On the Complexity of Bounded Second-order Unification and Stratified Context Unification

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

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

Abstract. Bounded Second-Order Unification is a decidable variant of undecidable Second-Order Unification. Stratified Context Unification is a decidable restriction of Context Unification, whose decidability is a long-standing open problem. This paper is a join of two separate previous, preliminary papers on NP-completeness of Bounded Second-Order Unification and Stratified Context Unification. It clarifies some omissions in these papers, joins the algorithmic parts that construct a minimal solution, and gives a clear account of a method of using singleton tree grammars for compression that may have potential usage for other algorithmic questions in related areas.

1 Introduction

Bounded Second-Order Unification (BSOU) is a decidable variant of the undecidable Second-Order Unification Problem [Gol81]. Stratified Context Unification (SCU) is a decidable restriction of Context Unification, whose decidability is a long-standing open problem. This paper is a join of two separate previous, preliminary papers on NP-completeness of Bounded Second-Order Unification [LSSV06a] and Stratified Context Unification [LSSV06b], which adds more explanation and proofs, and also gives a simplified account of the common structure of the algorithmic and the proof parts. The main idea of the proof is, given a unification problem, to guess a polynomially-sized (compressed) representation of a minimal unifier, and then test (in polynomial time) whether this is really a unifier. This is a paradigmatic method to show that a unification problem is in NP. We will explain the details below.

Second-Order Unification (SOU) is unification in the simply typed lambda calculus [Dow01] restricted to terms with variables of order at most two and function symbols of order at most three. In other words, the problem of, given two λ-terms of the same type that satisfy the previous conditions, deciding if

there exists a substitution of free variables (unknowns) by equally typed terms, such that when applied to both terms result in the same term modulo $\alpha\beta\eta$-equality. Second-Order Unification is undecidable [Gol81]. It is undecidable even under severe restrictions like the number of second-order variables (just one), their number of occurrences (just four) or their arity [Far91,LV00]. In addition, all these languages require having function symbols with arity at least two. In [Far91] and [LV02] it is proved that one single binary function symbol is enough to get undecidability. The fragment, where arities of function symbols is at most one, called *Monadic Second-Order Unification (MSOU)*, is decidable [Far88]. In [LSSV04] and [LSSV08] we prove the NP-completeness of MSOU, where the method of guessing a compressed representation (as a string) and then checking unifiability, is similar to the method used in this paper. Apart from these syntactic restrictions on the equations of Second-Order Unification, there may be semantic restriction on the permitted solutions.

Bounded Second-Order Unification (BSOU) is a variant of Second-Order Unification where instantiation of second-order variables in the unifier can use their arguments only a bounded number of times. In [SS04] the decidability of this problem is proved, where the algorithm has at least a worst-case exponential execution time. In this paper we investigate a simplification of it: second-order variables are at most unary and instantiation can use their argument once or ignore the argument. In [SS04] the general case of Bounded Second-Order Unification is NP-reduced to this restricted case.

Context Unification (CU) is typically defined as an extension of First-Order Unification where context variables are allowed. These variables have arity one and can only be instantiated by contexts, i.e. terms with a hole, where their argument will be "plugged in". Context Unification can also be defined as a variant of Second-Order Unification with the syntactic restriction of having at most unary variables, and the semantic restriction is that instances of second-order variables use their argument exactly once. Despite the similarities between Context Unification and the simplification of Bounded Second-Order Unification that we are considering, decidability of Context Unification is an open problem, even under the restriction of having at most one binary function symbol [LV02]. Nevertheless, there are some decidable fragments, like the case where at most two second-order variables are permitted [SSS02] or the case where at most two occurrences per variable are permitted [Lev96], and other variants [LNV05,KLV07].

Stratified Context Unification (SCU) is a fragment of Context Unification where the nesting of second-order variables is restricted to be the same for all occurrences of the same variable. Stratified Context Unification is decidable [SS02]. As we can see from the definitions of the problems, Bounded Second-Order Unification and Stratified Context Unification are quite similar problems. Also the algorithms have lots in common, though there are also significant differences in semantics and in the algorithm.

Minimal-size solutions for these problems may be exponentially large in the size of the equations. In this paper, we show that for any bounded second-order, or context equation $s \stackrel{?}{=} t$, and any minimal solution σ, we can represent $\sigma(s)$

and $\sigma(t)$ by means of a polynomially sized *Singleton Tree Grammar (STG)*, where the main purpose is a compression of terms. STGs are a generalization of singleton context free grammars, which can compress words. This method is a variant of so-called straight-line programs [PR99], which are used to describe and analyze compression technique for words. A central theorem for these compression techniques is the Theorem of Plandowski, that shows that the equality test of two compressed words can be done in polynomial time in the size of the compression [Pla94,Pla95]. For more information and complexity analyses see [Loh06,Lif07]. In [BLM05,SS05] and [Lif06,Lif07] it is proved that, given a singleton tree grammar, we can decide in polynomial time ($\mathcal{O}(n^3)$) on the size n of the grammar whether two non-terminals define the same word, by using the corresponding results for words. These results serve us to show the NP-ness of Bounded Second-Order Unification and Stratified Context Unification, and hence, together with their NP-hardness proved in [SS04] and [SS02], to obtain their NP-completeness. Similar techniques to describe efficient algorithms and good complexity bounds for unification algorithms using SCFGs are in [LSSV08] for solving monadic second-order unification. Efficient context-matching and first-order matching using STGs is described in [GGSS08], and efficient first-order unification of already compressed terms in [GGSS09].

The ideas behind the proof of the bound on the size of the grammar for representing size-minimal solutions are the following. Typical proofs of unification decidability/complexity start by proving that some unifier σ of a problem $s \stackrel{?}{=} t$ can be decomposed as

$$\sigma = [X_n \mapsto u_n] \circ \cdots \circ [X_1 \mapsto u_1] \tag{1}$$

where n is bounded by some function of the size of the problem, and u_i's can be constructed from a bounded number of pieces of the previous partial instances

$$[X_{i-1} \mapsto u_{i-1}] \circ \cdots \circ [X_1 \mapsto u_1](s) \stackrel{?}{=} [X_{i-1} \mapsto u_{i-1}] \circ \cdots \circ [X_1 \mapsto u_1](t) \tag{2}$$

From the proof, we can usually derive an algorithm that finds the unifier as follow. For $i = 1, \ldots, n$, we iteratively find the term u_i, and apply the substitution $[X_i \mapsto u_i]$ to the problem.

In the case of First-Order Unification, the situation is very simple. Variables $X_1, \ldots, X_n$ are original variables from the problem, where n is bounded by the size of the problem. Moreover, since u_i's are subterms of the original problem, we do not need to instantiate the problem each time we find one of these u_i's and may reuse them. This results in a polynomial version of the well-known Robinson-algorithm that works on term-dags.

Decidability proofs of Bounded Second-Order Unification and Stratified Context Unification also follow this schema. However, to prove the tight complexity bound of this paper, we will follow a quite different approach. We will prove that for size minimal solutions we only need a polynomially bounded number or partial instances like (2), and that the terms and contexts u_i's required to obtain these partial instances can be built using a polynomial number of pieces

like prefixes, suffixes, concatenations of contexts and subcontexts from

$$[X_{i-1} \mapsto u_{i-1}] \circ \cdots \circ [X_1 \mapsto u_1](s) \stackrel{?}{=} [X_{i-1} \mapsto u_{i-1}] \circ \cdots \circ [X_1 \mapsto u_1](t)$$

In some cases, we will also require to rise the resulting contexts to a power exponentially bounded by the size of the original problem. The proofs of these properties of minimal-size solutions rely also on the algorithms described in [SS04,SS02].

From these results we can not directly prove the NP-ness of these problems. Although everything is polynomially bounded, the need of instantiating $s \stackrel{?}{=} t$ as (2) such that u_i can be computed from it, makes the size of the problem increasing as a composition of a polynomially bounded number of polynomials, and this is not just a polynomial. So we need other methods to show a polynomial size bound.

In Section 3 we prove that, using Singleton Tree Grammars, we can represent these minimal solutions in polynomial space. The basic idea is that given a grammar that represents (2), we can also represent $[X_i \mapsto u_i] \circ \cdots \circ [X_1 \mapsto u_1](s) \stackrel{?}{=} [X_i \mapsto u_i] \circ \cdots \circ [X_1 \mapsto u_1](t)$ by extending the grammar. As we have said, u_i is built reusing pieces resulting from prefixes, suffixes, concatenations of contexts, subcontexts and exponentiations. We can extend the grammar in a controlled manner to construct these pieces, and at the end, these extensions result in a polynomial size STG. An improvement over earlier techniques is to use the so-called *Vdepth* that allows us to show that a polynomial number of instantiations of variables, as done in constructing unifiers in a unification algorithm, leads to polynomial space increase. This technique is already used in [GGSS09] for first-order unification and extended in this paper to instantiations of second-order variables.

The structure of the paper follows the main ideas in the proof. In Section 2 we define some necessary notations and notions. In Section 3, the grammar mechanism of STGs is described. In particular, the construction methods for new pieces are explained, and detailed method for estimating the size increase by different constructions is given. In Section 4 there is a joint construction method for minimal-size solutions of bounded second-order unification problems as well as for stratified context unification problems. Finally, we can summarize and present the obtained results in Section 5.

2 Preliminaries

We use a *signature* $\Sigma = \bigcup_{i \geq 0} \Sigma_i$, where constants of Σ_i are i-ary, and a set of *variables* $\mathcal{X} = \bigcup_{i=0,1} \mathcal{X}_i$, where variables of $\mathcal{X}_i$ are also i-ary. Variables of $\mathcal{X}_0$ are therefore *first-order variables* and those of $\mathcal{X}_1$ are (unary) *second-order variables*. We assume that $\Sigma_0 \neq \emptyset$ and $\Sigma_2 \neq \emptyset$. Notice that we do not consider second-order variables with arity greater than one. We denote variables with capital letters Z, if it may be first-order as well as second-order variables, and use the convention that X, Y mean second-order variables, and x, y, z mean first-order variables. Constants are denoted by lower-case letters a, b, f, g,... respectively, and the

arity of a constant f is denoted as $ar(f)$. *First-order terms* are built using the grammar $t ::= x \mid f(t_1, \ldots, t_{ar(f)}) \mid X(t)$, where $f \in \Sigma$, $x \in \mathcal{X}_0$, and $X \in \mathcal{X}_1$. *Second-order terms* or *functions* are built using the grammar $s ::= \lambda z.t$, where t is a first-order term, and z a first-order variable. Notice that, like for variables, we do not consider terms with more than one parameter. Terms are denoted as $r, s, t, u, v, \ldots$.

The set of variables occurring in terms or other syntactic objects is denoted as $Var(\cdot)$. A term without occurrences of free variables is said to be *ground*. The *size* of a term t is denoted $|t|$. It is defined for first-order terms as their number of symbols, and for second-order terms $\lambda z.t$ as the number of symbols of t but not counting occurrences of z. We use *positions* in terms, denoted p, q, as sequences of non-negative integers following Dewey notation. In $f(t_1, \ldots, t_n)$ or $X(r)$, respectively, the position of the function symbol and the second-order variable is the empty word denoted as ε, and the position of the i^{th} argument is i. The symbol at position ε is also called the *head* of the term, $p \prec q$ denotes the prefix relation, $p \cdot q$ the concatenation, and $t|_p$ the subterm at position p of t.

