal form. For any substitution σ, the set of variables x, such that $\sigma(x) \neq_{\beta\eta} x$, is finite and is called the *domain* of the substitution and denoted $Dom(\sigma)$. A substitution σ can be represented

[1] This also means that we do not allow symbols or expressions of third- or a higher-order type.

as $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$, where $x_i \in Dom(\sigma)$ and t_i has the same type as x_i and satisfies $t_i = \sigma(x_i)$.

An instance of the *(general) MSOU problem* is a pair $\langle \Sigma, E \rangle$, where $\Sigma = \Sigma_0 \cup \Sigma_1$ is a monadic signature and E is a *set of equations* $E = \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\}$, where s_i and t_i are normalized first-order terms over Σ, i.e., terms not containing λ-abstractions.

Note that, in monadic signatures, closed second-order terms are of two forms: $\lambda x_1 . \cdots . \lambda x_n . (a_1 (\ldots (a_m x_i) \ldots))$ or $\lambda x_1 . \cdots . \lambda x_n . (a_1 (\ldots (a_m b) \ldots))$ for unary constants a_i and a zero-ary constant b. Since solutions σ map variables to closed terms, in the first case we say that $\sigma(x)$ *uses one* of its arguments, and if x is unary, we say that $\sigma(x)$ *uses* its argument, and in the second case we say that $\sigma(x)$ is constant or that it *ignores* its argument.

Unifiability of MSOU problems does not depend on the signature, as long as the symbols of the equations are in the signature. However, its solvability has some dependence on the signature; more precisely it depends on the existence of at least one first-order constant.

In MSOU, as in general SOU, we can prove that *for every unifiable set of equations E, and every most general unifier σ, all constants occurring in σ also occur in E.* The proof is by contradiction. If there is a most general unifier using constants not occurring in the equations, we can replace these constants by fresh variables, obtaining a more general unifier. The statement does not hold for variable occurrences in unifiers. Even if the set of equations is built from unary variables and unary constants, most general unifiers may introduce fresh n-ary variables with $n \geq 2$. For instance, the set of equations $\{(x\, a) \stackrel{?}{=} (y\, b)\}$ has only one most general second-order unifier $[x \mapsto \lambda x . ((z\, x)\, b), y \mapsto \lambda x . ((z\, a)\, x)]$ that introduces a binary variable z.

PROPOSITION 3.1. *Unifiability of MSOU problems is NP-reducible to solvability of basic MSOU problems.*

Proof. The reduction is done in three steps.

(i) First, we reduce unifiability of MSOU problems to solvability, provided that Σ_0 contains at least one constant. In other words, for any signature Σ, any set of equations E over Σ, and first-order constant b, $\langle \Sigma, E \rangle$ is unifiable, if and only if $\langle \Sigma \cup \{b\}, E \rangle$ is solvable.

For the *if* direction, assume given a solution σ. If $b \notin \Sigma$, then $b \notin E$ and we can replace b by a fresh first-order variable x_b everywhere in σ and obtain a (maybe nonground) unifier of $\langle \Sigma, E \rangle$. For the *only if* part, assume given a unifier σ. Then we can define a substitution ρ such that, for every $n \geq 0$, every n-ary variable $x \in FV(\sigma(E))$ is instantiated by $\lambda x_1 . \cdots . \lambda x_n . b$. Then $\rho \circ \sigma$ is a solution of $\langle \Sigma \cup \{b\}, E \rangle$.

(ii) Second, we prove that solvability of MSOU problems is reducible in polynomial time to solvability of MSOU problems with just one first-order constant.

Assume given an MSOU problem $\langle \Sigma, E \rangle$. If $\Sigma_0 = \emptyset$, then the problem is unsolvable. Otherwise, we reduce the problem as follows. We transform the signature Σ into a new signature $\Sigma' = \Sigma'_1 \cup \Sigma'_0$, where the set of unary constants is $\Sigma'_1 = \Sigma_1 \cup \Sigma_0$; i.e., the former zero-ary constants are unary ones in the new signature, and the set of zero-ary constants is $\Sigma'_0 = \{\flat\}$. We replace every first-order constant occurrence a in the equations E by $(a\, \flat)$, obtaining a set of equations E' over Σ'. We will see that any solution σ of $\langle \Sigma, E \rangle$ can be translated into a solution σ' of $\langle \Sigma', E' \rangle$, and vice versa.

Any solution σ of $\langle \Sigma, E \rangle$ can be translated into a solution of $\langle \Sigma', E' \rangle$ using the same transformation as for the equations. To show the other direction, let σ' be

a solution of $\langle \Sigma', E' \rangle$. Before retranslating σ' we transform it into σ'' as follows: For every x in $Dom(\sigma')$, we remove every occurrence of symbols $a \in \Sigma_0$ in $\sigma'(x)$ which is not of the form $(a\,\flat)$; i.e., we replace $(a\,s)$ by s when $s \neq \flat$, until this replacement is no longer applicable. The translation from E to E' ensures that in E' every occurrence of all x is in subterms of the form $((\ldots(x\,s_1)\ldots)\,s_n)$, where $s_i \neq \flat$. Looking at the different cases, the removal of symbols takes place only within instantiations of variables and does not conflict with the constants occurring in E. Hence, σ'' is a solution of $\langle \Sigma', E' \rangle$ and can be immediately retranslated to a solution of $\langle \Sigma, E \rangle$.

The translations of signature, set of equations, and solutions in either direction are polynomial.

(iii) Third, we prove that we can go a step further, assuming that all variables are unary: We show that solvability of MSOU, where $\Sigma_0 = \{\flat\}$, is nondeterministically reducible in polynomial time to solvability of MSOU with the same signature, and where all variables occurring in the equations are unary: $\mathcal{X}_n = \emptyset$ for all $n \neq 1$.

Given an MSOU problem $\langle \Sigma, E \rangle$, where $\Sigma_0 = \{\flat\}$, we consider substitutions ρ that instantiate every first-order variable $x \in FV(E)$ by $(x'\,\flat)$, where x' is a fresh unary variable, and every n-ary variable $y \in FV(E)$ (with $n \geq 2$) by either $\lambda x_1 . \cdots . \lambda x_n . (y'\,x_i)$, where $1 \leq i \leq n$, or $\lambda x_1 . \cdots . \lambda x_n . (y'\,\flat)$, where y' is a fresh unary variable, and the selection is nondeterministic. Obviously, if for some ρ as given above, $\langle \Sigma, \rho(E) \rangle$ is solvable, so is $\langle \Sigma, E \rangle$.

Conversely, if $\langle \Sigma, E \rangle$ is solvable, we prove that for some ρ satisfying the specified conditions, $\langle \Sigma, \rho(E) \rangle$ is also solvable. Mainly, we prove that there is some ρ as specified above and a substitution τ with $\sigma(x) = \tau \circ \rho(x)$ for all $x \in FV(E)$; thus τ solves $\langle \Sigma, \rho(E) \rangle$.

Since all constants have arity at most one and solutions are ground, instantiations $\sigma(x)$ of n-ary variables, for $n \geq 2$, use at most one of their arguments: $\sigma(x) = \lambda x_1 . \cdots . \lambda x_n . t$, where t has a unique occurrence of some x_i, or none. Therefore, we can take $\rho(x) = \lambda x_1 . \cdots . \lambda x_n . (x'\,x_i)$ or $\rho(x) = \lambda x_1 . \cdots . \lambda x_n . (x'\,\flat)$. Instantiations of first-order variables use at least the first-order constant $\flat$; therefore they can also be replaced by a fresh unary variable applied to this constant. It is obvious how to construct the solution τ of $\langle \Sigma, \rho(E) \rangle$ from σ.

Finally, note that we can compute the substitution ρ in nondeterministic polynomial time on the size of E because for any nonunary variable x, we can guess whether x uses one of its arguments and, in the positive case, which argument $\rho(x)$ uses. ∎

The following lemma states that if there is just one zero-ary constant in the signature, the set of size-minimal solutions is independent from the rest of the signature. Therefore, since we are dealing with size-minimal solutions of basic MSOU problems, in the next sections, we will not specify the signature.

LEMMA 3.2. *For any MSOU problem $\langle \Sigma, E \rangle$, where $\Sigma_0 = \{\flat\}$, every size-minimal solution σ contains only constants that also occur in E or are equal to $\flat$.*

Proof. Suppose that a size-minimal solution of $\langle \Sigma, E \rangle$ introduces a constant a that does not occur in E. Then, we could generate a solution with a strictly smaller size by replacing all subterms of the form $(a\,s)$ by $\flat$. This would contradict minimality. ∎

4. Singleton context-free grammars. In this section we prove some properties of context-free grammars. They will be used to *compactly* represent solutions of MSOU problems. In particular, we will use singleton context-free grammars that define languages with just one word.

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 *nonterminal* symbols (contrary to the standard conventions and in order to avoid confusion between free variables (unknowns) and nonterminal symbols, all terminal and nonterminal symbols are denoted by lowercase letters), P is a finite set of rules, and $s \in N$ is the *start symbol*. In fact, we will not distinguish any particular start symbol, and we will represent a CFG as a 3-tuple (Σ, N, P). Moreover, we will use Chomsky grammars with at most two symbols on the right-hand sides of the rules.

DEFINITION 4.1. *We say that a CFG $G = (\Sigma, N, P)$* generates *a word $v \in \Sigma^*$ if there exists a nonterminal symbol $a \in N$ such that v belongs to the language defined by (Σ, N, P, a). In such a case, we also say that a* generates *v.*

We say that a CFG is singleton *if it is in Chomsky normal form, i.e., the right-hand sides of the productions consist of words of length at most* 2*, it is not recursive, and there exists just one production for each nonterminal symbol. Then, every nonterminal symbol $a \in N$ generates just one word, denoted w_a, and we say that a* generates *w_a. In general, for any sequence $\alpha \in (\Sigma \cup N)^*$, $w_\alpha \in \Sigma^*$ denotes the word generated by α.*

Plandowski [17, 18] defines singleton grammars, but he calls them *grammars defining set of words*. Note that so-called straight-line programs are an equivalent device [7]. Plandowski proves the following result.

