Context Matching for Compressed Terms

Adria Gascon and Guillem Godoy
LSI Department, Universitat Politècnica de Catalunya
Jordi Girona, 1-3 08034 Barcelona, Spain,
adriagascon@gmail.com, ggodoy@lsi.upc.edu

Manfred Schmidt-Schauss
Inst. Informatik, J.W.Goethe-Universität,
D-60054 Frankfurt, Germany,
schauss@ki.informatik.uni-frankfurt.de

Abstract

This paper is an investigation of the matching problem for term equations $s = t$ where s contains context variables, and both terms s and t are given using some kind of compressed representation. In this setting, term representation with dags, but also with the more general formalism of singleton tree grammars, are considered.

The main result is a polynomial time algorithm for context matching with dags, when the number of different context variables is fixed for the problem. NP-completeness is obtained when the terms are represented using singleton tree grammars. The special cases of first-order matching and also unification with STGs are shown to be decidable in PTIME.

1 Introduction

Solving equations is a fundamental part of any mathematically founded science. In general, solving an equation $s \doteq t$ consists of finding a substitution σ for variables such that $\sigma(s) = \sigma(t)$. The range for the variables, the kind of expressions s and t, and their semantics, as well as the semantics of $=$, depend on the context.

The first-order term unification problem is a particular case of solving equations where the expressions s and t are terms with leaf variables ranging over terms, all function symbols are non-interpreted, and $=$ is interpreted as syntactic equality. First-order term unification and its variants are basic ingredients in many areas of logic in computer science. A particular case of term unification appears when one of the sides of the equation $s \doteq t$, say t, contains no variables. This is the term matching problem, and it is again a fundamental ingredient of many areas like functional and logic programming, rewriting, automated deduction, deductive databases, artificial intelligence, pattern matching, compilation, etc.

The above mentioned particular problems are efficiently solvable, where the most efficient algorithm uses dags to represent the solutions, but their expressivity is often insufficient. For this reason, several variants and generalizations of the first-order term matching and unification problems have been studied. Incorporating more complex interpretation of the function symbols and equality predicate under equational theories has been widely considered (see [**?**, 1]). Another direction of extending expressivity is to reconsider complexity issues for the original problem or its variants by assuming that the input terms are given in some compressed representation. This is important from a practical point of view, since much of the applications involving terms and matching use some kind of internal succinct representation for terms. Finally, extensions like allowing other kinds of variables related to terms have also been explored. This is the case of context variables, i.e. variables which can be substituted by contexts, which are trees with a single hole.

For example, let $t = f(g(a, b), g(a, h(b)))$ in the match-equation $F(a) = t$, where F is a context variable. This has the solutions $F \mapsto f(g([.], b), g(a, h(b)))$ (where $[\cdot]$ means the hole) and $F \mapsto f(g(a, b), g([.], h(b)))$; the equation $f(F(b), F(h(b)) = t$ has the solution $F \mapsto g(a, [.])$, whereas $f(F(b), F(b)) = t$ has no solution. Context matching is known to be NP-complete, but there are several subcases that can be solved efficiently [20]. If terms are large but have lots of common subterms, like $t_1 = f(a, b)$, $t_2 = f(t_1, t_1), \ldots, t_n = f(t_{n-1}, t_{n-1})$, then the context matching equation $X(a) = t_n$ will require exponential space using the term representation to represent t_n, whereas a dag representation requires linear space. This motivates to investigate context matching with dags and also in connection with more general compression techniques for terms.

An interesting application of context matching is the search within tree structures and the corresponding extraction of information. For example, the match equation $F(s) = t$ where t is ground and s has no occurrences of F corresponds to the question whether there is a subtree of t that is matched by s. This can easily be combined as conjunctive search $F_1(s_1) = t; \ldots; F_n(s_n) = t$, where F_i are all different and do not occur elsewhere. This match equations correspond to the search question whether there are

subterms r_i of t that can be matched by s_i for $i = 1, \ldots, n$, where variables within s_i must have a common instance in t. Using dags, the multiple occurrences of t can be represented as the same node. The instantiation of a context variable by a match is a context, i.e. tree with a hole. Thus multiple occurrences of the same context variable correspond to the question whether there are occurrences of the same subtree, but up to one position. This interpretation has applications in computational linguistics [14]. Questions that ask for subtrees that are equal up to several positions can be encoded using context variables. Applications of context matching are in querying XML-data bases: see [2] for the XPATH-standard, [20] for investigating context matching, and [8] for analyzing conjunctive query mechanisms over trees.

Term compression generalizes string compression. A general string compression mechanism is the grammar-based compression formalism introduced by Plandowski [15, 16, 17], where it is shown that the word problem for compressed strings is polynomial in the size of the (restricted) context free grammar, which are also known as *straight-line programs*. For some complexity results for the word problem under compression see [**?**, **?**]. This grammar formalism is extended also to trees (see [3, 18]) and comprises the dag-representation of terms and also compressing context composition, such that e.g. C^n for a context C and a fixed number n can be represented in linear space. Term compression was applied to the complexity analysis of unification algorithms in [13, 12], where the grammars are called singleton tree grammars (STG), and also to compressing XML-data bases [3].