Terms that contain a single occurrence of the *hole* $[\cdot]$, which is syntactically like an extra 0-ary constant, are called *contexts*. We denote contexts by lower case letters $c, d, \ldots$. If the term s or context d, respectively, is plugged into the hole of c, i.e. the hole is replaced by s, or d, respectively, then we denote the result as the term $c[s]$, or the context $c[d]$, respectively. The latter is also denoted as $c \cdot d$. The position of the hole in a context d is called *main path*, denoted $mp(d)$, and the length of the main path is called the *main depth* of d. If $d_1 = d_2[d_3]$, for contexts d_i, then d_2 is called a *prefix* of d_1, and d_3 is called a *suffix* of d_1. Concatenation $c_1[\ldots[c_n]\ldots]$ is written $c_1 \cdot \ldots \cdot c_n$. The notation d^n, for a context d and $n \in \mathbb{N}$, means concatenation of n copies of the context d. If $t = d[s]$ for some s, then d is a *prefix context* of the term t. A *subcontext* of a context or term is a prefix of some suffix or a prefix context of some subterm. A second-order term $\lambda z.t$ where z occurs in t exactly once is called a *linear function*, and if z does not occur in t, a *constant term function*. Contexts and linear terms are equivalent, and the hole replacement can be seen as a function application. Thus, we will not distinguish between the linear term $\lambda z.f(z)$ and the context $f[\cdot]$. Therefore, the *size* of a context is its size as a term but not counting the hole, and the size of a constant function $\lambda z.t$ is the size of t as a term.

Second-order substitutions, denoted by greek letters $\sigma, \theta, \ldots$, are functions from terms to terms, defined as usual, where first-order-variables are mapped to first-order terms, and second-order variables are mapped to second-order terms. When all second-order variables are mapped to linear functions (i.e. contexts), we call it a *context substitution*, and when all second-order variables are mapped to either linear or constant term functions, we call it a *bounded substitution*. The application of a substitution σ to a term t is written $\sigma(t)$, where we always assume that the result is a term, i.e. if $\sigma = [X \mapsto c]$ or $\sigma = [X \mapsto \lambda z.c[z]]$, then $\sigma(X(s)) = c[\sigma(s)]$, and if $\sigma = [X \mapsto \lambda z.t]$, where z does not occur in t, then $\sigma(X(s)) = t$.

2.1 Second-Order Unification Problems

We consider two kinds of unification problems in this paper: stratified context unification problems and bounded second-order unification problems. Note that in [SS04], a more general condition is used, but it is shown there that the general case can be NP-reduced to the case considered here, under mild restrictions.

A *second-order unification problem* is a *set of equations* $E = \{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\}$, where t_i and u_i are first-order terms. The size of an equation E is denoted as $|E|$ and is the number of its symbols. We assume that equations are symmetric. A second-order substitution σ is said to be a *bounded unifier* (*context unifier*, respectively) of E, if σ is a bounded substitution (context substitution, respectively), and for all $i = 1, \ldots, n$, $\sigma(t_i) = \sigma(u_i)$. A (bounded or context) unifier σ is said to be a (bounded or context) *solution* of E, if for all $i = 1, \ldots, n$ $\sigma(t_i)$ and $\sigma(u_i)$ are ground. If E has a bounded unifier (context unifier, respectively), then we say that E is *bounded-unifiable* (*context-unifiable*, respectively). Similarly for solutions and solvability.

It is easy to see that the following holds (the proof relies on the assumption $\Sigma_0 \neq \emptyset$):

Lemma 2.1.

1. *For every bounded-solvable set of equations E, there exists a bounded solution σ, such that every function symbol g with $ar(g) \geq 1$ occurring in $\sigma(E)$, also occurs in E.*
2. *For every set of equations E, we have E is bounded-unifiable if, and only if, E is bounded-solvable.*

The next lemma is specialized to context unifiers and solutions (the proof relies on the assumptions $\Sigma_0 \neq \emptyset$ and $\Sigma_2 \neq \emptyset$):

Lemma 2.2.

1. *For every context-solvable set of equations E, if E contains a function symbol f with $ar(f) \geq 2$, then there is also a context solution σ, such that every function symbol g with $ar(g) \geq 1$ occurring in $\sigma(E)$, also occurs in E.*
2. *For every context-solvable set of equations E, if all function symbols f occurring in E satisfy $ar(f) \leq 1$, then there exists a context solution σ, such that $\sigma(E)$ only contains function symbols occurring in E and at most one binary function symbol.*
3. *For every set of equations E, we have E is context-unifiable if, and only if, E is context-solvable.*

Note that the second case of Lemma 2.2 occurs in the equation $X(a) \stackrel{?}{=} Y(b)$. It has a context solution $[X \mapsto f(b, [\cdot]), Y \mapsto f([\cdot], a)]$, but it has no context solution using only the symbols occurring in the equation. Note, however, that there is a bounded solution $[X \mapsto \lambda z.a, Y \mapsto \lambda z.a]$ that only uses function symbols of the equation.

Since we assume that the signature contains at least one binary function symbol, w.l.o.g. we can restrict E to consist of just one equation.

A bounded solution (context solution, respectively) σ of an equation E is said to be *size-minimal* if it minimizes $\sum_{Z \in Var(E)} |\sigma(Z)|$ among all bounded solutions (context solutions, respectively) of E. Size-minimal bounded solutions (context solutions) of a second-order problem satisfy the exponent of periodicity lemma [Mak77,KP96,SSS98,SS02,SS04]:

Lemma 2.3 ([SS02,SS04]). *There exists a constant $\alpha \in \mathbb{R}$ such that, for every equation E, and every size-minimal bounded solution (context solution, respectively) σ, every variable Z, every nontrivial context d, and any $n \in \mathbb{N}$, if d^n is a subcontext of $\sigma(Z)$, then $n \leq 2^{\alpha|E|}$.*

In the following, we denote by $\mathrm{eop}(\sigma)$ the maximal n such that, for nontrivial d, $d^n([\cdot])$ is a subcontext of $\sigma(Z)$, for some variable Z.

The next lemma helps us to avoid the use of constant term functions in the construction of a compressed solution.

Lemma 2.4. *For every equation E and every size-minimal bounded solution σ, we can find a decomposition $\sigma' \circ \rho$ of σ, such that $\sigma(Z) = (\sigma' \circ \rho)(Z)$ for all variables $Z \in Var(E)$ and σ' is a size-minimal context solution of $\rho(E)$ and ρ has the form $\rho = [X_1 \mapsto \lambda z.x'_1, \ldots, X_n \mapsto \lambda z.x'_n]$, for some second-order variables $X_1, \ldots, X_n$ and first-order variables $x'_1, \ldots, x'_n$.*

Proof. Let σ be $\sigma = [X_1 \mapsto \lambda z.t_1, \ldots, X_n \mapsto \lambda z.t_n, Y_1 \mapsto c_1, \ldots, Y_m \mapsto c_m, x_1 \mapsto u_1, \ldots, x_r \mapsto u_r]$, where we have distinguished between variables X_i that are instantiated by constant functions, and variables Y_j that are instantiated by contexts. We can decompose σ as $\rho = [X_1 \mapsto \lambda z.x'_1, \ldots, X_n \mapsto \lambda z.x'_n]$ and $\sigma' = [x'_1 \mapsto t_1, \ldots, x'_n \mapsto t_n, Y_1 \mapsto c_1, \ldots, Y_m \mapsto c_m, x_1 \mapsto u_1, \ldots, x_r \mapsto u_r]$, for some fresh first-order variables $x'_1, \ldots, x'_n$. It is easy to see that if σ' is not minimal for $\rho(E)$, then σ is not minimal for E.

We say that a set of equations E is *stratified* if for every variable $Z \in Var(E)$, for every pair of occurrences p_1, p_2 of Z in the terms of E, the sequence of second-order variables on the path p_1 and on the path p_2 are the same. Here, we mean that X is *on the path* p in t, if for some prefix p' of p, $t_{|p'}$ is of the form $X(r)$. For example, $X(g(b, Y(b))) \stackrel{?}{=} X(Y(g(x, b)))$ is stratified, whereas $f(X(a)) \stackrel{?}{=} Y(f(X(a)))$ is not stratified.

Bounded Second-Order Unification is defined as the problem of deciding if a given set of second-order equations E has a bounded unifier, and *Stratified Context Unification* is defined as the problem of deciding if a given set of stratified second-order equations has a context unifier.

3 Singleton Tree Grammars (STG)

We define *Singleton Tree Grammars (STG)* as a generalization of Singleton Context Free Grammars (SCFG) [LSSV04,Pla94], extending the expressivity

of SCFGs by terms and contexts. This is consistent with the definitions given by [SS05] and [BLM05], with straight line programs, and with the context free tree grammars [Rou69,ES77,ES78,CDG+07]. However, it is a special case, where only one parameter is used. In a recent paper [LMSS09], it was shown that multi-parameter compression can be linearly encoded as one-parameter compression (as used here) if compression is concerned.

Definition 3.1 (Singleton Tree Grammar). *A* singleton tree grammar (STG) *is a 4-tuple $G = (\mathcal{T}, \mathcal{C}, \Sigma, R)$, where $\mathcal{T}$ are tree symbols, $\mathcal{C}$ are context symbols, and Σ is a signature of terminal symbols, such that the sets $\mathcal{T}$, $\mathcal{C}$, Σ are pairwise disjoint.*

The rules in R may be of the form:

$$\begin{aligned}
A_1 &::= A_2 \\
A_1 &::= C[A_2] \\
A &::= f(A_1, \ldots, A_n) \\
C_1 &::= C_2 \\
C_1 &::= C_2 \cdot C_3 \\
C &::= [\cdot] \\
C &::= f(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_n)
\end{aligned}$$

where $A, A_i \in \mathcal{T}$ and $C, C_i \in \mathcal{C}$ are nonterminals, and $f \in \Sigma$, with arity $n \geq 0$, is terminal.
For every nonterminal $D \in \mathcal{T} \cup \mathcal{C}$ there is at most one *rule having D as left hand side.*

Given two nonterminals $D_1, D_2 \in \mathcal{T} \cup \mathcal{C}$, we say that $D_1 >_G D_2$, if D_2 occurs in the right-hand side of the rule deriving D_1. The STG must be non-recursive, i.e. the transitive closure $>_G^+$ must be terminating.

Given a term t with occurrences of nonterminals, the derivation $\xrightarrow{*}_G$ *by G is an exhaustive iterated replacement of the nonterminals by the corresponding right hand sides, using the convention for second-order terms, until there are no more applicable rules. The result is denoted as $val_G(t)$, and may contain non-terminal symbols, because there can be non-terminal symbols without deriving rules.*

In the case of nonterminals $A \in \mathcal{T}$ and $C \in \mathcal{C}$, we also say that G defines *$val_G(A)$ or $val_G(C)$, respectively.*

If the grammar G is clear, we omit the index in our notation. Usually, less rule possibilities are sufficient for the expressiveness, e.g. the rules $C_1 ::= C_2$ and $A_1 ::= A_2$ could be eliminated, however, we keep them, since during generation of instantiations of terms in Section 4, we have to add such rules.