THEOREM 4.2 (see [18, Theorem 33]). *The word equivalence problem for singleton CFGs is defined as follows: Given a singleton grammar and two nonterminal symbols a and b, decide whether $w_a = w_b$. This problem can be solved in polynomial worst-case time in the size of the grammar.*

Recent work [15] claims that this can be done in cubic time.

For nonrecursive grammars we define their depth as follows. The usage of both size and depth of the grammar is necessary for a good estimation, since they reflect balancing conditions for a singleton grammar seen as a tree. Using only a single measure leads to unsatisfactory upper bounds (see Remark 1 in section 8).

DEFINITION 4.3. *Let $G = (\Sigma, N, P)$ be a nonrecursive CFG. For any terminal symbol $a \in \Sigma$ we define $depth(a) = 0$, and for any nonterminal symbol $a \in N$ we define*

$$depth(a) = \max\{depth(b) + 1 \mid a \to \alpha \in P,\ b \text{ occurs in } \alpha\}.$$

We define the depth *of G as $depth(G) = \max\{depth(a) \mid a \in N\}$.*

Given a Chomsky CFG G, we define the size *of G, noted $|G|$, as the number of its rules.*

We say that $G' = (\Sigma', N', P')$ is an extension of $G = (\Sigma, N, P)$, denoted as $G' \supseteq G$, if and only if $\Sigma' \supseteq \Sigma$, $N' \supseteq N$, and $P' \supseteq P$, where we require only $\Sigma' = \Sigma$. We can extend a singleton grammar in order to generate concatenation, exponentiation, and prefixes and suffixes of words already generated by the grammar. We use these extension operations in the next sections to build the grammar defining some solution of the unification problem. The following three lemmas state how the size and the depth of the grammar are increased with these transformations. Since in the final step of this paper a grammar of polynomial size is guessed and checked in polynomial time, we only need the existence of polynomial-sized grammars. Thus we do not care about the algorithmic complexity of constructing these grammars.

LEMMA 4.4 (concatenation). *Let G be a singleton grammar generating the words $v_1, \ldots, v_n$ for $n \geq 1$. Then there exists a singleton grammar $G' \supseteq G$ that generates*

the word $v_1 \dots v_n$ and satisfies

$$\begin{array}{lcl} |G'| & \leq & |G| + n - 1, \\ \mathit{depth}(G') & \leq & \mathit{depth}(G) + \lceil \log n \rceil. \end{array}$$

Proof. Let a_i be the nonterminal symbol generating v_i, for any $i = 1, \dots, n$. We define G' by adding a set of rules to G of the form

$$b_{i,j} \to b_{i,\, \lfloor \frac{i+j}{2} \rfloor}\, b_{\lfloor \frac{i+j}{2} \rfloor + 1,\, j},$$

where $1 \leq i \leq j \leq n$ and $b_{i,i}$ is a_i. Then, $b_{1,n}$ generates $v_1 \dots v_n$, and to generate it we need only to add $n-1$ of such rules. The depth is increased by at most $\lceil \log n \rceil$. ∎

LEMMA 4.5 (exponentiation). *Let G be a singleton grammar generating the word v. For any $n \geq 1$, there exists a singleton grammar $G' \supseteq G$ that generates the word v^n and satisfies*

$$\begin{array}{lcl} |G'| & \leq & |G| + 2\,\lfloor \log n \rfloor, \\ \mathit{depth}(G') & \leq & \mathit{depth}(G) + \lfloor \log n \rfloor + 1. \end{array}$$

Proof. Let a be the nonterminal symbol generating v, $m = \lfloor \log n \rfloor$, and let $n = k_0 2^0 + k_1 2^1 + \cdots + k_m 2^m$ be a binary representation satisfying $k_i \in \{0, 1\}$. We add the following set of rules to G:

$$\begin{array}{lcl} a_1 & \to & a\,a, \\ a_2 & \to & a_1\,a_1, \\ & \cdots & \\ a_m & \to & a_{m-1}\,a_{m-1}, \end{array} \qquad \begin{array}{lcl} b_0 & \to & \begin{cases} a & \text{if } k_0 = 1, \\ \varepsilon & \text{if } k_0 = 0, \end{cases} \\ b_1 & \to & \begin{cases} a_1\,b_0 & \text{if } k_1 = 1, \\ b_0 & \text{if } k_1 = 0, \end{cases} \\ & \cdots & \\ b_m & \to & \begin{cases} a_m\,b_{m-1} & \text{if } k_m = 1, \\ b_{m-1} & \text{if } k_m = 0. \end{cases} \end{array}$$

Then, the nonterminal symbol b_m generates v^n, and it is easy to see that this grammar satisfies the bounds stated by the lemma. ∎

LEMMA 4.6 (prefixes and suffixes). *Let G be a singleton grammar generating the word v. For any prefix or suffix v' of v, there exists a singleton grammar $G' \supseteq G$ that generates v' and satisfies*

$$\begin{array}{lcl} |G'| & \leq & |G| + \mathit{depth}(G), \\ \mathit{depth}(G') & = & \mathit{depth}(G). \end{array}$$

Proof. Let a be the nonterminal symbol generating v. By induction on $\mathit{depth}(a)$, we will prove a stronger result: For any prefix v' of w_a, there exists a grammar G' generating v' and satisfying $|G'| \leq |G| + \mathit{depth}(a)$ and $\mathit{depth}(G') = \mathit{depth}(G)$.

The base case is trivial since $\mathit{depth}(a) = 0$ implies that a is a terminal symbol, and $v' = v$ or v' is empty. For the induction case, assume that $v' \neq v$; otherwise we are done. Let $a \to \alpha$ be the rule for a. Note that $|\alpha| \leq 2$. There exists a prefix $\beta\,b$ of α, where b is a nonterminal, such that w_β is a prefix of v' and v' is a prefix of $w_{\beta\,b}$; i.e., $v' = w_\beta\,v''$, where v'' is a prefix of w_b. By induction hypothesis, there exists a grammar $G'' \supseteq G$ deriving v'' from some b' with the same depth as G and size $|G''| \leq |G| + \mathit{depth}(b) \leq |G| + \mathit{depth}(a) - 1$. We add $a' \to \beta\,b'$ to get the grammar G'

from G'' such that $w_{a'} = v'$. Notice that $|G'| = |G''| + 1$, and $depth(G') = depth(G'')$ because $depth(b') \leq depth(b)$ implies $depth(a') \leq depth(a)$.

For suffixes the proof is very similar. □

We illustrate Lemmas 4.4, 4.5, and 4.6 by means of Example 5.

Example 5. Let G be the grammar defined by the productions $\{c_1 \to c_2\, c_3,$ $c_2 \to c_3\, c_4,\ c_3 \to ff,\ c_4 \to gg\}$. The words generated by the nonterminals c_1, c_2, c_3, and c_4 of G are $v_1 = ffggff$, $v_2 = ffgg$, $v_3 = ff$, and $v_4 = gg$, respectively.

(i) *Concatenation.* We first show how to build the word

$$v_1 v_2 v_3 v_4 = ffggffffggffgg$$

using the techniques of the proof of Lemma 4.4. Following the definitions of $b_{i,j} \to b_{i,\lfloor \frac{i+j}{2} \rfloor}\, b_{\lfloor \frac{i+j}{2} \rfloor + 1,\, j}$, and $b_{i,i} = c_i$, we extend G with the following rules:

$$\begin{array}{lcl} b_{1,4} & \to & b_{1,2}\, b_{3,4}, \\ b_{1,2} & \to & c_1\, c_2, \\ b_{3,4} & \to & c_3\, c_4. \end{array}$$

Then, $b_{1,4}$ generates $v_1 v_2 v_3 v_4$.

(ii) *Exponentiation.* Now we show how to build the word $(v_1)^5$ according to the techniques of the proof of Lemma 4.5. We have $5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$; hence we extend G with the following rules:

$$\begin{array}{lcllcl} a_1 & \to & c_1\, c_1, & b_0 & \to & c_1, \\ a_2 & \to & a_1\, a_1, & b_1 & \to & b_0, \\ & & & b_2 & \to & a_2\, b_1. \end{array}$$

Then b_2 generates $(v_1)^5$.

(iii) *Prefix.* Finally we show how to build the word prefix $v' = ffg$ of v_1 using the techniques of the proof of Lemma 4.6. We extend G with the following rules:

$$\begin{array}{lcl} c'_1 & \to & c'_2, \\ c'_2 & \to & c_3\, c'_4, \\ c'_4 & \to & g. \end{array}$$

Then c'_1 generates v'.

5. Compact representations. In this section we use singleton grammars to compact the representation of solutions of basic MSOU problems. We go a step further and also compact the representation of equations, allowing the use of nonterminal symbols of a singleton grammar to represent large words also in the equations.

DEFINITION 5.1. *Let Σ be a signature of unary symbols, and let $\mathcal{X}$ be a set of unary variables.*

A compact representation *of a basic MSOU problem E is a pair $\langle E', G\rangle$, where $G = \langle \Sigma, N, P\rangle$ is a singleton CFG and E' is a set of equations of the form $\{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\}$, where $s_i, t_i \in (\Sigma \cup \mathcal{X} \cup N)^* \flat$, for $i = 1, \ldots, n$, such that when replacing in E' every nonterminal symbol a by the word w_a that it generates, it results in the set of equations E.*

A compact representation *of a monadic substitution σ is a pair $\langle \sigma', G\rangle$, where $G = \langle \Sigma, N, P\rangle$ is a singleton CFG and σ' is a mapping from variables X to terms of the form $\lambda x\,.\,\alpha\flat$ or $\lambda x\,.\,\alpha\, x$, where $\alpha \in (\Sigma \cup N)^*$, such that, after replacing every nonterminal symbol by the word it represents, we obtain σ.*

We say that $\langle \tau, G' \rangle$ *is a* compacted solution *of* $\langle E, G \rangle$ *if the substitution represented by* $\langle \tau, G' \rangle$ *is a solution of the set of equations represented by* $\langle E, G \rangle$*, where* G' *is an extension of* G*.*

Notice that nonterminal symbols derive into sequences of unary function symbols, that we do not consider first-order variables, and that $\flat$ is the only constant. Words of $(\Sigma \cup \mathcal{X} \cup N)^*$ are denoted by Greek letters $\alpha, \beta, \ldots$.

Example 6. Let $\Sigma = \{f\}$ and $N := \{a, c, d\}$. Consider the compacted equations $\langle E, G \rangle$ defined by

$$\begin{array}{rcl} E & = & \{a\, X\, X\, f\, \flat \stackrel{?}{=} Y\, Y\, Y\, \flat\}, \\ G & = & \{a \to c\, c,\ c \to f\, f\}. \end{array}$$

Then, the pairs $\langle \sigma_1, G_1 \rangle$ and $\langle \sigma_2, G_2 \rangle$, defined by

$$\begin{array}{rcl} \sigma_1 & = & [X \mapsto \lambda x\,.\, c\, x,\ Y \mapsto \lambda x\,.\, d\, x], \\ G_1 & = & \{a \to c\, c,\ c \to f\, f,\ d \to c\, f\} \end{array} \quad \text{and} \quad \begin{array}{rcl} \sigma_2 & = & [X \mapsto \lambda x\,.\, e\, \flat,\ Y \mapsto \lambda x\,.\, a\, \flat], \\ G_2 & = & \{a \to c\, c,\ c \to f\, f,\ e \to \varepsilon\}, \end{array}$$

are compacted representations of solutions of $\langle E, G \rangle$. The first solution is not size-minimal. The second solution is size-minimal, but it is not a most general unifier. In fact, the second is an instantiation of the most general unifiers $[X \mapsto \lambda x\,.\, Z\, x,\ Y \mapsto \lambda x\,.\, a\, Z\, Z\, f\, \flat]$ and $[X \mapsto \lambda x\,.\, Z\, a\, Z\, a\, Z\, \flat, Y \mapsto \lambda x\,.\, a\, Z\, x]$.

We generalize the basic MSOU problem in the sense that, given some compacted equations $\langle E, G \rangle$, we will try to find a compacted solution $\langle \sigma, G' \rangle$. Moreover, the grammar G' used to represent the solution will be an extension of the grammar G given to represent the equations.

Notice that solvability of a set of monadic equations and solvability of compact equations are, w.r.t. decidability, equivalent problems. With respect to their complexity, we will prove that solvability of compact equations can be decided in NP-time. This implies that solvability of MSOU is also in NP (since $\langle E, \emptyset \rangle$ is a trivial compact representation of E).

Notice that the straightforward translation of a compacted set of equations into the set of monadic equations that they represent may exponentially increase the size of the equations. Using another translation, we can show that solvability of MSOU problems and solvability of compacted MSOU problems are polynomially equivalent.

PROPOSITION 5.2. *Given a compacted set of equations* $\langle E, G \rangle$*, there is a P-time translation into a basic MSOU problem* E'*, such that* $\langle E, G \rangle$ *is solvable if and only if* E' *is solvable.*

Proof. For every nonterminal a in G, define a fresh unary variable X_a. For every production $a \to b\, c$ of the grammar, where $a, b, c \in N$, define a set of two equations $E_a = \{X_a\, \flat \stackrel{?}{=} X_b\, X_c\, \flat,\ X_a\, f\, \flat \stackrel{?}{=} X_b\, X_c\, f\, \flat\}$, where $f \in \Sigma$ and $X_a, X_b, \ldots$ are fresh variables. This is similar for the other kinds of rules, where the right-hand sides are shorter. The terminal symbols are not translated. For instance, for $a \to b\, c$, where $a \in N$ and $b, c \in \Sigma$, $E_a = \{X_a\, \flat \stackrel{?}{=} b\, c\, \flat, X_a\, f\, \flat \stackrel{?}{=} b\, c\, f\, \flat\}$. Then, $E' = \hat{E} \cup \bigcup_{a \in N} E_a$, where $\hat{E}$ is the translation of the equations E by replacing nonterminals a with the corresponding unary variable X_a. The size of E' is smaller than $|E| + 12\,|G|$, and it can be constructed in polynomial time.

The equations in $\bigcup_{a \in N} E_a$ enforce that, for every nonterminal a, $\sigma(X_a)$ uses its argument, and therefore $\sigma(X_a) = \lambda x\,.\, w_a\, x$. Now it is easy to see that $\langle E, G \rangle$ is solvable if and only if E' is solvable. ∎

6. The graph of surface dependencies. In this section we define graphs of surface dependencies. The purpose of these graphs is to support constructing the compact representation of a minimal solution σ of a compacted set of equations and estimating the size of this representation. Later on this will be used to show that only a polynomial-sized representation has to be guessed in order to check solvability. Observe that we only impose a bound on the size of the representation and do not care about the complexity of finding such a representation. In our proof, we start from the compact representation $\langle E, G\rangle$ of some basic MSOU problem and a given solution σ. Then, we find a variable X whose instantiation can be compactly represented. This is done by extending the grammar to $G' \supseteq G$. Then, we repeat the process starting from the same equation with the variable already instantiated. Observe that G' (apart from the instantiation of the X) is able to generate all the words represented by G. This iteration describes a proof by induction, not the unification algorithm. It could be interpreted as a nondeterministic unification procedure, however, with the restriction of finding a size-minimal solution, and moreover, without a guarantee of being in NP.

There are cases in which for some variable X of the problem, its instantiation $\sigma(X)$ is immediately given or immediately constructible from the "surface" of the equations. We identify two such cases: when σ has a *small component* (Lemma 6.6) and when the graph contains a *cycle* (Lemma 6.12). We also identify three situations which ensure that any size-minimal solution has small components: when the graph has a constant equation (Lemma 6.7), when the graph has no edges (Lemma 6.8), and when there are strong divergences (Lemma 6.10). In the rest of the cases, it becomes necessary to rewrite the graph to obtain a new graph that describes the instantiation of some variable. This graph rewriting process will be described in section 7.

The graph of surface dependencies is defined only for *simplified* equations, where a simplified equation is defined as follows.

DEFINITION 6.1. *Given a compacted set of equations $\langle E, G\rangle$, we say that they are* simplified *if E does not contain equations of the following forms (symmetric cases omitted):*

(i) $a\,s \stackrel{?}{=} b\,t$, *where* $a, b \in \Sigma \cup N$,
(ii) $s \stackrel{?}{=} a\,b\,t$, *where* $a, b \in \Sigma \cup N$, *or*
(iii) $s \stackrel{?}{=} a\,t$, *where* $a \in N$ *and* $w_a = \varepsilon$.

Note that simplified equations are of the forms (symmetric cases omitted) $X\,s \stackrel{?}{=} a\,\flat$, $X\,s \stackrel{?}{=} a\,Y\,t$, $X\,s \stackrel{?}{=} \flat$ (flexible-rigid), or $X\,s \stackrel{?}{=} Y\,t$ (flexible-flexible), where a is a nonterminal. Note also that solvable equations without variables have the form $\alpha\,\flat \stackrel{?}{=} \beta\,\flat$, where α, β are in $(\Sigma \cup N)^*$, satisfy $w_\alpha = w_\beta$, and are not simplified.

We describe a simplification algorithm in the proof of the following lemma. This algorithm will be used as a subroutine in the proof of Theorem 7.7. Notice that it can increase the size of the associated grammar as stated in the lemma.

LEMMA 6.2 (simplification). *Given the solvable and compacted set of equations $\langle E, G\rangle$, there exists a simplified and compacted set of equations $\langle E', G'\rangle$ with the same solutions, such that*

$$\begin{aligned} |G'| &= |G| + \mathcal{O}(|E|(depth(G) + \log|E|)), \\ depth(G') &= depth(G) + \mathcal{O}(\log|E|), \end{aligned}$$

and the number of equations, number of variables, and number of occurrences of variables in E' are not greater than the corresponding numbers for E.

Proof. For all equations $s_i \stackrel{?}{=} t_i$ of E, let α_i, β_i be the longest prefixes of s_i and t_i, respectively, that do not contain variables or $\flat$; i.e., $s_i = \alpha_i\,s'_i$, and $t_i = \beta_i\,t'_i$, where

s'_i and t'_i have a variable or $\flat$ in the head. Let v_i be the word satisfying $w_{\alpha_i} = w_{\beta_i} v_i$ or $w_{\alpha_i} v_i = w_{\beta_i}$.

Using Lemma 4.4, construct an extension of the grammar with a nonterminal for the word $\alpha_1 \beta_1 \dots \alpha_n \beta_n$. For every $i = 1, \dots, n$, since v_i is a suffix of some prefix of this word, use Lemma 4.6 to prove that there exists another extension of the grammar that generates v_i. This last process will be repeated at most $2\,\#Eq(E)$ times to obtain a new grammar G'. This ensures that $depth(G') \leq depth(G) + \lceil \log |E| \rceil$ and $|G'| \leq |G| + |E| + 2\,\#Eq(E)\,(depth(G) + \lceil \log |E| \rceil)$. This implies the estimations given in the lemma.

Now, we construct E' from E as follows. For every $i = 1, \dots, n$,

(i) if $s'_i = t'_i = \flat$, then remove the equation from E;

(ii) if $w_{\alpha_i} = w_{\beta_i}$, then replace $s_i \stackrel{?}{=} t_i$ in E by $s'_i \stackrel{?}{=} t'_i$;

(iii) if w_{α_i} is a prefix of w_{β_i}, then replace $s \stackrel{?}{=} t$ in E by $s'_i \stackrel{?}{=} b\,t'_i$, where b is the nonterminal of G' generating v_i; and

(iv) proceed similarly if w_{β_i} is a prefix of w_{α_i}.