Definition 3.2 (Size and Depth of a Grammar). *The* size *$|G|$ of a grammar (STG) G is the number of its rules.*

The depth *of a nonterminal D w.r.t. a grammar G, denoted $depth_G(D)$, is defined as the maximal number of $>_G$-steps from D.*

The depth of a grammar*, denoted as $depth(G)$, is the maximum of the depths of all nonterminals.*

As a generalization of a theorem by [Pla94,Pla95], (see also [BLM05], [SS05] and [Lif06,Lif07]) the following theorem holds:

Theorem 3.3. *Given an STG G, and two tree nonterminals A, B from G, it is decidable in polynomial time $O(|G|^3)$ whether $val_G(A) = val_G(B)$.*

3.1 Depth of a Grammar Relative to Sets of Non-Terminals

We can generalize the definition of depth of a nonterminal, making it relative to a set of nonterminals. This technique was introduced by [GGSS09], and is extended here. The use of this measure, in addition to the depth and size of the grammar, allows us to avoid the use of complicated compression techniques in [LSSV06a].

Definition 3.4 (Vdepth). *The Vdepth of a nonterminal symbol D w.r.t. a grammar G and a subset of nonterminals V of G, denoted $Vdepth_G(D, V)$, is defined as $depth_{G'}(D)$, where G' is constructed from G by removing all the rules deriving v, for $v \in V$, i.e. by treating all symbols in V as terminals.*

The number $Vdepth(G, V)$ is defined as the maximum of all $Vdepth_G(D, V)$, for all nonterminals D of G.

In the following, when we say an STG (G, V) we mean a grammar G and a subset V of its nonterminal symbols.

Lemma 3.5. *For any STG G, and set of nonterminal symbols V*

$$depth(G) < (Vdepth(G, V) + 1)\,(|V| + 1)$$

Proof. Since $<_G$ is not cyclic, a maximal $<_G$-chain is as follows

$$D_{1,1} <_G \cdots <_G D_{1,n_1} <_G v_1 <_G D_{2,1} <_G \cdots <_G D_{2,n_2} <_G v_2 <_G \\ \cdots <_G v_{|V|} <_G D_{|V|+1,1} <_G \cdots <_G D_{|V|+1,n_{|V|+1}}$$

where $v_i \in V$. There are at most $|V| + 1$ subsequences without symbols of V, and the maximal length of such a sequence is $Vdepth(G, V)$.

3.2 Grammar Extensions

The following lemmas state how the size and the *Vdepth* of the grammar are increased by extending the STG with concatenations, exponentiation, prefixes, suffixes and subterms of contexts, and subterms and subcontexts of terms. When using log, we mean the binary logarithm.

The Vdepth/depth/size bounds for these operations are related to balancing conditions for trees. In other words, the main idea is to use balanced concatenations of a list of contexts, whenever possible. For instance, if we want to represent d^{16} where $d = f(g([\cdot]))$, then this can be done by concatenating a sequence of 16 d's: $d \cdot \cdots \cdot d$. First we add the rules $D ::= D' \cdot D'', D' ::= f([\cdot])$ and $D'' ::= g([\cdot])$. The unbalanced possibility is to do it sequentially like

$D_1 ::= D \cdot D, D_2 ::= D_1 \cdot D, \ldots, D_{15} ::= D_{14} \cdot D$, which produces an STG of depth 15. Constructing d^{16} by a divide-and-conquer method produces a balanced grammar: $\{D_1 ::= D \cdot D, D_2 ::= D_1 \cdot D_1, D_3 ::= D_2 \cdot D_2, D_4 ::= D_3 \cdot D_3\}$ of depth 4, which is logarithmic in the number of contexts that have to be concatenated.

Definition 3.6 (Grammar Extension). *We say that a STG $G' = (\mathcal{T}', \mathcal{C}', \Sigma, \mathcal{R}')$ is a grammar extension of another STG $G = (\mathcal{T}, \mathcal{C}, \Sigma, \mathcal{R})$, denoted $G' \supseteq G$, if $\mathcal{T}' \supseteq \mathcal{T}$, $\mathcal{C}' \supseteq \mathcal{C}$ and $\mathcal{R}' \supseteq \mathcal{R}$.*

Lemma 3.7 (Combining). *Let G be an STG with an n-ary function symbol f in its signature and defining $n-1$ terms $t_1, \ldots t_{n-1}$. Then, there exists a grammar extension $G' \supseteq G$ that, for a given position of the hole, defines the context $f(t_1, \ldots, [\cdot], \ldots, t_{n-1})$ and satisfies, for every set V,*

$$|G'| \leq |G| + 1$$
$$Vdepth(G', V) \leq Vdepth(G, V) + 1$$

Proof. Let $A_1, \ldots A_{n-1}$ be the non-terminals of G defining the terms $t_1, \ldots t_{n-1}$ respectively. We simply need to add the rule $A ::= f(A_1, \ldots, [\cdot], \ldots A_{n-1})$.

Lemma 3.8 (Concatenation). *Let G be an STG defining the contexts $c_1, \ldots, c_n$, for $n \geq 1$. Then there exists a grammar extension $G' \supseteq G$ that defines the context $c_1 \cdot \cdots \cdot c_n$ and satisfies, for every set V,*

$$|G'| \leq |G| + n - 1$$
$$Vdepth(G', V) \leq Vdepth(G, V) + \log n + 1$$

Proof. The construction is by divide and conquer and adding fresh nonterminals. First, add rules to construct the concatenation of the first $n/2$ contexts such that the nonterminal C_1 defines this concatenation.Similarly, let C_2 be the context defining the second half. Then construct the whole concatenation adding the rule $C_3 ::= C_1 \cdot C_2$. The *Vdepth* bound is then obvious.

Lemma 3.9 (Exponentiation). *Let G be an STG defining the context c. For any $n \geq 1$, there exists a grammar extension $G' \supseteq G$ that defines the context c^n and satisfies, for every set V,*

$$|G'| \leq |G| + 2 \log n$$
$$Vdepth(G', V) \leq Vdepth(G, V) + \log n + 1$$

Proof. The proof uses the same ideas as the previous lemma.

Lemma 3.10 (Prefix and Suffix). *Let G be an STG defining the context c. For any nontrivial prefix or suffix c' of the context c, there exists a grammar extension $G' \supseteq G$ that defines c', and satisfies, for every set V,*

$$|G'| \leq |G| + depth(G) - 1$$
$$Vdepth(G', V) \leq Vdepth(G, V) + \log(depth(G)) + 1$$

Proof. For suffixes, let C be the nonterminal that derives into the context c. We recursively compute a list of contexts $list(C)$ as follows (written as a sequence): if the rule that derives C is $C ::= C'$, then $list(C) = list(C')$; if the rule is $C ::= [\cdot]$, then $list(C) = [\cdot]$; and if the rule is $C ::= f(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_n)$, then $list(C) = f(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_n)$. Finally, if the rule is $C ::= C_1 \cdot C_2$, then depending on where the hole of c' should be, either $list(C) = list(C_2)$ or $list(C) = list(C_1); C_2$. This will produce a list of at most $depth(G)$ nonterminals of G. The concatenation will represent the desired suffix c' of c. Then, Lemma 3.8 gives the stated bounds.

For prefixes, we first compute a list of contexts top-down and then construct the concatenation. This may produce a list of $depth(G)$ contexts. Thus, $Vdepth(G') \leq Vdepth(G) + \log(depth(G)) + 1$

Lemma 3.11 (Subterm). *Let G be an STG defining the context c or term t. For any nontrivial subterm t' of the context c or of the term t, there exists a grammar extension $G' \supseteq G$ that defines t' and satisfies, for every set V,*

$$\begin{aligned} &|G'| \leq |G| + depth(G) \\ &Vdepth(G', V) \leq Vdepth(G, V) + \log(depth(G)) + 2 \end{aligned}$$

Proof. There are two possibilities: either there is already a tree nonterminal defining u, then we are finished, or, by recursively descending, there is a rule $A_1 ::= C[A_2]$, and t' has to be constructed as $C_2[A_2]$, where C_2 defines a suffix of C. So, we have the same estimations as for the suffix given by Lemma 3.10, but there is one additional symbol and rule and a possible further increase of $|G|$ and $Vdepth(G', V)$ by 1.

Lemma 3.12 (Subcontext). *Let G be an STG defining the term t. For any nontrivial prefix context c of the term t, there exists a grammar extension $G' \supseteq G$ that defines c and satisfies, for every set V,*

$$\begin{aligned} |G'| &\leq |G| + depth(G)(depth(G) + 3/2) \\ Vdepth(G') &\leq Vdepth(G) + 2\log(depth(G)) + 4 \end{aligned}$$

Proof. Let A be the nonterminal symbol defining the term $t = val_G(A)$ and let p be the main path of c.

First we show by induction that we can extend the grammar and generate a list of context nonterminals that can be concatenated to construct c. The induction is on $depth_G(A)$.

The base case is that $depth_G(A) = 0$, that implies $c = [\cdot]$ and $|p| = 0$. In this case the list is empty.

For the induction step we consider the (nontrivial) different possibilities for rules deriving A:

1. The rule is $A ::= f(A_1, \ldots, A_n)$ and $p = k\,p'$. Then, we extend the grammar with the rule $C_1 ::= f(A_1, \ldots, [\cdot]_k, \ldots, A_n)$, where C_1 is a fresh context (nonterminal) symbol. The list of context nonterminals is then C_1 concatenated with the list generated inductively for A_k and p', where $depth_G(A_k) \leq depth_G(A) - 1$.

2. The rule is $A ::= C[A']$. There are some subcases:
   (a) If p is a prefix of $mp(val_G(C))$, then we construct the list of context nonterminals, as in the proof of Lemma 3.10, for the prefix of $val_G(C)$ until position p. This list has length at most $depth_G(C)$. The grammar has to be extended with at most $depth_G(C)$ new rules.
   (b) If $mp(val_G(C))$ is a prefix of p, then $p = mp(val_G(C))\,p'$, for some p', and we construct the list of contexts nonterminals as C concatenated with the list generated inductively for A' and p', where $depth_G(A') \leq depth_G(A) - 1$.
   (c) Otherwise, the position p is within $val_G(C)$ but it is not a prefix of $mp(val_G(C))$. Then, $p = p'\,k\,p''$ and $mp(val_G(C)) = p'\,k'\,p'''$, for some $k \neq k'$, and for some p', p'' and p'''. Hence, p' is the longest common prefix of p and $mp(val_G(C))$. Since contexts are unary, we have a n-ary function symbol f with $n \geq 2$ in the splitting point of the two paths. Therefore, there must be a rule $C' ::= f(B_1, \ldots, [\cdot]_{k'}, \ldots B_n)$, where $k \neq k'$. Assume w.l.o.g. $k < k'$. We extend the grammar with the rules

$$\begin{aligned} C_1 &::= C_2 \cdot C_3 \\ C_3 &::= f(B_1, \ldots, B_{k-1}, [\cdot], B_{k+1}, \ldots, B_{k'-1}, A_1, B_{k'+1}, \ldots, B_n) \\ A_1 &::= C_4[A'] \end{aligned}$$

   where $C_1, \ldots, C_4$ and A_1 are fresh nonterminal symbols.
   We also add, as in the proof of Lemma 3.10, at most $depth_G(C)$ rules to derive from C_2 the prefix of $val_G(C)$ with main path p'. Therefore, $Vdepth_{G'}(C_2, V) \leq Vdepth(G, V) + \log(depth(G)) + 1$.
   We also add at most $depth_G(C)$ rules to derive from C_4 the suffix of $val_G(C)$, starting at $p'\,k'$ and with main path p'''. Therefore, $Vdepth_{G'}(C_4, V) \leq Vdepth(G, V) + \log(depth(G)) + 1$.
   In total, we introduce at most $2\,depth_G(C) + 4 \leq 2\,depth_G(A) + 2$ new rules, and get

$$\begin{aligned} Vdepth(C_1, V) = \max\{\ & Vdepth_{G'}(C_2, V) + 1, \\ & Vdepth_G(B_i, V) + 2, \qquad \text{for } i \neq k, k' \\ & Vdepth_{G'}(C_4, V) + 3, \\ & Vdepth_G(A', V) + 3\ \} \\ \leq\ & Vdepth(G, V) + \log(depth(G)) + 4 \end{aligned}$$

   Finally, we construct the list of context nonterminals as C_1 concatenated with the list generated inductively for B_k and p'', where $depth_G(B_k) \leq depth_G(A) - 2$.