Notice that the case where neither w_{α_i} is a prefix of w_{β_i} nor w_{β_i} is a prefix of w_{α_i} is not possible for solvable equations. Notice also that, if we simplify equations one by one, generating a suffix of a prefix of α_i or of β_i each time, we would get a worse estimation. □

DEFINITION 6.3. *A* constant equation *is a simplified and compacted equation of the form* $X\,t \stackrel{?}{=} a\flat$ *or* $X\,t \stackrel{?}{=} \flat$, *where* $t \in (\Sigma \cup \mathcal{X} \cup N)^* \flat$ *is a compacted term and* $a \in N$ *is a nonterminal symbol.*

We define the graph of surface dependencies only for solvable, simplified, and compacted sets of equations.

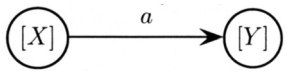
DEFINITION 6.4. *Let* $\langle E, G \rangle$ *be a solvable, simplified, and compacted set of equations. Let* $\approx$ *be the minimal equivalence relation satisfying* $X \approx Y$ *whenever* E *contains an equation of the form* $X\,s \stackrel{?}{=} Y\,t$. *This defines a partition on* $FV(E)$.

The graph of surface dependencies *of* $\langle E, G \rangle$ *is a labeled directed multigraph*[2] *defined as follows:*

Nodes: *The nodes are the* $\approx$*-equivalence classes of variables and the empty set* $\emptyset$.

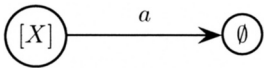
Edges: *There are two cases:*

(i) *For every equation of the form* $X\,s \stackrel{?}{=} a\,Y\,t$, *where* $X, Y \in \mathcal{X}$ *are variables and* $a \in N$ *is a nonterminal symbol, there is an edge*



Simplification yields that $w_a \neq \varepsilon$ *for the edge label* a.

(ii) *For every constant equation* $X\,s \stackrel{?}{=} a\flat$, *where* $X \in \mathcal{X}$ *and* $a \in N$, *there is an edge*



In the case of a constant equation $X\,s \stackrel{?}{=} \flat$, *we use* ε *as the label of the edge.*

The size *of a graph of surface dependencies* D, *denoted as* $|D|$, *is defined as its number of edges.*

Note that the node $\emptyset$ has no outgoing edges.

[2]There may be several edges, even labeled differently between two nodes.

6.1. Small components of solutions. We say that a solution of $\langle E, G\rangle$ has a small component if there exists a variable whose value is "small" enough to be described just as the prefix of some word defined by G. This will be helpful in the construction of a compact representation of a size-minimal solution for several reasons: It is a case where the instantiation of a variable is completely known, it can be constructed with only a small increase of the grammar, and it eases the definition and argumentation for the remaining cases.

We identify some classes of compacted equations that have a solution with a small component.

DEFINITION 6.5. *Given the simplified and compacted set of equations $\langle E, G\rangle$, a variable X occurring in E, and a solution σ, we say that X is a* small component *of σ if $\sigma(X) = \lambda x\,.\,v\,x$, or $\sigma(X) = \lambda x\,.\,v\,\flat$ and either $v = \varepsilon$ or there is a nonterminal a in G such that v is a prefix of w_a.*

LEMMA 6.6 (small component). *Let $\langle E, G\rangle$ be a simplified and compacted set of equations, and let σ be a solution of $\langle E, G\rangle$ with a small component X such that $\sigma(X) = \lambda x\,.\,v\,x$ or $\sigma(X) = \lambda x\,.\,v\,\flat$. Then, there exists a singleton CFG $G' \supseteq G$ generating v and satisfying*

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

Proof. The inequalities follow from Definition 6.5 and Lemma 4.6. ∎

This lemma is helpful in two ways: It allows us to eliminate variables from the problem and it restricts the cases where this elimination does not work (and it is necessary to rewrite the equations) to cases in which solutions do not have small components; i.e., they have only "large" instantiations. This will simplify the reasoning.

LEMMA 6.7. *All solutions of simplified and compacted sets of equations containing constant equations have at least one small component.*

Proof. Let $\langle E, G\rangle$ be a simplified and compacted set of equations, and let σ be a solution. Since σ solves a constant equation of the form $X\,s \stackrel{?}{=} a\,\flat$, it must instantiate X either with $\lambda x\,.\,v\,x$ or with $\lambda x\,.\,v\,\flat$ for some prefix v of w_a. Similar arguments hold for equations of the form $X\,s \stackrel{?}{=} \flat$. ∎

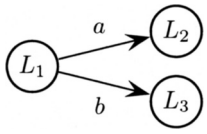
The following lemma describes the situation that reflects the difference between basic MSOU problems and WU when all equations are *flexible-flexible*. While in MSOU we can instantiate variables by terms not using their arguments, such as $\lambda x\,.\,\flat$, this is not possible when considering WU; hence the lemma does not hold for WU. This is the point that shows that our result is not straightforwardly transferable to WU.

LEMMA 6.8. *Let $\langle E, G\rangle$ be a simplified and compacted set of equations such that the graph of surface dependencies does not contain edges and such that $FV(E) \neq \emptyset$, and let σ be a size-minimal solution of $\langle E, G\rangle$. Then, for every variable $X \in FV(E)$, either $\sigma(X) = \lambda x\,.\,\flat$ or $\sigma(X) = \lambda x\,.\,x$. This also means that there is at least one small component in σ.*

Proof. If there are not edges, then all equations are of the form $X\,s \stackrel{?}{=} Y\,t$. The substitution σ, with $\sigma(X) = \lambda x\,.\,\flat$ for every variable $X \in FV(E)$, is a size-minimal solution; hence for every other size-minimal solution σ' and for every variable $X \in FV(E)$, only $\sigma'(X) = \lambda x\,.\,\flat$ or $\sigma'(X) = \lambda x\,.\,x$ is possible.[3] ∎

[3]We could make the solution with $\sigma'(X) = \lambda x\,.\,\flat$, for all variables, be the only size-minimal solution by changing the term size measure to make $\lambda x\,.\,\flat$ smaller than $\lambda x\,.\,x$. However, then the exponent of periodicity bound (see Lemma 2.4) must be adapted.

DEFINITION 6.9. *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 following form (where L_1, L_2, and L_3 are not necessarily distinct nodes):*



If neither w_a is a prefix of w_b nor w_b is a prefix of w_a, then it is called a strong divergence *and otherwise a* weak divergence.

Strong divergences are easy to eliminate, whereas weak divergences require a more complex treatment, which will be done by rewriting the graph.

LEMMA 6.10. *Given a simplified and compacted set of equations, if its graph of surface dependencies contains a strong divergence, then every solution has small components.*

Proof. Given $\langle E, G\rangle$, let D be its graph of surface dependencies, and let σ be any of its solutions. Assume that D contains a strong divergence $L_2 \overset{a}{\leftarrow} L_1 \overset{b}{\rightarrow} L_3$. For every variable $X \in L_1$, let $v_X \in \Sigma^*$ be a word such that either $\sigma(X) = \lambda y\,.\,v_X\,y$ or $\sigma(X) = \lambda y\,.\,v_X\,\natural$. By definition of D, we have a pair of equations in E of the form $X_1 \cdots \overset{?}{=} a\,t_1$ and $X_2 \cdots \overset{?}{=} b\,t_2$, where $X_1, X_2 \in L_1$. Therefore, v_{X_1} is a prefix of $\sigma(w_a\,t_1)$, and v_{X_2} is a prefix of $\sigma(w_b\,t_2)$. Now, let $Y \in L_1$ be the variable such that v_Y is the shortest word of $\{v_X \mid X \in L_1\}$. By the equivalence relation and the dependence graph definitions, we have that v_Y is a prefix of both v_{X_1} and v_{X_2}, and thus a prefix of $\sigma(w_a\,t_1)$ and of $\sigma(w_b\,t_2)$. Since w_a is not a prefix of w_b, and vice versa, we have that v_Y is a proper prefix of both w_a and w_b. (Notice that v_Y is not necessarily the longest common prefix of w_a and w_b). Therefore, Y is a small component of σ. ∎

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

LEMMA 6.11. *Let E be a (noncompacted) set of equations with a cycle of the form*

$$\begin{array}{lcl} X_1 \cdots & \overset{?}{=} & w_1\,X_2 \cdots \\ X_2 \cdots & \overset{?}{=} & w_2\,X_3 \cdots \\ & \cdots & \\ X_m \cdots & \overset{?}{=} & w_m\,X_1 \cdots, \end{array}$$

where $w_i \in \Sigma^*$, for every $i = 1, \ldots, m$, and $w_1 \ldots w_m \neq \varepsilon$.

Then, for every solution σ of E, there is some variable X_k, with $1 \leq k \leq m$, such that $\sigma(X_k) = \lambda x \,.\, u\, x$, where u is a prefix of $(w_k \dots w_m\, w_1 \dots w_{k-1})^n$ for sufficiently large n.

Proof. The proof is by induction on the size of σ.

For any $i = 1, \dots, m$, if σ solves $X_i \cdots \stackrel{?}{=} w_i\, X_{i+1} \cdots$, then either $\sigma(X_i) = \lambda x \,.\, u_i\, x$ for some proper prefix u_i of w_i, or we have $\sigma(X_i) = \lambda x \,.\, w_i\, v_i\, x$ or $\sigma(X_i) = \lambda x \,.\, w_i\, v_i \flat$, for some word v_i. Therefore, there are two cases:

If, for some $k = 1, \dots, m$, we have the first situation that there is some X_k such that $\sigma(X_k) = \lambda x \,.\, u_k\, x$, where u_k is a prefix of w_k, then the claim of the lemma holds.