The worst bound is obtained for case 2 (c). In this case, the list of context nonterminals obtained for A and p has length bounded by $depth_G(A)/2$. We can construct c as the concatenation of all the symbols of the list. By Lemma 3.8, this concatenation can be done adding at most $depth_G(A)/2 - 1$ new rules, and increasing the Vdepth in at most $\log(depth_G(A)/2) + 1$. Therefore, for the total grammar size we have

$$\begin{aligned} &|G'| \leq |G| + depth_G(A)/2 - 1 + depth_G(A)/2 \cdot \big(2\,depth_G(A) + 2\big) \\ &\leq |G| + depth(G)\,(depth(G) + 3/2) \end{aligned}$$

The Vdepth increase is bounded by $\log(depth_G(A)/2) + 1$ plus the maximal Vdepth of all the symbols of the list, i.e. by

$$\begin{aligned} &Vdepth(G', V) \leq \log(depth_G(A)/2) + 1 + Vdepth(G, V) + \log(depth(G)) + 4 \\ &\leq Vdepth(G, V) + 2\ \log(depth(G)) + 4 \end{aligned}$$

3.3 Representing Terms and Contexts with Grammars

In this subsection we describe how grammars can be used to define terms and instances of terms. There must be a connection between their respective signatures. Thus, constants and function symbols are exactly the terminal symbols of the grammar, and variables are a subset of the nonterminal symbols.

Definition 3.13 (Grammars Defining Terms and Contexts). *Given a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ and an STG $G = \langle \mathcal{T}, \mathcal{C}, \Sigma, R \rangle$, where $\mathcal{X}_0 \subseteq \mathcal{T}$ and $\mathcal{X}_1 \subseteq \mathcal{C}$, we say that the tree nonterminal $A \in \mathcal{T}$* defines *t, if $t = val_G(A)$.*

Similarly for a context c and a context nonterminal symbol C.

Accordingly, if A defines t (or C defines c, respectively) for some nonterminals of G, then we say that G defines *t (or c, respectively).*

Notice that in the previous definition first-order variables are a subset of the tree nonterminal symbols of G, and second-order variables are a subset of the context nonterminal symbols. Moreover, nonterminal symbols representing variables may have no deriving rules in G.

Context substitutions are naturally modelled as grammar extensions, where some rules deriving nonterminal symbols representing variables are added. For instance, the grammar $G = \{A ::= X[y]\}$ defines from A the term $t = X(y)$. Given $\sigma = [X \mapsto f([\cdot]), y \mapsto a]$, we can define $\sigma(t)$ from A using $G \subseteq G' = \{A ::= X[y], X ::= f([\cdot]), y ::= a\}$. Hence, the context substitution σ is modelled by the grammar extension $\{X ::= f([\cdot]), y ::= a\}$. Lemma 3.14 generalizes this idea. Vice versa, any grammar extension corresponds to a context substitution, as Lemma 3.15 states.

For bounded substitutions the situation is more complicated. For instance, $G = \{A ::= X[y]\}$ defines from A the term $t = X(y)$. Given $\sigma = [X \mapsto \lambda z.a]$, the grammar $G' = \{A ::= x', x' ::= a\}$ defines from A the term $\sigma(t)$, where x' is a first-order variable, hence a tree nonterminal symbol. Notice that, in this case, $G \not\subseteq G'$. This construction can be generalized as follows: given a STG G defining t and u, we can construct a smaller STG G' defining $[X \mapsto \lambda z.u](t)$. However, in this case, G' is not a grammar extension of G. We have not found a natural way to model bounded substitutions as grammar extensions.

Lemma 3.14. *If the STG G defines the term t and the term u (context c, respectively), then there exists a grammar extension $G' \supseteq G$ that defines $[Z \mapsto u](t)$ (that defines $[Z \mapsto c](t)$, respectively) and satisfies, for every set V,*

$$\begin{aligned} &|G'| \leq |G| + 1 \\ &Vdepth(G', V \cup \{Z\}) = Vdepth(G, V) \end{aligned}$$

Proof. Let A (or C) be the term (or context) nonterminal defining u (or c). Take $G' = G \cup \{Z ::= A\}$ (or $G' = G \cup \{Z ::= C\}$). The *Vdepth* of the new grammar does not change, since the new nonterminals are inserted into the variable set V.

Lemma 3.15. *For any STG G and any grammar extension $G' \supseteq G$, there exists a context substitution σ such that, for any term $t = val_G(A)$ defined by G, we have $\sigma(t) = val_{G'}(A)$.*

Proof. Let $\mathcal{X}$ be the set of non-terminals of G without any deriving rules in G, but with deriving rules in G'. Non-terminals of $\mathcal{X}$ define variables of t. Then, define σ as the substitution that instantiates $X \in \mathcal{X}$ by $val_{G'}(X)$. The substitution σ is a context substitution because the grammar extension replaces tree symbols by terms and context symbols by contexts, but not context symbols by terms.

3.4 Size Bounds for Iterated Grammar Constructions

In previous subsections we have described several grammar extensions $G' \supseteq G$ that allow us to define terms and contexts constructed from pieces already defined by G. We have also seen that, if the grammar G already defines two terms t and u, we can construct a grammar extension $G' \supseteq G$ that defines the instantiation of x by u in t. In this case, we have to measure the *Vdepth* relative to a bigger set $V' = V \cup \{x\}$ that also includes the instantiated variable, if we do not want to get a blow up in the value of the depth, that might result in non-polynomial blow up in the size of the grammar.

In Section 4, we will see how, given an equation E defined by a STG G, and a size-minimal solution σ, we can get a grammar extension $G' \supseteq G$ defining $\sigma(E)$. This will be done using the constructions described in this section a polynomial number of times. We call these constructions *grammar extension steps*.

In this subsection we also measure the grammar size and grammar *Vdepth* increase after applying a polynomial number of grammar extension steps. In some of the constructions there are additional parameters (the number of contexts that we concatenate, the value of the exponent) that also have to be bound.

Definition 3.16 (Grammar Extension Step). *We say that the pair $\langle G', V' \rangle$ is constructed from the pair $\langle G, V \rangle$ using an* α-bounded grammar extension step *if it can be constructed using Lemmas 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, or 3.14, where the exponent used in Lemma 3.9 is bounded by 2^α, and the number of concatenated contexts in Lemma 3.8 is bounded by α.*

Theorem 3.17. *If the grammar G has size $|G| = \mathcal{O}(n)$, and $\langle G', V' \rangle$ is constructed from $\langle G, \emptyset \rangle$ using $\mathcal{O}(n^k)$ many $\mathcal{O}(n)$-bounded grammar extension steps, then*

$$|G'| = \mathcal{O}(n^{5k+2})$$
$$depth(G') = \mathcal{O}(n^{2k+1})$$
$$Vdepth(G', V') = \mathcal{O}(n^{k+1})$$
$$|V'| = \mathcal{O}(n^k)$$

Proof. Let the sequence of grammar extension steps be $\langle G, \emptyset\rangle = \langle G_0, V_0\rangle, \ldots, \langle G_m, V_m\rangle = \langle G', V'\rangle$, where $m = \mathcal{O}(n^k)$.

Taking the worst case of Lemmas 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, and 3.14, we can construct the recurrences:

$$\begin{array}{l} |G_{i+1}| = |G_i| + \mathit{depth}^2(G_i) + 3\,\mathit{depth}(G_i) + \mathcal{O}(n) \\ \mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq \mathit{Vdepth}(G_i, V_i) + 2\,\log \mathit{depth}(G_i) + \mathcal{O}(n) \\ |V_{i+1}| \leq |V_i| + 1 \end{array}$$

These worst cases are calculated as follows. In Lemmas 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12, we have $V_{i+1} = V_i$, and the worst case is given by instantiation Lemma 3.14, which increases the size of V_i by at most one. For the *Vdepth*, the biggest increase is $2\,\log \mathit{depth}(G_i) + 5$, given in Lemma 3.12 or the α increment given by Lemma 3.9, where α is the bound of the extension step (in our case $\mathcal{O}(n)$). For the size, the worst increment is the $\mathit{depth}(G_i)(\mathit{depth}(G_i)+3)$ given by Lemma 3.12 or the $2\,\alpha$ increment given by Lemma 3.9, where again $\alpha = \mathcal{O}(n)$.

Since $m = \mathcal{O}(n^k)$ and $|V_i| \leq i$, we have $|V_i| = \mathcal{O}(n^k)$, for every $i = 0, \ldots, m$.

From this bound and Lemma 3.5, we have $\mathit{depth}(G_i) = \mathcal{O}(n^k)$ $\mathit{Vdepth}(G_i, V_i)$. Therefore, the recurrence for $\mathit{Vdepth}(G_i, V_i)$ may be replaced by

$$\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq \mathit{Vdepth}(G_i, V_i) + 2\,\log \mathit{Vdepth}(G_i, V_i) + \mathcal{O}(n). \qquad (3)$$

A first bound for these recurrence can be computed relaxing the inequality as $\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq 3\,\mathit{Vdepth}(G_i, V_i) + \mathcal{O}(n)$ that has as solution $\mathit{Vdepth}(G_i, V_i) = 3^i\,\big(\mathit{Vdepth}(G_0, V_0) + \mathcal{O}(n)\big) = 3^i\,\mathcal{O}(n)$. Replacing this approximated solution in (3) results in

$$\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq \mathit{Vdepth}(G_i, V_i) + 2\,\log\big(3^i\,\mathcal{O}(n)\big) + \mathcal{O}(n) = \mathit{Vdepth}(G_i, V_i) + 2\,i\,\mathit{log}3 + \mathcal{O}(n)$$

Using the bound $i < m = \mathcal{O}(n^k)$, we get the approximated solution $\mathit{Vdepth}(G_i, V_i) = \mathcal{O}(n^{2k})$. Replacing again this approximated solution in (3) results on

$$\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq \mathit{Vdepth}(G_i, V_i) + 2\,\log\big(\mathcal{O}(n^{2k})\big) + \mathcal{O}(n) = \mathit{Vdepth}(G_i, V_i) + \mathcal{O}(n)$$

Using again the bound $i < m = \mathcal{O}(n^k)$, we get the solution $\mathit{Vdepth}(G_i, V_i) = \mathcal{O}(n^{k+1})$.

Therefore, $\mathit{depth}(G_i) = \mathcal{O}(n^k)$ $\mathit{Vdepth}(G_i, V_i) = \mathcal{O}(n^{2\,k+1})$. Replacing this in the recursion for $|G_i|$ we get $|G_{i+1}| = |G_i| + \mathcal{O}(n^{4\,k+2})$. Hence, $|G_i| = \mathcal{O}(n^{5\,k+2})$.

For generalized SCU-equations, i.e. if an initial equation already has compressed subterms and subcontexts, we need a slightly modified upper bound computation: In this case the set of variables that may be instantiated is at most $|G|$, and hence polynomial, however, the exponent of periodicity may be linear exponential, since it may depend on the expanded size of the equations.

Lemma 3.18. *If the grammar G has size $|G| = \mathcal{O}(n)$, and $\langle G', V' \rangle$ is constructed from $\langle G, \emptyset \rangle$ using $\mathcal{O}(n^k)$ many $\mathcal{O}(a^n)$-bounded grammar extension steps for some $a > 1$, then for all $\hat{a} > a$:*

$$\begin{aligned} &|G'| = \mathcal{O}(\hat{a}^{2n}) \\ &\mathit{depth}(G') = \mathcal{O}(\hat{a}^{2n}) \\ &\mathit{Vdepth}(G', V') = \mathcal{O}(\hat{a}^n) \\ &|V'| = \mathcal{O}(n^k) \end{aligned}$$

Proof. Let the sequence of grammar extension steps be $\langle G, \emptyset \rangle = \langle G_0, V_0 \rangle, \ldots, \langle G_m, V_m \rangle = \langle G', V' \rangle$, where $m = \mathcal{O}(n^k)$.

Taking the worst case of Lemmas 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, and 3.14, we can construct the recurrences:

$$\begin{aligned} &|G_{i+1}| = |G_i| + \mathit{depth}^2(G_i) + 3\,\mathit{depth}(G_i) + \mathcal{O}(a^n) \\ &\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq \mathit{Vdepth}(G_i, V_i) + 2\,\log \mathit{depth}(G_i) + \mathcal{O}(a^n) \\ &|V_{i+1}| \leq |V_i| + 1 \end{aligned}$$

These worst cases are calculated as follows. In Lemmas 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12, we have $V_{i+1} = V_i$, and the worst case is given by instantiation Lemma 3.14, that increases the size of V_i by one. For the *Vdepth*, the biggest increase is $2\,\log \mathit{depth}(G_i) + 5$, given in Lemma 3.12 or the α increment given by Lemma 3.9, where α is the bound of the extension step (in our case $\mathcal{O}(a^n)$). For the size, the worst increment is the $\mathit{depth}(G_i)(\mathit{depth}(G_i)+3)$ given by Lemma 3.12 or the $2\,\alpha$ increment given by Lemma 3.9, where again $\alpha = \mathcal{O}(a^n)$.

Since $m = \mathcal{O}(n^k)$ and $|V_i| \leq i$, we have $|V_i| = \mathcal{O}(n^k)$, for every $i = 0, \ldots, m$.

From this bound and Lemma 3.5, we have $\mathit{depth}(G_i) = \mathcal{O}(n^k)$ $\mathit{Vdepth}(G_i, V_i)$. Therefore, the recurrence for $\mathit{Vdepth}(G_i, V_i)$ may be replaced by

$$\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq \mathit{Vdepth}(G_i, V_i) + 2\,\log \mathit{Vdepth}(G_i, V_i) + \mathcal{O}(a^n).$$

A first bound for these recurrence can be computed relaxing the inequality as follows $\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq 3\,\mathit{Vdepth}(G_i, V_i) + \mathcal{O}(a^n)$ that has as solution $\mathit{Vdepth}(G_{i+1}, V_{i+1}) = 3^i \left(\mathit{Vdepth}(G_0, V_0) + \mathcal{O}(a^n)\right) = 3^i\,\mathcal{O}(a^n)$. Replacing this approximated solution in the original inequality results on

$$\begin{aligned} &\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq \mathit{Vdepth}(G_i, V_i) + 2\,\log\left(3^i\,\mathcal{O}(a^n)\right) + \mathcal{O}(a^n) \\ &= \mathit{Vdepth}(G_i, V_i) + 2\,i\,(\mathit{log}3)\mathcal{O}(n) + \mathcal{O}(a^n) \end{aligned}$$

Using the bound $i < m = \mathcal{O}(n^k)$ and the domination of the exponential function, we get the solution $\mathit{Vdepth}(G_i, V_i) = \mathcal{O}(a_1^n)$, for any $a_1 > a$. Therefore, $\mathit{depth}(G_i) = \mathcal{O}(n^k)$ $\mathit{Vdepth}(G_i, V_i) = \mathcal{O}(a_2^n)$, for any $a_2 > a_1$.

Replacing this in the recursion for $|G_i|$ we get $|G_{i+1}| = |G_i| + \mathcal{O}(a_2^{2n})$. Hence, $|G_i| = \mathcal{O}(a_3^{2n})$ for any $a_3 > a_2$.

4 Constructing the Compressed Instantiation

In this section we prove that the instantiation of the the initial equation by a size-minimal solution for bounded second-order unification problems and for

stratified context unification problems can be represented in a polynomially-sized STG. This representation is described constructively. The algorithm used for this construction is reminiscent of the ones used by [SS02,SS04] to prove the decidability of SCU and BSOU. Nevertheless, in this case, we do not compute the solution σ. Given an equation E and a minimal-size solution σ, we construct a compact representation of $\sigma(E)$. Therefore, the complexity of this algorithm is irrelevant, only the final size of the compressed representation is of importance.

4.1 Generalized Equations and Properties

In this section we operate on a *generalized equation* $(G, A \stackrel{?}{=} B)$, which consists of an STG G and two tree nonterminals A, B of G. From this generalized equation we can obtain the expanded equation $E = \{s \stackrel{?}{=} t\}$, where $val_G(A) = s$ and $val_G(B) = t$.

The construction algorithm follows the schema:

Input: an equation $E_0 = \{s \stackrel{?}{=} t\}$ and a minimal-size solution σ_0
Output: an STG G that derives $\sigma_0(s)$ and $\sigma_0(t)$
$G :=$ a STG with tree nonterminals A and B such that $s = val_G(A)$ and $t = val_G(B)$
$E := E_0$
$\sigma := \sigma_0$
while $val_G(A) \neq val_G(B)$ **do**
    Analyzing E, find an appropriate decomposition $\sigma' \circ \rho$ of σ with
        $\sigma(Z) = \sigma' \circ \rho(Z)$ for all variables $Z \in Var(E)$
    $G := G \cup \{$rules necessary to define $\rho(E)\}$
    $E := \rho(E)$
    $\sigma := \sigma'$
endwhile

Example 4.1. Consider for instance the equation $E = f(X(a), b) \stackrel{?}{=} X(f(a, y))$ and a given (in this example not-minimal) solution $\sigma_0 = [X \mapsto f([\cdot], b)^8, y \mapsto b]$. We can follow the construction algorithm. First we construct a generalized equation for E with STG G:

$$\begin{array}{ll} A ::= f(A1, A2) & B ::= X[B1] \\ A1 ::= X[A3] & B1 ::= f(A3, y) \\ A2 ::= b & \\ A3 ::= a & \end{array}$$

Then $E = val_G(A) \stackrel{?}{=} val_G(B)$ and we can see that $val_G(A) \neq val_G(B)$. We can decompose the solution σ according to the fact that we have a cycle as follows: σ as $\sigma' \circ \rho$ restricted to variables in E, where $\sigma' = [y \mapsto b]$ and $\rho = [X \mapsto f([\cdot], b)^8]$ hence, we extend G to represent $\rho(E)$ with the following rules:

$$\begin{array}{ll} C ::= C1 \cdot C1 & C1 ::= C2 \cdot C2 \\ C2 ::= C3 \cdot C3 & C3 ::= f([\cdot], b) \\ X ::= C & \end{array}$$

Now we can see again that the new E, $\rho(E)$ is not yet solved because now $val_G(A) = f(\, f([\cdot], b)^8[a], b) \neq f([\cdot], b)^8[f(a, y)] = val_G(B)$. Now $\rho = [y \mapsto b]$ and we extend G to represent the new $\rho(E)$ with the rule:

$$y \ ::= \ b$$

and now we are done because $val_G(A) = f(\, f([\cdot], b)^8[a], b) = f([\cdot], b)^8[f(a, b)] = val_G(B)$.

Keep in mind that E_0 is the initial equation that is assumed to be uncompressed. In the following we only speak of an equation E, but always mean a generalized equation w.r.t. the current STG G and solution σ. We also assume, due to Lemma 2.4, that in the case of BSOU the minimal solutions that we consider are context solutions.

To bound the number of executions of the loop of the algorithm, we define an ordering on the equations, and prove that $\rho(E)$ is strictly smaller than E w.r.t. it. This ordering is also reminiscent of the ones proposed by [SS02,SS04] to prove the termination of the decision algorithms for BSOU and for SCU. The algorithms in these paper have an at least exponential worst-case execution time. Here we prove that the length of any strictly decreasing sequence of equations in the construction process is polynomially bounded on the size of E_0. The rules used to enlarge G are grammar extension steps (see Definition 3.16). The decomposition of σ into two parts ρ and σ' is described in the rest of this section, after some introductory definitions.

Definition 4.2 (Surface and relations on variables). *We say that p is a* surface position *of t, if for every proper prefix p' of p, the term $t_{|p'}$ is not of the form $X(t')$.*

Given an equation $E = \{s \stackrel{?}{=} t\}$, we define the set $SurfEq(E)$ of surface equations as the set of equations $s|_p \stackrel{?}{=} t|_p$, where p is a surface position of s and of t.

The relation $\approx_E \subseteq Var(E) \times Var(E)$ is defined as the reflexive-symmetric-transitive closure of the relation given by: if there is an equation $X(\ldots) \stackrel{?}{=} Y(\ldots) \in SurfEq(E)$, then $X \approx_E Y$, and if there is an equation $X(\ldots) \stackrel{?}{=} y \in SurfEq(E)$, then $X \approx_E y$.

The relation $\succ_E \subseteq Var(E) \times Var(E)$ is the relation defined by: if there is an equation $X(\ldots) \stackrel{?}{=} s \in SurfEq(E)$, and Z (first-order or second-order variable) occurs at some proper surface position in s, then $X \succ_E Z$. Also: if there is an equation $x \stackrel{?}{=} s \in SurfEq(E)$, and Z occurs at some proper surface position in s, then $x \succ_E Z$. We extend this relation to $\approx_E$-equivalence classes: if $Z_1 \succ_E Z_2$ then $\overline{Z_1} \succ_E \overline{Z_2}$.

We say that $Z_1 \succeq_E Z_2$ if $Z_1 \succ_E Z_2$ or $Z_1 \approx_E Z_2$.

If $\succ_E^+$ is irreflexive, then E is said to be cycle-free*, otherwise E is called* cyclic*.*

In first-order unification all variable occurrences are at surface positions. Moreover, if $\succ_E^+$ is not irreflexive then there is an occurs-check situation and the

equation is unsolvable. In second-order unification this is not the case, $\succ_E^+$ may be not irreflexive (i.e. E cyclic) and E solvable.