Otherwise, for every $i = 1, \dots, m$, we have $\sigma(X_i) = \lambda x \,.\, w_i\, v_i\, x$ or $\sigma(X_i) = \lambda x . w_i\, v_i \flat$. In this case we generate a new system by instantiating X_i by $\lambda x \,.\, w_i\, X'_i\, x$, where X'_i are fresh and different unary second-order variables. Then, the equations in the cycle become

$$\begin{array}{lcl} w_1\, X'_1 \cdots & \stackrel{?}{=} & w_1\, w_2\, X'_2 \cdots, \\ w_2\, X'_2 \cdots & \stackrel{?}{=} & w_2\, w_3\, X'_3 \cdots, \\ & \cdots & \\ w_m\, X'_m \cdots & \stackrel{?}{=} & w_m\, w_1\, X'_1 \cdots. \end{array}$$

Simplifying the equations, we obtain a new system of equations,

$$\begin{array}{lcl} X'_1 \cdots & \stackrel{?}{=} & w_2\, X'_2 \cdots, \\ X'_2 \cdots & \stackrel{?}{=} & w_3\, X'_3 \cdots, \\ & \cdots & \\ X'_m \cdots & \stackrel{?}{=} & w_1\, X'_1 \cdots. \end{array}$$

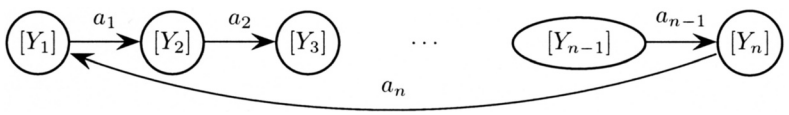
From the original solution σ we get a solution σ' of the new equations satisfying $\sigma'(X'_i) = \lambda x \,.\, v_i\, x$ or $\sigma(X'_i) = \lambda x \,.\, v_i \flat$, for all $i = 1, \dots, m$. Now the induction hypothesis applies since σ' is smaller than σ; hence there is a variable X'_k such that $\sigma'(X'_k) = \lambda x \,.\, v_k\, x$, where v_k is a prefix of $(w_{k+1} \dots w_m\, w_1 \dots w_k)^n$, for large enough n. Hence $\sigma(X_k) = \lambda x . w_k\, v_k\, x$, and the claim holds. ∎

LEMMA 6.12 (cycles). *Let $\langle E, G \rangle$ be a solvable, simplified, and compacted set of equations, with a graph of surface dependencies D with some cycle. Then, for every solution σ without small components, there exists a variable X such that $\sigma(X) = \lambda y \,.\, w\, y$ and w is generated by some grammar $G' \supseteq G$ satisfying*

$$\begin{array}{lcl} |G'| & = & |G| + \mathcal{O}(\mathit{depth}(G) + \#Eq(E) + \log \mathrm{eop}(\sigma)), \\ \mathit{depth}(G') & = & \mathit{depth}(G) + \mathcal{O}(\log \#Eq(E) + \log \mathrm{eop}(\sigma)). \end{array}$$

More precisely, the corresponding node $[X]$ is inside the cycle, and, for some $0 \leq n \leq \mathrm{eop}(\sigma)$ and some prefix v of w_α, we have $\sigma(X) = \lambda y \,.\, (w_\alpha)^n\, v\, y$, where $\alpha \in N^*$ is the sequence of labels of the edges completing the cycle from $[X]$.

Proof. Select a cycle in the graph D:



Therefore, there is a subset of equations in E of the form

$$\begin{array}{llll} Y_{1,m_1} \ldots \stackrel{?}{=} a_1\, Y_{2,1} \ldots; & Y_{2,1} \ldots \stackrel{?}{=} Y_{2,2} \ldots; & \cdots & Y_{2,m_2-1} \ldots \stackrel{?}{=} Y_{2,m_2} \ldots; \\ Y_{2,m_2} \ldots \stackrel{?}{=} a_2\, Y_{3,1} \ldots; & Y_{3,1} \ldots \stackrel{?}{=} Y_{3,2} \ldots; & \cdots & Y_{3,m_3-1} \ldots \stackrel{?}{=} Y_{3,m_3} \ldots; \\ \ldots & & & \\ Y_{n,m_n} \ldots \stackrel{?}{=} a_n\, Y_{1,1} \ldots; & Y_{1,1} \ldots \stackrel{?}{=} Y_{1,2} \ldots; & \cdots & Y_{1,m_1-1} \ldots \stackrel{?}{=} Y_{1,m_1} \ldots, \end{array}$$

where $\{Y_{i,1}, \ldots, Y_{i,m_i}\} \subseteq [Y_i]$ for $i = 1, \ldots, n$. Note that $w_{a_i} \neq \varepsilon$ for all i, since E is simplified.

Now, fix a solution σ and proceed as follows. Let E' be the set of equations represented by the compacted equations above. Notice that $w_{a_1} \cdots w_{a_n} \neq \varepsilon$, and E' fulfills the conditions of Lemma 6.11. The substitution σ also solves E', and applying Lemma 6.11, we get a variable $Y_{k,l}$ such that $\sigma(Y_{k,l}) = \lambda x.w_\alpha^n\, v\, x$, where $\alpha = a_k \cdots a_m\, a_1 \cdots a_{k-1}$ and v is a prefix of w_α. Moreover, we have $n \leq \mathrm{eop}(\sigma)$.

To prove the existence of G', we proceed by adding new rules to G. Note that all symbols labeling the edges of D are nonterminals in G. We construct a sequence of grammars $G \subseteq G_1 \subseteq G_2 \subseteq G_3 \subseteq G'$ such that G_1, apart from the words generated by G, also generates w_α, G_2 also generates v, G_3 also generates $(w_\alpha)^n$, and G' also generates $(w_\alpha)^n\, v$.

Since the length of α is at most $|D|$, by Lemma 4.4, we have

$$\begin{array}{lcl} |G_1| & \leq & |G| + |D| - 1, \\ \mathit{depth}(G_1) & \leq & \mathit{depth}(G) + \lceil \log |D| \rceil. \end{array}$$

By Lemma 4.6, we can define v with

$$\begin{array}{lcl} |G_2| & \leq & |G_1| + \mathit{depth}(G_1), \\ \mathit{depth}(G_2) & = & \mathit{depth}(G_1). \end{array}$$

By Lemma 4.5, we can define $(w_\alpha)^n$ with

$$\begin{array}{lcl} |G_3| & \leq & |G_2| + 2\, \lfloor \log \mathrm{eop}(\sigma) \rfloor, \\ \mathit{depth}(G_3) & \leq & \mathit{depth}(G_2) + \lceil \log \mathrm{eop}(\sigma) \rceil \end{array}$$

and, since we still need another rule to define $(w_\alpha)^n\, v$,

$$\begin{array}{lcl} |G'| & \leq & |G_3| + 1, \\ \mathit{depth}(G') & \leq & \mathit{depth}(G_3) + 1. \end{array}$$

The composition of all these inequalities results in the inequalities

$$\begin{array}{lcl} |G'| & \leq & |G| + \mathit{depth}(G) + |D| + \lceil \log |D| \rceil + 2\, \lfloor \log \mathrm{eop}(\sigma) \rfloor, \\ \mathit{depth}(G') & \leq & \mathit{depth}(G) + \lceil \log |D| \rceil + \lceil \log \mathrm{eop}(\sigma) \rceil + 1. \end{array}$$

In terms of O-notation, this reduces to

$$\begin{array}{lcl} |G'| & = & |G| + \mathcal{O}(\mathit{depth}(G) + |D| + \log \mathrm{eop}(\sigma)), \\ \mathit{depth}(G') & = & \mathit{depth}(G) + \mathcal{O}(\log |D| + \log \mathrm{eop}(\sigma)). \end{array}$$

And, since $|D| \leq \#Eq(E)$,

$$\begin{array}{lcl} |G'| & = & |G| + \mathcal{O}(\mathit{depth}(G) + \#Eq(E) + \log \mathrm{eop}(\sigma)), \\ \mathit{depth}(G') & = & \mathit{depth}(G) + \mathcal{O}(\log \#Eq(E) + \log \mathrm{eop}(\sigma)), \end{array}$$

as stated in the lemma. □

7. Rewriting the graph of dependencies. In the previous section we saw that Lemmas 6.6 and 6.12 both describe the instantiation $\sigma(X)$ of some variable and allow us to eliminate it during the construction of a compact representation of the solution σ. In other words, they describe parts of the solution, i.e., a substitution $\rho = [X \mapsto \sigma(X)]$ satisfying $\rho \preceq_{FV(E)} \sigma$.

In this section we will see that, when these two lemmas are not applicable, we can *rewrite* the set of equations (and its corresponding graph of dependencies) until one of the two lemmas becomes applicable. This rewriting process is done by *partially* instantiating some variables, i.e., applying a substitution ρ also satisfying $\rho \preceq_{FV(E)} \sigma$. The substitution has the form $\rho = [X \mapsto \lambda y\,.\,w\,X'\,y]$, where X' is a fresh variable and $w \in \Sigma^*$. Substitutions of such form, as well as the total instantiations of the form $\rho = [X \mapsto \lambda y\,.\,w\,y]$ and $\rho = [X \mapsto \lambda y\,.\,w\,\flat]$ described in Lemmas 6.6 and 6.12, are all called *partial instantiations*. Formally, we define partial instantiations as follows.

DEFINITION 7.1. *We say that a substitution ρ is a* partial instantiation *if it can be decomposed as $\rho = (\rho_1 \circ \cdots \circ \rho_n)|_{Dom(\rho)}$, where each ρ_i either has the form $\rho_i = [X_i \mapsto \lambda y\,.\,w_i\,X_i'\,y]$, $\rho_i = [X_i \mapsto \lambda y\,.\,y]$, or $\rho_i = [X_i \mapsto \lambda y\,.\,\flat]$, for some $X_i, X_i' \in \mathcal{X}_1$ and some $w_i \in \Sigma^*$.*

The following lemma states the preservation of some properties of partial instantiations of equations. Notice that some of these properties do not hold for arbitrary substitutions satisfying $\rho \preceq_{FV(E)} \sigma$.