Definition 4.3 (Cycle). *A cycle in an equation $E = \{s \stackrel{?}{=} t\}$ is a sequence of variables $Z_1, \ldots, Z_n$ such that $Z_i \succeq_E Z_{i+1}$, and $Z_n \succeq_E Z_1$, and the case $\succ_E$ occurs at least once. The length of the cycle is n.*

Definition 4.4 (Equation Cycle). *An* equation cycle *K of E is a sequence*

$$X_1(s_1) \stackrel{?}{=} d_1(X_2(t_1)) \,, \ldots, \; X_{h-1}(s_{h-1}) \stackrel{?}{=} d_{h-1}(X_h(t_{h-1})) \,, \; X_h(s_h) \stackrel{?}{=} d_h(X_1(t_h))$$

of surface equations, where some of the contexts d_i are not trivial, i.e. $d_i \neq [\cdot]$.

The length *of the cycle is h, and its* reduced-length *is $h - k$, where w.l.o.g. $d_1 = \ldots = d_k = [\cdot]$ is a maximal-length sequence of trivial contexts.*

The following lemma is easily derived from the definition of surface equations and the ordering $\approx_E$ and $\succ_E$.

Lemma 4.5. *If there is an equation cycle, then E is cyclic. If E is cyclic, and every surface equation $x \stackrel{?}{=} s$ for first-order variables x is of the form $x \stackrel{?}{=} x$, then there is an equation cycle.*

When the length h of the cycle is clear from the context, all indexes i greater than h are replaced by $((i-1) \mod h) + 1$. Notice that every cycle defines a sequence of classes of variables $\overline{Z_1} \succeq_E \overline{Z_2} \succeq_E \cdots \succeq_E \overline{Z_h} \succeq_E \overline{Z_1}$, where for some $i \in \{1, \ldots, h\}$, $\overline{Z_i} \succ_E \overline{Z_{i+1}}$.

Definition 4.6 (Depth of a variable). *If there are no cycles in E, then for a variable Z, we define $depth_E(Z)$ as the maximal length d of a chain $Z = Z_1 \succ_E Z_2 \succ_E Z_2 \succ_E \ldots \succ_E Z_d$.*

Lemma 4.7. *In a solvable equation E there is no cycle where all variables are first-order.*

The shortest equation cycle in an equation E is not longer than $|Var(E)|$.

Definition 4.8 (Ordering on equations). *Given an equation E, the measure $\mu(E)$ is a lexicographic combination $\langle \mu_1(E), \mu_2(E) \rangle$ of the following components:*

1. *$\mu_1(E) = |Var(E)|$ is the number of variables occurring in E.*
2. *$\mu_2(E) = (0, \chi(E))$ if E is cyclic, and $(1, \nu(E))$ if E is acyclic.*

where $\chi(E)$ for cyclic equations has the lexicographically ordered components:

1. *$\chi_1(E)$ is the shortest length of the cycles of E,*
2. *$\chi_2(E)$ is the shortest reduced-length of the length-minimal cycles of E.*

and where $\nu(E)$ for non-cyclic E has the lexicographically ordered components:

1. *ν_1 is the sum of all $depth_E(X)$ for all second-order variables X that occur on the surface of E.*

2. $\nu_2 := \big|\{Z \mid Z \text{ is a } \succ_E\text{-maximal variable}\}\big| - \big|\{\overline{Z} \mid \overline{Z} \text{ is a } \succ_E\text{-maximal equivalence class}\}\big|$, *i.e. the number of maximal variables minus the number of maximal equivalence classes.*

Lemma 4.9. *Any strictly μ-decreasing sequence of equations starting with E terminates in at most $\mathcal{O}(|\,Var(E)|^4)$ steps.*

Proof. The upper bound is obvious from the components, since μ_1 has only $|E|$ possibilities, ν permits $|E|^3$ possibilities, which dominates χ which permits only $|E|^2$ possibilities.

4.2 Construction of the Grammar

We want to join the algorithms for BSOU and for SCU as much as possible, since the algorithms are very similar. There is an obvious difference for the case where all surface equations are of the form $X(\ldots) \stackrel{?}{=} Y(\ldots)$, since then in BSOU there is an obvious unifier, whereas for SCU the algorithm has to look further for partial instances and is far from being finished. We assume $\sigma(X)$ to be a context for all second-order variables X, since for BSOU we assume by Lemma 2.4 that the minimal solution σ is also a context substitution.

The following construction methods (see Section 3) are already described and the corresponding estimations for the grammar-size increases are already given. These constructions correspond to the *grammar extension steps* of Definition 3.16. If G defines the terms t, u (the contexts c, d), then the following can be constructed: a subterm of t, a prefix context of t, a suffix of c, a prefix of c, and exponentiation of the context c, and a concatenation $c \cdot d$ of contexts c, d. Also the following instantiations are constructible: $[x \mapsto u]t$, $[x \mapsto u]c$, $[X \mapsto c]t$, and $[X \mapsto c]s$.
The plan is to present the following cases in order:

- First, we give some basic rules for instantiations in trivial cases and instantiations of first-order variables.
- Then, the case where there are cycles for BSOU as well as for SCU.
- Finally, the case when there are no cycles. Here a large part of the construction is common for BSOU and SCU. Only the case that all surface variables are in $\approx_E$-equivalence classes that are maximal as well as minimal w.r.t. $\succeq_E$ requires a different treatment.

We will count different types of STG-extensions according to Theorem 3.17. For this estimations we assume that the signature Σ is fixed and hence the maximal arity of a function symbol is $O(1)$. In the following σ is a minimal context solution of the current equation $E = \{s \stackrel{?}{=} t\}$. We also assume that every instantiation step partially instantiates the given problem by ρ, where ρ is built accordingly to a decomposition of σ as $\sigma' \circ \rho$ with $\sigma(Z) = \sigma' \circ \rho(Z)$ for all $Z \in Var(E)$, depending on the substitution σ and the possible cases of the construction. The substitution σ' will have the components needed for the freshly introduced variables by ρ. When we argue that the measure strictly decreases, we only provide arguments for the extreme case and omit the trivial arguments.

Case 1 Trivial Cases

There are some trivial cases that we solve by means of the following rule which we refer to as TRIVIAL-INSTANTIATION:

Case 4.9.1 There is a second-order variable $X \in Var(E)$ with $\sigma(X) = [\cdot]$. Take $\rho = [X \mapsto [\cdot]]$, and enlarge the grammar with the rule $X ::= [\cdot]$ according to Lemma 3.14. The new grammar defines $\rho(E)$ and is obtained by a grammar extension step of the old grammar. The new equation $\rho(E)$ is strictly smaller than the old one E, because it contains less variables, i.e. $\mu_1(\rho(E)) < \mu_1(E)$.

Case 4.9.2 From now on assume that the Case 4.9.1 is not applicable, i.e. $\sigma(X) \neq [\cdot]$ for any second-order variable $X \in Var(E)$.
Let p be a surface position in s and t such that p is a position of a first-order variable in s or t, w.l.o.g. let $s_{|p}$ be the first-order variable, say $x \in Var(E)$. Let also $s_{|p} \neq t_{|p}$. Notice that, since $s \stackrel{?}{=} t$ has σ as solution, the variable x does not occurs in $t_{|p}$. Take $\rho = [x \mapsto t_{|p}]$. According to the subterm construction of Lemma 3.11, extend G with the rules necessary to define $t_{|p}$ by a new non-terminal T and, according to Lemma 3.14, with the rule $x ::= T$. Therefore the new grammar defines $\rho(E)$ and it is obtained with two grammar extension steps. Like in the previous case, $\rho(E)$ is strictly smaller than E because contains less variables.

Case 2 There Are Cycles

There is an equation cycle K of the form

$$X_1(s_1) \stackrel{?}{=} d_1(X_2(t_1)), \ldots, X_{h-1}(s_{h-1}) \stackrel{?}{=} d_{h-1}(X_h(t_{h-1})), X_h(s_h) \stackrel{?}{=} d_h(X_1(t_h))$$

where $d_h \neq [\cdot]$ is not trivial. We can assume that there are no TRIVIAL INSTANTIATIONS possible, hence there are no first-order variables in the cycle. We also assume that this is a *minimal equation cycle*: it is of minimal length, and of minimal reduced-length among length-minimal equational cycles, i.e. its length is $\chi_1(E)$ and its reduced-length is $\chi_2(E)$.
We define a single construction step, which we refer to as CONSTRUCT-FOR-CYCLIC. There are two cases:

Case 4.9.1 There are two or more nontrivial contexts $d_{k+1} \neq [\cdot]$ and $d_h \neq [\cdot]$, where $k+1 \neq h$. We focus on a maximal-length sequence of trivial contexts: $X_1(s_1) \stackrel{?}{=} X_2(t_1), \ldots, X_k(s_k) \stackrel{?}{=} X_{k+1}(t_k), X_{k+1}(s_{k+1}) \stackrel{?}{=} d_{k+1}[X_{k+2}(t_{k+1})]$. Let q be the maximal position satisfying the following conditions:

1. q is a prefix of $mp(d_{k+1})$, and
2. q is a prefix of $mp(\sigma(X_i))$, for all $i = 1, \ldots, k+1$.

Let c be the prefix subcontext of d_{k+1} with hole at position q. There are several subcases:

Case 4.9.1.1 If $q = mp(d_{k+1})$, then take $\rho = [X_1 \mapsto c[X'_1], \ldots, X_{k+1} \mapsto c[X'_{k+1}]]$, where $X'_1, \ldots, X'_k$ are fresh second-order variables. The new equation $\rho(E)$ will contain a cycle of the same length as the one we analyze, but the list of trivial contexts will be longer. Therefore, $\rho(E)$ will be strictly smaller than E because $\chi_2(\rho(E)) < \chi_2(E)$.

Case 4.9.1.2 If q is a position of the hole in some context $\sigma(X_j)$, for some $1 \leq j \leq k+1$, then take

$$\rho = \left[X_1 \mapsto c[X'_1], \ldots, X_j \mapsto c, \ldots, X_{k+1} \mapsto c[X'_{k+1}]\right]$$

Since we remove $k+1$ variables X_i's, and we only add k variables X'_i's, $\mu_1(\rho(E))$ will be strictly smaller that $\mu_1(E)$, hence $\rho(E)$ smaller that E w.r.t. μ.

Case 4.9.1.3 [The derailing case] Otherwise, for $i = 1, \ldots, k+1$, let q_i be the sequence satisfying $|q_i| = 1$ and $q \cdot q_i$ is a prefix of $mp(\sigma(X_i))$. Note that the contexts $\sigma(X_i)_{|q}$ have the same function symbol f as head, which is also the head of the suffix context $d_{k+1}|_q$.
Take the substitution

$$\rho' = \left[X_1 \mapsto c[f(y_{1,1}, \ldots, X'_1[\cdot], \ldots y_{1,m})], \ldots, X_{k+1} \mapsto c[f(y_{k+1,1}, \ldots, X'_{k+1}[\cdot], \ldots y_{k+1,m})]\right]$$

where $y_{i,j}$ are fresh first-order variables, and X'_i are fresh second-order variables. Applying the substitution ρ' to the original equations we obtain, among others, the following surface equations:

$$\begin{gathered} f(y_{1,1}, \ldots, X'_1(s_1), \ldots y_{1,m}) \stackrel{?}{=} f(y_{2,1}, \ldots, X'_2(t_1), \ldots y_{2,m}) \\ \cdots \\ f(y_{k,1}, \ldots, X'_k(s_k), \ldots y_{k,m}) \stackrel{?}{=} f(y_{k+1,1}, \ldots, X'_{k+1}(t_k), \ldots y_{k+1,m}) \\ f(y_{k+1,1}, \ldots, X'_{k+1}(s_{k+1}), \ldots y_{k+1,m}) \stackrel{?}{=} d_{k+1}|_q\,[X_{k+2}(t_{k+1})] \end{gathered}$$

Notice that we may introduce more variables than we remove, therefore $\rho'(E)$ can be bigger than E w.r.t. the ordering μ. Fortunately, for $i \in \{1 \ldots k\}$, we can construct a substitution ρ''_i that, for $j \in \{1 \ldots m\}$, instantiates the first-order variable $y_{i,j}$ by either $y_{i+1,j}$, when $q_{i+1} \neq j$, or by $X'_{i+1}(t_i)$, when $q_{i+1} = j$. We can also construct a substitution ρ_{k+1} that, for $j \in \{1 \ldots m\}$, instantiates the first-order variable $y_{k+1,j}$ by $d_{k+1}|_{q \cdot j}\,[X_{k+2}(t_{k+1})]$. The substitution $\rho = \rho''_{k+1} \circ \cdots \rho''_1 \circ \rho'$ restricted to the domain $\{X_1, \ldots, X_{k+1}\}$ does not introduce fresh first-order variables.
Notice that $\rho(X_i) = c[f(u_{i,1}, \ldots, X'_1[\cdot], \ldots u_{i,m})]$, where terms $u_{i,j}$ are of the form $X'_{l+1}(t_l)$, for some $l > i$, or $d_{k+1}|_{q \cdot j}\,[X_{k+2}(t_{k+1})]$. The c, u's, t's and $d_{k+1}|_{q \cdot j}\,[X_{k+2}(t_{k+1})]$ are subterms or subcontexts of E. Therefore, the grammar can be extended to define $\rho(E)$ with $\mathcal{O}(k)$ grammar extension steps.
We can see also that, since not all q_i's are equal, ρ applied to the original cycle gives a smaller cycle. Hence, $\chi_1(\rho(E)) < \chi_1(E)$.

Case 4.9.2 The Case 4.9.1 does not apply, i.e. the equation cycle is as follows: $X_1(s_1) \stackrel{?}{=} X_2(t_1), \ldots, X_{h-1}(s_{h-1}) \stackrel{?}{=} X_h(t_{h-1}), X_h(s_h) \stackrel{?}{=} d_h[X_1(t_h)]$. Let q be the maximal position satisfying the following:

1. q is a prefix of $mp(d_h^{\mathrm{eop}(\sigma)+1})$, and
2. q is a prefix of $mp(\sigma(X_i))$, for all $i = 1, \ldots, h$.

Let c be the subcontext of $d_k^{\mathrm{eop}(\sigma)+1}$ with hole at position q. There are several subcases:

Case 4.9.2.1 The position q is the main path of some $\sigma(X_j)$, for some $j \in \{1, \ldots, h\}$. Take

$$\rho = \left[X_1 \mapsto c[X_1'], \ldots, X_j \mapsto c, \ldots, X_h \mapsto c[X_h']\right]$$

The new equation $\rho(E)$ contains a variable less than E, the one corresponding to X_j.

Case 4.9.2.2 [The derailing case] This case is equal to Case 4.9.1.3, since we do not assume that $k < h$. The only difference is that now c is not a subcontext of E, but d_h raised to some exponent bounded by $\mathrm{eop}(\sigma)$, and composed with some prefix of d_h.

Like in the other derailing case, the new equation contain a shorter cycle.

Case 3 There Are No Cycles

For the construction, we assume that the trivial construction steps are already done. We define a single construction step, which we refer to as Construction-for-NonCyclic.

Case 4.9.1 Let W be some $\succeq_E$-maximal $\approx_E$-equivalence class in $Var_G(E)$, which is in addition not $\succeq_E$-minimal. Note that W consists only of second-order variables, since no trivial steps are applicable, and there is some surface equation of the form $X(\ldots) \stackrel{?}{=} r$, where r has a function symbol as head.

Let q be the maximal position such that the following holds:

1. q is a prefix of all main paths of $\sigma(X)$, for all $X \in W$, and
2. q is a surface position in r.

There are some subcases:

Case 4.9.1.1 Position q is the main path of some context, say $\sigma(X_1)$, where $X_1 \in W$. Let c be the prefix of r with main path q. Take

$$\rho = \left[X_1 \mapsto c, X_2 \mapsto c[X_2'], \ldots, X_n \mapsto c[X_n']\right]$$

where $W = \{X_1, \ldots, X_n\}$ and $X_2', \ldots, X_n'$ are fresh second-order variables.

The new equation $\rho(E)$ has a variable less, therefore it is smaller than E.

Case 4.9.1.2 Assume Case 4.9.1.1 does not apply. If $r_{|q}$ is of the form x or $X(u)$, let c be the context prefix of r with main path q. Take

$$\rho = \left[X_1 \mapsto c[X_1'], \ldots, X_n \mapsto c[X_n']\right]$$

where $W = \{X_1, \ldots, X_n\}$ and $X_1', \ldots, X_n'$ are fresh second-order variables.

In this case the number of variables does not decrease, but $\rho(E)$ contains a maximal class $\{X_1', \ldots, X_n', X\}$ or $\{X_1', \ldots, X_n', x\}$ bigger than the maximal class W of E, which has a smaller sum of all the numbers $depth_E(X_i')$. Therefore, $\nu_1(\rho(E)) < \nu_1(E)$ and $\rho(E)$ is smaller than E w.r.t. μ.

Case 4.9.1.3 [The derailing case] Cases 4.9.1.1 and 4.9.1.2 do not apply. This is the situation where the contexts $\sigma(X)$ go into different directions. Let $W = \{X_1, \ldots, X_n\}$, for all $i = 1, \ldots, n$, let q_i be a position of length 1, such that $q \cdot q_i$ is a prefix of the main path of $\sigma(X_i)$, let c be the prefix context of r with main path q, and let f the function symbol (of non-zero arity k) at position q of r. For simplicity, assume that one of the surface equations is $X_n(s_n) \stackrel{?}{=} r$. Like in the other derailing cases, take

$$\rho' = \left[X_1 \mapsto c[f(y_{1,1}, \ldots, X'_1[\cdot], \ldots y_{1,m})], \ldots, X_n \mapsto c[f(y_{n,1}, \ldots, X'_n[\cdot], \ldots y_{n,m})]\right]$$

where $y_{i,j}$ are fresh first-order variables, the variables $X'_i[\cdot]$ occur at argument index q_i, and X'_i are fresh second-order variables. Applying the substitution ρ' to the original equations we obtain, among others, the following surface equation:

$$f(y_{n,1}, \ldots, X'_n(s_n), \ldots, y_{n,m}) \stackrel{?}{=} r|_q$$

Now we can construct the instantiation ρ''_n, such that $y_{n,j}$ is replaced by $r|_{q \cdot j}$ for $j \neq q_n$. For $i \in \{1 \ldots n-1\}$, we can inductively construct (perhaps by rearranging the indices if necessary), substitutions ρ''_i that, for $j \in \{1 \ldots m\}$, instantiates the first-order variable $y_{i,j}$ by either $y_{i+1,j}$, when $q_{i+1} \neq j$, or by $X'_{j'}(t_i)$ for some j', when $q_{i+1} = j$. Finally, take $\rho = \rho''_n \circ \cdots \rho''_1 \circ \rho'$ restricted to the domain $\{X_1, \ldots, X_n\}$. After instantiating E with ρ, the maximal class W is split into at least two nonempty new maximal classes. This produces a decrement in the value of ν_2.

Case 4.9.2 Assume that the case 4.9.1 is not applicable. The remaining case is that all second-order variables $\{X_1, \ldots, X_n\}$ that occur at surface positions are in $\succeq_E$-maximal $\approx_E$-equivalence classes of E, that in addition are $\succeq_E$-minimal. Now we have to describe the construction for BSOU and SCU separately.

Case for BSOU As we prove below in Lemma 4.11, this case is very special, due to minimality of the solution σ. Only the following can occur: for $i = 1, \ldots, n$, $\sigma(X_i) = f([\cdot])$, where f is some unary function symbol of the signature. Take $\rho = \sigma$, and the algorithm finishes.

Case for SCU Let $W = \{X_1, \ldots, X_n\}$ be one of the $\approx_E$-equivalence classes of E, that are $\succeq_E$-maximal as well as $\succeq_E$-minimal. Note that W consists only of second-order variables. For $i = 1, \ldots, n$, let q_i be a position of length 1 that is a prefix of the main path of $\sigma(X_i)$. Stratifiedness implies that all occurrences of variables of W are on surface positions. Minimality of σ_0 implies that $|\{q_i \mid i = 1, \ldots, n\}| \geq 2$. Since $\sigma(X_i) \neq [\cdot]$, there is a function symbol $f \in \Sigma$, which is the head of all $\sigma(X_i)$. Take

$$\rho' = \left[X_1 \mapsto f(y_{1,1}, \ldots, X'_1[\cdot], \ldots, y_{1,m}), \ldots, X_n \mapsto f(y_{n,1}, \ldots, X'_n[\cdot], \ldots, y_{n,m})\right]$$

where the X_i's and $y_{i,j}$'s are fresh second and first-order variables, respectively. Applying the same argument as for the derailing cases, construct a substitution ρ'' that instantiates all $y_{i,j}$'s by subterms of E. Finally, take $\rho = \rho'' \circ \rho'$, restricted to the variables X_i's. Since $|\{q_i \mid i = 1, \ldots, n\}| \geq 2$, the equivalence class W of E will be split into at least two nonempty equivalence classes in $\rho(E)$. Therefore $\rho(E)$ is strictly smaller than E.

4.3 Properties of the Construction and Bounds for G

The following lemma has been proved in the previous analysis by cases and establishes the correctness of the grammar construction.

Lemma 4.10. *Let $E = \{s \stackrel{?}{=} t\}$ be an equation, G be an STG defining s and t, and σ be a minimal solution of the SCU E [a minimal and context solution of the BSOU E]. The substitution ρ obtained by the rules* Trivial-instantiation, Construction-for-Cyclic, Construction-for-NonCyclic *satisfies the following properties:*

1. *There exists a substitution σ' such that*
   (a) *$\sigma(Z) = \sigma' \circ \rho(Z)$, for all variables $Z \in Var(E)$,*
   (b) *σ' with domain restricted to $Var(\rho(E))$ is a SCU minimal solution [a BSOU minimal and context solution] of $\rho(E)$, and*
   (c) *we have $\mathrm{eop}(\sigma') \leq \mathrm{eop}(\sigma)$.*
2. *$\rho(E)$ is strictly smaller than E, w.r.t. the ordering μ,*
3. *if E is stratified, then $\rho(E)$ is also stratified, and*
4. *an STG grammar G' can be constructed defining $\rho(E) = \{\rho(s) \stackrel{?}{=} \rho(t)\}$, from G with $\mathcal{O}(Var(E))$ grammar extension steps.*

Lemma 4.11. *In the construction of the STG for a solution in the non-cyclic case, for BSOU-problems, in Case 4.9.2 only $\sigma(X_i) = f([\cdot])$ for a unary $f \in \Sigma$ is possible.*

Proof. This is the case where all second-order variables $\{X_1, \ldots, X_n\}$ that occur at surface positions are in a $\succeq_E$-maximal $\approx_E$-equivalence class in $Var_G(E)$, that is in addition $\succeq_E$-minimal. Since we are in the case of BSOU, it is possible in this case to construct a small solution as follows: For a signature constant a, define the substitution $\rho := \{X_i \mapsto \lambda\_.a \mid i = 1, \ldots, n\}$. Then, together with the already computed instantiation σ, the solution $\rho \circ \sigma$, restricted to the variable in $Var(E_0)$ is a unifier of the initial equation E_0 that is in addition size-minimal, which follows from the soundness part of Lemma 4.10. Since we have assumed that our size-minimal solution is a context solution, and $\sigma(X_i) = [\cdot]$ is excluded because the trivial cases are not possible, the size of $\sigma(X_i))$ must be 1, and hence, for every i, we have $\sigma(X_i) = f([\cdot])$, for some $f \in \Sigma_1$.