LEMMA 7.2 (preservation). *For any (noncompacted) set of equations E, any solution σ, and any partial instantiation ρ, satisfying $\rho \preceq_{FV(E)} \sigma$, there exists a substitution σ' satisfying*

(i) $\sigma = (\sigma' \circ \rho)|_{FV(E)}$,
(ii) *σ' is a solution of $\rho(E)$,*
(iii) *if σ is a size-minimal solution of E, then σ' is also a size-minimal solution of $\rho(E)$,*
(iv) $\mathrm{eop}(\sigma) \geq \mathrm{eop}(\sigma')$,
(v) $|\,FV(E)| \geq |\,FV(\rho(E))|$, *and*
(vi) *the number of occurrences of variables in E is greater than or equal to the number of occurrences of variables in $\rho(E)$.*

Proof. The requirement $\rho \preceq_{FV(E)} \sigma$ ensures that there exists a substitution σ' such that $\sigma(X) = \sigma' \circ \rho(X)$ for any $X \in FV(E)$. The restriction of σ' to the domain $FV(\rho(E))$ also satisfies this property. Therefore, the required σ' always exists.

Since σ is a solution of E, for any variable $X \in FV(E)$, $\sigma(X)$ is a closed term. Moreover, since $\sigma = (\sigma' \circ \rho)|_{FV(E)}$, for any variable $X \in FV(E)$, $\sigma(X) = \sigma' \circ \rho(X)$. The same applies to any term containing only variables of E, hence to any side of any equation of E. Therefore, for any equation $\rho(s) \stackrel{?}{=} \rho(t)$ of $\rho(E)$, we have $\sigma'(\rho(s)) = \sigma(s) = \sigma(t) = \sigma'(\rho(t))$. Hence, σ' solves $\rho(E)$.

Minimality is proved by contradiction. Assume that σ' is not size-minimal. Let τ' be a size-minimal solution of $\rho(E)$. Obviously, $\tau' \circ \rho$ is a solution of E. Since τ' is size-smaller than σ', we have $\sum_{X \in FV(\rho(E))} |\tau'(X)| < \sum_{X \in FV(\rho(E))} |\sigma'(X)|$. Now, since ρ is a partial instantiation, we have

$$\sum_{X \in FV(E)} |\tau'(\rho(X))| = \sum_{X \in FV(E)} |\rho(X)| + \sum_{Y \in FV(\rho(E))} (|\tau'(Y)| - 1) \quad \text{and}$$
$$\sum_{X \in FV(E)} |\sigma'(\rho(X))| = \sum_{X \in FV(E)} |\rho(X)| + \sum_{Y \in FV(\rho(E))} (|\sigma'(Y)| - 1).$$

Therefore, $\sum_{X\in FV(E)} |\tau'(\rho(X))| < \sum_{X\in FV(E)} |\sigma'(\rho(X))| = \sum_{X\in FV(E)} \sigma(X)$, which contradicts the assumption that σ is size-minimal.

Let $X' \in Dom(\sigma')$ be a variable, and let v be a nonempty word, such that $\sigma'(X') = \lambda y\,.\,u\,v^{\mathrm{eop}(\sigma')}\,w\,\flat$ (or similarly replacing $\flat$ by y). Since $Dom(\sigma') = FV(\rho(E))$, either $X' \in FV(E)$ or, for some variable $Y \in FV(E)$, $\rho(Y)$ contains X. Hence, in both cases, there exists a variable $X \in FV(E) \subseteq Dom(\sigma)$ such that $\sigma(X)$ contains $v^{\mathrm{eop}(\sigma')}$. Therefore $\mathrm{eop}(\sigma)$ is at least $\mathrm{eop}(\sigma')$.

The requirement that ρ is a partial instantiation ensures that after applying this substitution the number of variables and the number of occurrences of variables in E do not increase. □

Now we deal with the following case: There are compacted sets of equations whose graph of surface dependencies does not have any cycle, and the focused solution does not have any small component; i.e., we deal with the case not covered by Lemmas 6.6 and 6.12. Since there are no cycles, the edges define a partial order $\succ$ on the nodes, with $N \succ N'$ if and only if $N \to N'$ is an edge. Hence we can speak of $\succ$-maximal nodes. Since there are size-minimal solutions without small components, Lemma 6.8 shows that these graphs contain at least one edge, Lemma 6.10 states that they do not contain strong divergences, and Lemma 6.7 shows that they do not contain any edge to the node labeled with $\emptyset$. There is a $\succ$-maximal node with at least one outgoing edge to other nodes and without any strong divergence. We will transform the equations, whose graph of dependencies contains such maximal nodes, in order to obtain a description of some variable instantiation. In fact, this transformation on the equations carries over to a graph transformation. An example of this graph transformation (or rewriting) is shown in Example 7.

DEFINITION 7.3. *Let $\langle E, G\rangle$ with $FV(E) \neq \emptyset$ be a simplified and compacted set of equations without cycles such that there is a solution without small components. Then the transformation rule $\langle E, G\rangle \Rightarrow \langle E', G'\rangle$ is defined as follows.*

Let D be the graph of surface dependencies of $\langle E, G\rangle$. Let $[X]$ be a $\succ$-maximal node in D with at least one outgoing edge. Let $\{a_1, \ldots, a_m\}$ be the set of all labels of the outgoing edges of $[X]$, where w_{a_1} is a prefix of w_{a_i}, for all $i = 1, \ldots, m$. For every $Y \in [X]$, let Y' be a fresh unary variable, and let ρ be the substitution that maps each $Y \in [X]$ to $\lambda y\,.\,w_{a_1}\,Y'\,y$. Then let $\langle E', G'\rangle$ be the simplification of $\langle \rho(E), G\rangle$.

If D' is the graph of surface dependencies of $\langle E', G'\rangle$, we write $D \Rightarrow D'$.

Notice that in the previous definition, since there are not any cycles and there are solutions without small components, by Lemma 6.8 there are edges, and hence there is a $\succ$-maximal node with some outgoing edge; by Lemmas 6.7 and 6.10 there are neither constant equations nor strong divergences. This allows us to assume that w_{a_1} is a prefix of w_{a_i} for $i = 1, \ldots, m$. Notice also that for all equations of the form $Y\,s \stackrel{?}{=} a\,Z\,t$, where $Y \in [X]$, the symbol a is the label of some outgoing edge of $[X]$. Finally, notice that the substitution ρ is a partial instantiation; hence Lemma 7.2 applies.

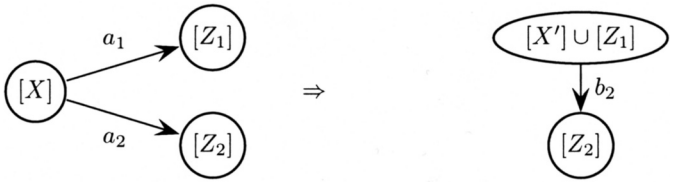
The transformation of $\langle E, G\rangle$ results in the compacted equations $\langle E', G'\rangle$ satisfying the following:

(i) The grammar G' is an extension of G such that for $i = 1, \ldots, m$, the nonterminal b_i generates the word v_i, which is defined by $w_{a_i} = w_{a_1}\,v_i$.

(ii) The set of equations E' is obtained from E by replacing all equations of the form $Y\,s \stackrel{?}{=} Z\,t$, where $Y, Z \in [X]$, by the equation $Y'\,\rho(s) \stackrel{?}{=} Z'\,\rho(t)$, and replacing every equation of the form $Y\,s \stackrel{?}{=} a_i\,Z\,t$, where $Y \in [X]$, by $Y'\,\rho(s) \stackrel{?}{=} b_i\,Z\,\rho(t)$.

Every solution σ without small components of E can be transformed into a solution of E' by defining it on the fresh variables Y' (see Lemma 7.5).

In the special case that $m = 2$, $w_{a_1} \neq w_{a_2}$, and $w_{a_1} \prec w_{a_2}$, the transformation on the graph of surface dependencies can be represented as a graph rewriting rule



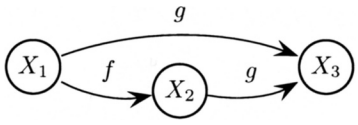
where b_2 is a new nonterminal of the grammar G', that generates a word satisfying $w_{a_2} = w_{a_1} w_{b_2}$.

Notice that in this rewriting process at least one arrow and also the node $[X]$ are removed. Note also that in the case $w_{a_1} = w_{a_2}$, the three nodes are merged into one node $[X'] \cup [Z_1] \cup [Z_2]$, thus removing more than one edge.

Example 7. Consider the following simplified and compacted set of equations and their set of solution components for $n \geq 0$:

$$\begin{array}{l} \Sigma = \{a, b, c, d, e\}, \\ N = \{f, g\}, \\ X_1\, c\flat \stackrel{?}{=} f\, X_2\, c\flat, \\ X_1\, d\flat \stackrel{?}{=} g\, X_3\, a\, b\, d\flat, \\ X_2\, e\flat \stackrel{?}{=} g\, X_3\, e\flat, \\ G\text{-rules: } \{f \to a\, b,\ g \to a\}. \end{array} \qquad \begin{array}{lcl} X_1 & \mapsto & \lambda x\,.\,(a\, b)^{n+2}\, x, \\ X_2 & \mapsto & \lambda x\,.\,(a\, b)^{n+1}\, x, \\ X_3 & \mapsto & \lambda x\,.\,b\,(a\, b)^{n}\, x, \end{array}$$

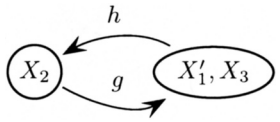
The graph of surface dependencies is



Applying the transformation rule to the only $\succ$-maximal node $[X_1]$, we get the following simplified and compacted set of equations E':

$$\begin{array}{l} \Sigma = \{a, b, c, d, e\}, \\ N' = \{f, g, h\}, \\ X'_1\, c\flat \stackrel{?}{=} h\, X_2\, c\flat, \\ X'_1\, d\flat \stackrel{?}{=} X_3\, a\, b\, d\flat, \\ X_2\, e\flat \stackrel{?}{=} g\, X_3\, e\flat, \\ G'\text{-rules: } \{f \to a\, b,\ g \to a,\ h \to b\}. \end{array}$$

The modified graph is as follows:



LEMMA 7.4 (rewriting). *Let $\langle E, G\rangle$ be a simplified and compacted set of equations, and let $\langle E, G\rangle \Rightarrow \langle E', G'\rangle$; then*

$$\begin{array}{lcl} |G'| & \leq & |G| + \#Eq(E)\ depth(G), \\ depth(G') & = & depth(G). \end{array}$$

Moreover, $|E'| \leq |E| + m$, where m is the number of variable occurrences of E.