We obtain the following upper bound on the size of an STG that represents a minimal solution of a BSOU or SCU-problem E.

Theorem 4.12. *Given a BSOU-problem (SCU-problem, respectively) $E = \{s \stackrel{?}{=} t\}$, and a minimal solution σ of E, the terms $\sigma(s)$ and $\sigma(t)$ can be represented with an STG G, such that $|G|$ is of size $\mathcal{O}(|E|^{27})$. Moreover, the STG G is a grammar extension of the STG defining E.*

Proof. Lemma 2.4 shows that w.l.o.g. we can assume that a minimal solution is a context solution, hence we can use our construction steps. Lemma 4.9 shows that the number of construction steps is of order $\mathcal{O}(|E|^4)$. Lemma 4.10 implies now that the number of required instantiations is of order $\mathcal{O}(|E|^5)$, the number of construction steps between two instantiations is of order $O(1)$, and the maximal exponent of periodicity obtained during the construction is bounded by $\mathcal{O}(2^{|E|})$. Now Theorem 3.17 shows that the size of G is of order $\mathcal{O}(|E|^{5*5+2}) = \mathcal{O}(|E|^{27})$.

5 Results

Main Theorem 5.1 *Stratified Context Unification and Bounded Second-Order Unification are NP-complete.*

Proof. Lemma 2.4 allows us to assume w.l.o.g. that a BSOU minimal solution is a context solution. We can implement a non-deterministic algorithm that decides solvability of the SCU (BSOU, respectively) problem $E = \{s \stackrel{?}{=} t\}$, in polynomial time, as follows: Given $s \stackrel{?}{=} t$, construct an STG G that from A and B generates $s = val_G(A)$ and $t = val_G(B)$. We guess a new STG G' as an extension of G of size bounded by $\mathcal{O}((|s| + |t|)^k)$, with $k = 27$ according to Theorem 4.12. If $val_{G'}(A) = val_{G'}(B)$ holds, then the algorithms stops with success and says "unifiable".

By Lemma 3.3, the most expensive step of the algorithm, the test $val_{G'}(A) = val_{G'}(B)$, can be done in time $\mathcal{O}(|G'|^3)$ therefore, the algorithm has time complexity $\mathcal{O}(n^{3*27})$ on the size of the input.

By Lemma 3.15, for any extension G' constructed from the original G, there exists a substitution σ such that $val_{G'}(A) = \sigma(val_G(A)) = \sigma(s)$, and $val_{G'}(B) = \sigma(val_G(B)) = \sigma(t)$. If $val_{G'}(A) = val_{G'}(B)$, then σ is a unifier of $s \stackrel{?}{=} t$. Therefore, the algorithm is sound.

By Theorem 4.12 (where we again use Lemma 2.4) for any minimal solution σ of a SCU (BSOU, respectively) problem $s \stackrel{?}{=} t$, the terms $\sigma(s)$ and $\sigma(t)$ can be represented with a polynomially-sized STG G', that in addition is a grammar extension of the original STG G. When the algorithm guesses this grammar, it accepts. Therefore, the algorithm is complete.

NP-hardness for both decision problems is already known [SSS98,SS04].

One-step rewrite constraints were introduced by [CCD93]. In [NTT00] it is proved that SCU and one-step rewrite constraints are equivalent problems. From this result and Theorem 5.1 we can conclude the following result.

Corollary 5.2. *Solvability of one-step rewrite constraints is NP-complete.*

It is not clear whether unifiability of generalized stratified context-unification problems (G, E) is in NP, since the usual encoding does not produce a stratified unification problem. However, the following is easy:

Corollary 5.3. *Unifiability of generalized bounded second-order unification problems is NP-complete.*

Proof. The STGs G and the generalized input equation E can be encoded into usual ones in linear time as follows: Every nonterminal is turned into a variable, where term nonterminals are turned into first-order variables and context nonterminals into second-order variables. The grammar has to be translated into equations in the usual way, where the rules for context nonterminals have to be translated as two equations. E.g. $C_1 ::= C_2 \cdot C_3$ is translated as the two equations $C_1^X(s_1) \stackrel{?}{=} C_2^X(C_3^X(s_1))$, $C_1^X(s_2) \stackrel{?}{=} C_2^X(C_3^X(s_2))$, where s_1, s_2 are two small different ground terms and C_i^X are fresh second-order variables. This translation is sound and complete and can be done in linear time. Then, we apply Theorem 5.1.

For stratified context-unification problems the complexity bound is a bit higher:

Corollary 5.4. *Unifiability of generalized stratified context unification problems is in NEXPTIME.*

Proof. Let E be the generalized SCU-problem compressed using the STG G. The exponent of periodicity is of order $\mathcal{O}(2^{(a^{|G|})})$, for some $a > 1$, since it must be determined using the size of the SCU-problem after expanding it using the rules from G. Then, we use the same construction as for plain SCU-problems. Note that the number of variables is $O(|G|)$, and hence the number of construction steps is the same as for plain SCU-problems. Applying Lemma 3.18 and using the same arguments as in Main Theorem 5.1, we obtain the upper complexity bound NEXPTIME.

6 Conclusion

We prove that bounded second-order unification and stratified context unification are in NP, exploiting compression of instantiations using singleton tree grammars and finally making a non-deterministic guess of a polynomial-sized grammar. We also compute upper bounds for the grammar that has to be guessed as $O(n^{27})$. Presumably, the bound can be improved by a finer analysis of the grammar extensions and instantiations.

Acknowledgments

This research has been partially supported by the research projects Mulog-2 (TIN2007-68005-C04-01) and SuRoS (TIN2008-04547) funded by the CICyT.

References

[BLM05] G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML documents. In *Proc. of the 10th Int. Symp. on Database Programming Languages, DBPL'05*, volume 3774 of *Lecture Notes in Computer Science*, pages 199–216, 2005.

[CCD93] Anne-Cécile Caron, Jean-Luc Coquidé, and Max Dauchet. Encompassment properties and automata with constraints. In *Proc. of the 5th Int. Conf. on Rewriting Techniques and Applications, RTA'93*, volume 690 of *Lecture Notes in Computer Science*, pages 328–342, 1993.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007.

[Dow01] G. Dowek. Higher-order unification and matching. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 16, pages 1009–1062. Elsevier Science, 2001.

[ES77] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. I. *Journal of Computer and System Sciences*, 15(3):328–353, 1977.

[ES78] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. II. *Journal of Computer and System Sciences*, 16(1):67–99, 1978.

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

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

[GGSS08] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Context matching for compressed terms. In *Proc. of the 23rd Annual IEEE Symp. on Logic in Computer Science, LICS'08*, pages 93–102, 2008.

[GGSS09] A. Gascón G. Godoy, and M. Schmidt-Schauß. Unification with singleton tree grammars. In *Proc. of the 20th Int. Conf. on Rewriting Techniques and Applications, RTA'09*, volume 5595 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2009.

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

[KLV07] Temur Kutsia, Jordi Levy, and Mateu Villaret. Sequence unification through currying. In *Proc. of the 18th Int. Conf. on Rewriting Techniques and Applications, RTA'07*, volume 4533 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2007.

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

[Lev96] J. Levy. Linear second order unification. In *Proc. of the 7th Int. Conf. on Rewriting Techniques and Applications, RTA'96*, volume 1103 of *Lecture Notes in Computer Science*, pages 332–346, 1996.

[Lif06] Y. Lifshits. Solving classical string problems on compressed texts. *CoRR*, abs/cs/0604058, 2006.

[Lif07] Y. Lifshits. Processing compressed texts: A tractability border. In *Proc. of the 18th Annual Symp. on Combinatorial Pattern Matching, CPM'07*, pages 228–240, 2007.

[LMSS09] Markus Lohrey, Sebastian Maneth, and Manfred Schmidt-Schauß. Parameter reduction in grammar-compressed trees. In *Proc. of the 12th Int. Conf. on Foundations of Software Science and Computational Structures, FoSSaCS'09*, volume 5504 of *Lecture Notes in Computer Science*, pages 212–226, 2009.

[LNV05] J. Levy, J. Niehren, and M. Villaret. Well-nested context unification. In *Proc. of the 20th Int. Conf. on Automated Deduction, CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 149–163, 2005.

[Loh06] M. Lohrey. Word problems and membership problems on compressed words. *SIAM Journal on Computing*, 35(5):1210–1240, 2006.

[LSSV04] J. Levy, M. Schmidt-Schauß, and M. Villaret. Monadic second-order unification is NP-complete. In *Proc. of the 15th Int. Conf. on Rewriting Techniques and Applications, RTA'04*, volume 3091 of *Lecture Notes in Computer Science*, pages 55–69, 2004.

[LSSV06a] Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. Bounded second-order unification is NP-complete. In *Proc. of the 17th Int. Conf. on Rewriting Techniques and Applications, RTA'06*, volume 4098 of *Lecture Notes in Computer Science*, pages 400–414, 2006.

[LSSV06b] Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. Stratified context unification is NP-complete. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning, IJCAR'06*, volume 4130 of *Lecture Notes in Computer Science*, pages 82–96, 2006.

[LSSV08] Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. The complexity of monadic second-order unification. *SIAM Journal on Computing*, 38(3):1113–1140, 2008.

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

[LV02] J. Levy and M. Villaret. Currying second-order unification problems. In *Proc. of the 13th Int. Conf. on Rewriting Techniques and Applications, RTA'02*, volume 2378 of *Lecture Notes in Computer Science*, pages 326–339, 2002.

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

[NTT00] J. Niehren, S. Tison, and R. Treinen. On rewrite constraints and context unification. *Information Processing Letters*, 74:35–40, 2000.

[Pla94] W. Plandowski. Testing equivalence of morphisms in context-free languages. In *Proc. of the 2nd Annual European Symp. on Algorithms, ESA'94*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470, 1994.

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

[PR99] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever*, pages 262–272. Springer, 1999.

[Rou69] William C. Rounds. Context-free grammars on trees. In *Proc. of the 1st ACM Symposium on Theory of Computing, STOC'69*, pages 143–148, 1969.

[SS02] M. Schmidt-Schauß. A decision algorithm for stratified context unification. *Journal of Logic and Computation*, 12(6):929–953, 2002.

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

[SS05] M. Schmidt-Schauß. Polynomial equality testing for terms with shared substructures. Frank report 21, Institut für Informatik. FB Informatik und Mathematik. J. W. Goethe-Universität Frankfurt am Main, 2005.

[SSS98] M. Schmidt-Schauß and K. U. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *Proc. of the 9th Int. Conf. on*

Rewriting Techniques and Applications, RTA'98, volume 1379 of *Lecture Notes in Computer Science*, pages 61–75, 1998.

[SSS02] Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. *Journal of Symbolic Computation*, 33(1):77–122, 2002.