Proof. Let m be the number of outgoing edges of the removed node. The new grammar G' extends G by defining $m-1$ suffixes of words defined by G. Therefore, according to Lemma 4.6, we can obtain such a grammar G' satisfying the upper bounds:

$$\begin{array}{lcl} |G'| & \leq & |G| + (m-1)\, depth(G), \\ depth(G') & = & depth(G). \end{array}$$

It is easy to check that $m \leq \#Eq(E)$.

Finally E' is obtained by replacing in E the variable occurrences of some variables Y (belonging to $[X]$) by $b\,Y'$ and then simplifying. This simplification only has to remove the same nonterminal symbol from the head of both sides of some equations; hence it does not increase the size of the grammar. Therefore, the rewriting increases the size of E at most by 1 for each variable occurrence in E. ∎

LEMMA 7.5. *For any simplified and compacted set of equations $\langle E, G\rangle$ without cycles in its graph of surface dependencies, any solution σ without small components, and any transformation $\langle E, G\rangle \Rightarrow \langle E', G'\rangle$ defined by the substitution ρ, there exists a substitution σ' such that*

(i) *σ' is a solution of $\langle E', G'\rangle$,*

(ii) *σ' satisfies $\sigma = (\sigma' \circ \rho)|_{FV(E)}$.*

Proof. If there are not any small components, then the substitution ρ satisfies $\rho \preceq_{FV(E)} \sigma$. Then the lemma is a direct consequence of Lemma 7.2. ∎

The previous lemma may be iterated: If σ' does not contain small components, and the graph of surface dependencies of $\langle E', G'\rangle$ does not contain cycles, we use it again to obtain a new solution σ'' of a new $\langle E'', G''\rangle$ and so on. By Lemma 7.6, this process cannot be repeated more than $\#Eq(E)$ times.

LEMMA 7.6. *Any graph rewriting sequence $D \Rightarrow^* D'$ has length at most $|D|$.*

Proof. This is clear, since in every transformation step at least one edge is removed. ∎

In Figure 7.1 we define an algorithm that, given the compacted set of equations $\langle E, G\rangle$ and a size-minimal solution σ, computes a polynomial-sized compacted solution $\langle \rho, G'\rangle$ representing σ. Note that the complexity of this algorithm is irrelevant; only the polynomial size of the obtained representation will be needed.

THEOREM 7.7 (compacted solution). *Given the initial compacted set of equations $\langle E_0, G_0\rangle$ and the size-minimal solution σ_0, the algorithm of Figure* 7.1 *computes a compacted solution $\langle \rho, G'\rangle$ representing σ_0.*

Moreover, the following inequalities hold:

$$\begin{array}{lcl} |G'| & \leq & |G_0| + \mathcal{O}(|E_0|^4\ depth(G_0) + |E_0|^6), \\ depth(G') & \leq & depth(G_0) + \mathcal{O}(|E_0|^2), \\ |\rho| & = & \mathcal{O}(|E_0|^3). \end{array}$$

Proof. The algorithm performs a sequence of transformations on the compacted equations $\langle E, G\rangle$, the compacted substitution $\langle \rho, G\rangle$, and the solution σ. These transformations are of four types: simplification (step 6), small components (step 9), cycles (step 16), and rewriting (step 23). First we show how many transformations of each type are performed and then how they modify some of the measures of the representations (number of equations, their size, etc.).

(i) *Termination*: Cycle and small component transformations remove a variable from E; therefore they cannot be executed more than $|E_0|$ times. According

Input: σ_0, $\langle E_0, G_0\rangle$
Output: ρ, G'

1. $\rho := Id$
2. $\langle E, G\rangle := \langle E_0, G_0\rangle$
3. $\sigma := \sigma_0$
4. while $FV(E) \neq \emptyset$ do
5. if $\langle E, G\rangle$ is not simplified then
6. let $\langle E', G'\rangle$ be the simplification of $\langle E, G\rangle$
7. $\langle E, G\rangle := \langle E', G'\rangle$
8. elseif σ has a small component X then
9. let G' be the grammar described in Lemma 6.6, and
10. let a be the nonterminal of G' generating v
11. if $\sigma(X) = \lambda x\,.\,v\,x$ then $\rho := [X \mapsto \lambda x\,.\,a\,x] \circ \rho$
12. if $\sigma(X) = \lambda x\,.\,v\,\flat$ then $\rho := [X \mapsto \lambda x\,.\,a\,\flat] \circ \rho$
13. $\langle E, G\rangle := \langle \rho(E), G'\rangle$
14. $\sigma = \sigma|_{FV(E)}$
15. elseif D contains a cycle then
16. let G' be the grammar,
17. let X be the variable in the cycle described in Lemma 6.12, and
18. let a be the nonterminal generating $(w_\alpha)^n\,v$, where $\sigma(X) = \lambda y\,.\,(w_\alpha)^n\,v\,y$
19. $\rho := [X \mapsto \lambda x\,.\,a\,x] \circ \rho$
20. $\langle E, G\rangle := \langle \rho(E), G'\rangle$
21. $\sigma = \sigma|_{FV(E)}$
22. else
23. compute $\langle E, G\rangle \Rightarrow \langle E', G'\rangle$
24. let $[X]$ be the $\succ$-maximal class transformed by this rewriting, and
25. let a_1 be the label of the outgoing edge of $[X]$ generating the shortest word
26. $\tau := Id$
27. for all $Y \in [X]$
28. $\tau := [Y \mapsto \lambda y\,.\,a_1\,Y'\,y] \circ \tau$
29. let σ' be the solution of $\langle E', G'\rangle$ satisfying $\sigma = (\sigma' \circ \tau)|_{FV(E)}$ given by Lemma 7.5
30. $\rho = \tau \circ \rho$
31. $\langle E, G\rangle := \langle E', G'\rangle$
32. $\sigma := \sigma'$
33. endwhile
34. $\rho := \rho|_{FV(E_0)}$

FIG. 7.1. *Pseudocode of the algorithm to compute a representation ρ of a solution.*

to Lemma 7.6, rewriting sequences cannot be longer than $|D|$. The size $|D|$ of the graph of surface dependencies is bounded by the number of equations (we will see in the following that this measure is decreasing), hence by $|E_0|$. After every rewriting sequence we get a set of equations E with a cycle, or a solution σ with a small component. Therefore, there is a total number of at most $|E_0|^2$ rewriting steps. Finally, after every rewriting step we get a simplified set of equations. Therefore, we cannot perform more simplification steps than cycle elimination plus small component elimination steps, hence not more than $|E_0|$.

(ii) *Number of equations, variables, and occurrences of variables*: Simplifications preserve or decrease all these parameters, as well as the partial instantiations performed by the cycle and small component transformations, according to Lemma 7.2. Rewritings are composed by a partial instantiation followed by a simplification; therefore they also preserve or decrease these parameters.

(iii) *Size of the equations,* $|E|$: The size of E is preserved or decreases with simplifications, cycles, and small component steps. However, it can be increased in the number of occurrences of variables at every rewriting step (see Lemma 7.4). Since there are no more than $|E_0|^2$ rewriting steps and the increase is bounded by $|E_0|$, we have $|E| \leq |E_0| + |E_0|^3$ along the execution of the algorithm.

(iv) *Exponent of periodicity,* $\mathrm{eop}(\sigma)$: According to Lemma 7.2, the exponent of periodicity of the solution, after a partial instantiation like the ones we perform in the cycle, small component and rewriting steps, is preserved or decreases. Since by Lemma 2.4, $\mathrm{eop}(\sigma_0) \leq 2^{\alpha\,|E_0|}$ for the initial minimal solution σ_0, this bound holds along the execution of the algorithm.

(v) *Depth of the grammar,* $depth(G)$: There are the following possibilities:

$$\begin{array}{lcll}
depth(G') & = & depth(G) & + \mathcal{O}(\log |E|) \quad \text{(simplification, Lemma 6.2),} \\
depth(G') & = & depth(G) & \quad \text{(small component, Lemma 6.6),} \\
depth(G') & = & depth(G) & + \mathcal{O}(\log \#Eq(E) \\
 & & & + \log \mathrm{eop}(\sigma)) \quad \text{(cycle, Lemma 6.12),} \\
depth(G') & = & depth(G) & \quad \text{(rewriting, Lemma 7.4).}
\end{array}$$

Since $\log \mathrm{eop}(\sigma) = \mathcal{O}(|E_0|)$ and the number of cycle steps as well as of simplifications is at most $|E_0|$, an upper bound is $depth(G) = depth(G_0) + \mathcal{O}(|E_0|^2)$.

(vi) *Size of the grammar,* $|G|$: There are the following possibilities:

$$\begin{array}{lcll}
|G'| & = & |G| & + \mathcal{O}(|E|(depth(G) + \log |E|)) \quad \text{(simplification, Lemma 6.2),} \\
|G'| & \leq & |G| & + depth(G) \quad \text{(small component, Lemma 6.6),} \\
|G'| & = & |G| & + \mathcal{O}(depth(G) + \#Eq(E) \\
 & & & + \log \mathrm{eop}(\sigma)) \quad \text{(cycles, Lemma 6.12),} \\
|G'| & \leq & |G| & + \#Eq(E)\ depth(G) \quad \text{(rewriting, Lemma 7.4).}
\end{array}$$

We know that the maximal number of rewriting steps is $|E_0|^2$ and the maximal number of small component, simplification, and cycle steps is at most $|E_0|$. We also have $\log \mathrm{eop}(\sigma) = \mathcal{O}(|E_0|)$, $|E| = \mathcal{O}(|E_0|^3)$, and $\#Eq(E) = \mathcal{O}(|E_0|)$. Together with the upper bound on $depth(G)$, this gives an upper bound (simplification is responsible for the dominating terms):

$$\begin{array}{rcl}
|G'| & = & |G_0| + |E_0|\,\mathcal{O}(|E_0|^3(depth(G) + \log |E_0|^3)), \\
 & & \quad + |E_0|\,depth(G), \\
 & & \quad + |E_0|\,\mathcal{O}(depth(G) + |E_0|), \\
 & & \quad + 6|E_0|^2\,|E_0|\ depth(G), \\
 & = & |G_0| + \mathcal{O}(|E_0|^4\,depth(G_0) + |E_0|^6).
\end{array}$$

(vii) *Size of the compacted solution,* $|\rho|$: We have to represent the instantiation of at most $|E_0|$ variables, where the size of each instantiation is bounded by the number of rewriting steps. This gives $\mathcal{O}(|E_0|^3)$.

All the transformations are sound according to Lemma 7.2. Now, if the compacted equations are not simplified, we can always simplify them. If they are simplified, either there is a cycle, or a solution with small components, or we can rewrite the equations. ∎

8. Main results and some remarks. Theorem 7.7 states that, given a compact representation $\langle E, G\rangle$ of a set of equations, we can build a new singleton grammar G' of polynomial size defining all components of the compact representation of a size-minimal solution. The final step is to use Plandowski's theorem (Theorem 4.2) to check that the polynomial-sized guessed substitution is really a unifier.

MAIN THEOREM 8.1. *Solvability of compact representations of basic MSOU problems is NP-complete.*

Proof. Theorem 7.7 shows that for every compact representation $\langle E, G\rangle$ of a basic MSOU problem and every size-minimal solution σ, there is compacted solution $\langle \rho, G'\rangle$ that represents σ, where G' is a singleton grammar of polynomial size in $|E| + |G|$ and $|\rho|$ is also polynomial in $|E|$.

Thus we can guess a polynomial-sized singleton grammar G' and a compacted solution ρ as above, and then test whether $\langle \rho, G'\rangle$ is a solution of $\langle E, G\rangle$. We can replace every variable occurrence in E by its instantiation in ρ, then normalize both sides of each equation $s_i \stackrel{?}{=} t_i$, to obtain $s'_i \stackrel{?}{=} t'_i$, and, finally, extend G' to obtain G'' by Lemma 4.4, generating $s'_1 \# \cdots \# s'_n$ and $t'_1 \# \cdots \# t'_n$, where $\#$ is a new constant symbol. Then, the test for solvability is an equality test w.r.t. the singleton grammar G'', which can be performed in polynomial time by Plandowski's theorem (see Theorem 4.2). This shows that the problem is in NP. Together with the NP-hardness of the problem, which was proved in [22], this leads us to conclude that the problem is NP-complete. ∎

COROLLARY 8.2. *Monadic second-order unification is NP-complete.*

Proof. The proof follows from Theorem 8.1 and Proposition 3.1. ∎

COROLLARY 8.3. *Monadic second-order matching is NP-complete.*

Proof. The proof follows from Theorems 2.6 and 8.1. ∎

Remark 1. Theorem 7.7 clarifies the increase of the size of the grammar representing a size-minimal solution of some compacted equations, after instantiating N variables. This theorem fixes the increase w.r.t. 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 4.6 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 $|G'| \leq |G| + depth(G)$. If we use only 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.

The combined use of size and depth allows us to keep track of balancing conditions of singleton grammars as trees and also to provide tighter measures.

Remark 2. Our method computes a compact representation of a size-minimal solution. This means that every solvable MSOU problem has at least one solution that can be represented by a polynomial-sized grammar. Our method can easily be extended to compute a compact representation of any solution; however, there is no longer any size-bound. If one is interested in representing all solutions, then our method does not help, since singleton grammars do not support the representation of infinite sets of words; e.g., the representation of $\{(a\,b)^n \mid n \in \mathbb{N}\}$ is not possible. Note that there is already an investigation of a representation of sets of solutions using words with exponents for MSOU (see [2]).

9. Conclusions. In this paper we proved in Corollary 8.2 that monadic second-order unification (MSOU) is in NP using a result of Plandowski about context-free grammars [17, 18] and the exponential bound on the exponent of periodicity [23, 22]. These results, together with the NP-hardness of the problem [22], prove its NP-completeness. As we mention in the introduction, MSOU is a specialization of bounded second-order unification (BSOU) [22], a variant of second-order unification, where instantiations of second-order variables can use their argument a bounded number of times. During revision of this paper we were able to apply variants of this method to prove that BSOU [11] and stratified context unification [12] are also NP-complete.

Acknowledgment. We acknowledge the meticulous reading and helpful comments of the anonymous referees, which helped us to improve the presentation of the paper.

REFERENCES

[1] G. Dowek, *Higher-order unification and matching*, in Handbook of Automated Reasoning, Vol. II, A. Robinson and A. Voronkov, eds., Elsevier Science, Amsterdam, 2001, pp. 1009–1062.

[2] W. M. Farmer, *A unification algorithm for second-order monadic terms*, Ann. Pure Appl. Logic, 39 (1988), pp. 131–174.

[3] W. M. Farmer, *Simple second-order languages for which unification is undecidable*, Theoret. Comput. Sci., 87 (1991), pp. 173–214.

[4] M. R. Garey and D. S. Johnson, *"Computers and Intractability": A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[5] W. D. Goldfarb, *The undecidability of the second-order unification problem*, Theoret. Comput. Sci., 13 (1981), pp. 225–230.

[6] G. Huet, *A unification algorithm for typed λ-calculus*, Theoret. Comput. Sci., 1 (1975), pp. 27–57.

[7] M. Karpinski, W. Rytter, and A. Shinohara, *An efficient pattern-matching algorithm for strings with short descriptions*, Nordic J. Comput., 4 (1997), pp. 172–186.

[8] A. Kościelski and L. Pacholski, *Complexity of Makanin's algorithm*, J. ACM, 43 (1996), pp. 670–684.

[9] J. Levy, *Decidable and undecidable second-order unification problems*, in Proceedings of the 9th Annual International Conference on Rewriting Techniques and Applications (RTA'98), Lecture Notes in Comput. Sci. 1379, Springer-Verlag, Berlin, 1998, pp. 47–60.

[10] J. Levy, M. Schmidt-Schauß, and M. Villaret, *Monadic second-order unification is NP-complete*, in Proceedings of the 15th Annual International Conference on Rewriting Techniques and Applications (RTA'04), Lecture Notes in Comput. Sci. 3091, Springer-Verlag, Berlin, 2004, pp. 55–69.

[11] J. Levy, M. Schmidt-Schauß, and M. Villaret, *Bounded second-order unification is NP-complete*, in Proceedings of the 17th Annual International Conference on Rewriting Techniques and Applications (RTA'06), Lecture Notes in Comput. Sci. 4098, Springer-Verlag, Berlin, 2006, pp. 400–414.

[12] J. Levy, M. Schmidt-Schauß, and M. Villaret, *Stratified context unification is NP-complete*, in Proceedings of the 3rd Annual International Joint Conference on Automated Reasoning (IJCAR'06), Lecture Notes in Comput. Sci. 4130, Springer-Verlag, Berlin, 2006, pp. 82–96.

[13] J. Levy and M. Veanes, *On the undecidability of second-order unification*, Inform. and Comput., 159 (2000), pp. 125–150.

[14] J. Levy and M. Villaret, *Currying second-order unification problems*, in Proceedings of the 13th Annual International Conference on Rewriting Techniques and Applications (RTA'02), Lecture Notes in Comput. Sci. 2378, Springer-Verlag, Berlin, 2002, pp. 326–339.

[15] Y. Lifshits, *Solving Classical String Problems on Compressed Texts*, The Computing Research Repository (CoRR); available online from http://www.avxiv.orgabs/cs/0604058 (2006).

[16] G. S. Makanin, *The problem of solvability of equations in a free semigroup*, Sb. Math. USSR, 32 (1977), pp. 129–198.

[17] W. Plandowski, *Testing equivalence of morphisms on context-free languages*, in Proceedings of the Second Annual European Symposium on Algorithms (ESA'94), Lecture Notes in Comput. Sci. 855, Springer-Verlag, London, 1994, pp. 460–470.

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

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

[20] W. Plandowski, *Satisfiability of word equations with constants is in PSPACE*, J. ACM, 51 (2004), pp. 483–496.

[21] M. Schmidt-Schauß, *Stratified context unification is in PSPACE*, in Proceedings of the 15th International Workshop in Computer Science Logic (CSL'01), Lecture Notes in Comput. Sci. 2142, Springer-Verlag, Berlin, 2001, pp. 498–512.

[22] M. Schmidt-Schauß, *Decidability of bounded second order unification*, Inform. and Comput., 188 (2004), pp. 143–178.

[23] M. Schmidt-Schauß and K. U. Schulz, *On the exponent of periodicity of minimal solutions of context equations*, in Proceedings of the 9th Annual International Conference on Rewriting Techniques and Applications (RTA'98), Lecture Notes in Comput. Sci. 1379, Springer-Verlag, Berlin, 1998, pp. 61–75.

[24] A. P. Zhezherun, *Decidability of the unification problem for second-order languages with unary function symbols*, Kibernetika (Kiev), 5 (1979), pp. 120–125 (in Russian); Cybernetics, 15 (1980), pp. 735–741 (in English).