In this paper we consider the context matching problem when the input is compressed using dags, but also using the more general representation of singleton tree grammars. Since the context matching problem itself with uncompressed terms is NP-complete, it seems to be difficult to obtain interesting results for the compressed case. But we consider the situation where the number of different context variables is small, or fixed for the problem. This kind of restriction has already been considered for context unification restricted to two context variables [19], and also proved useful in the context of program verification with procedure calls [9], where context unification for monadic signatures and a single context variable allows the automatic generation of invariants.

Our main result is a polynomial time algorithm for context matching when the input terms are represented with dags. This is presented in Section 5 with a nondeterministic polynomial time algorithm which permits only a polynomial number of possibilities. It is based on the observation that during the solution process, there is always either a context variable with an arbitrary solution, or a context variable, where a suffix of the path to the hole can be inferred, but perhaps not the exact path. Thus the necessary guessings are limited to a polynomial number. We also prove NP-completeness of context matching when the input terms are represented with singleton tree grammars using previously known results (Section 4). In our view this result shows that compression is well-behaved for context matching and can be tamed. The special case of first-order matching with STG-compressed terms is shown to be solvable in PTime by analysing the complexity of operations on STGs (Subsection 4.1). A slight extension of the argument than also shows that first-order unification with STG-compressed terms is in PTime.

2 Preliminaries

We use a *signature* Σ of function symbols that come with an arity $\mathtt{ar}(f)$, where we assume that at least one 0-ary function symbol (a constant) is in Σ. There are two sets of *variables*: first order variables x, y, z, and context variables X, Y, Z of arity 1. We employ the syntax of *second-order terms* (without abstraction) and denote them as $s, t, u, v, \ldots$. The set of variables occurring in terms or other syntactic objects is denoted as $FV(\cdot)$. A term without occurrences of free variables is said to be *ground*. The *size* of a term t is denoted $|t|$ and defined as its number of symbols. We use *positions* in terms, denoted p, q, as sequences of positive integers following Dewey notation. In $f(t_1, \ldots, t_n)$ or $X(r)$, respectively, the position of the function symbol and the context variable X is 0 and the position of the i^{th} argument is i. The empty word is denoted ϵ, $p \prec q$ means the prefix relation, $p \cdot q$ the concatenation, and $t|_p$ the subterm at position p of t. Contexts are terms with a single occurrence of a hole $[\cdot]$, which are denoted by upper case letters C, D. If the term s or context D, respectively, is plugged into the hole of $C[\cdot]$, we denote the result as the term $C[s]$ or the context $C[D]$. The position of the hole in a context D is called *hole path*, and denoted $\mathtt{hp}(C)$, and its length is denoted as $|\mathtt{hp}(C)|$. 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.

Substitutions are functions from terms to terms, defined as usual, where we assume that first-order variables can be instantiated with terms, and context variables can be instantiated with contexts. The application of a substitution σ to a term t is written $\sigma(t)$.

With $[i, n]$ we denote the set $\{i, i+1, \ldots, n\} \subseteq \mathbb{N}$. In later paragraphs we will use rooted directed acyclic graphs (dags) to represent first-order terms. The semantics of the dags is always the represented term, whereas dags are used for arguing about efficiency. We use the same notation for terms and dags. To avoid ambiguity we sometimes use $u =_t v$ to denote that u and v represent the same term.

3 Compression and Encodings

We will describe the compression mechanisms of STGs, the relation of context matching with compression to context unification using encodings and some first complexities.

3.1 Singleton Tree Grammars (STG)

For compacting trees we use singleton tree grammars [13] that can represent dags and e.g. in addition C^n for ground context C and a number n. We define singleton tree grammars as a generalization of singleton context free grammars (SCFG) [10, 15], extending the expressivity of SCFGs by terms and contexts. This is consistent with [3], and also with the context free tree grammars in [6], however, the latter is slightly more general in permitting contexts with several holes.

Definition 3.1 *A* singleton tree grammar (STG) *is a 4-tuple $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$, where $\mathcal{TN}$ are tree nonterminals, $\mathcal{CN}$ are context nonterminals, and Σ is a signature of function symbols and constants (the terminals), such that the sets $\mathcal{TN}$, $\mathcal{CN}$, Σ are pairwise disjoint. The set of nonterminals $\mathcal{N}$ is defined as $\mathcal{N} = \mathcal{TN} \cup \mathcal{CN}$. The rules in R may be of the form:*

- $A ::= f(A_1, \ldots, A_n)$*, where* $A, A_i \in \mathcal{TN}$*, and* $f \in \Sigma_n$*.*
- $A_1 ::= C[A_2]$ *where* $A_1, A_2 \in \mathcal{TN}$*, and* $C \in \mathcal{CN}$*.*
- $C ::= [\cdot]$*.*
- $C_1 ::= C_2C_3$*, where* $C_i \in \mathcal{CN}$*.*
- $C ::= f(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_n)$*, where* $A_i \in \mathcal{TN}$*,* $C \in \mathcal{CN}$*,* $[\cdot]$ *is the hole, and* $f \in \Sigma$ *an* n*-ary function symbol.*

Let $D' >_G D''$ for two nonterminals D', D'', iff $D' ::= t$, and D'' occurs in t. The STG must be non-recursive, i.e. the transitive closure $>_G^*$ must be terminating. Furthermore, for every non-terminal N there is exactly one rule having N as left hand side. Given a term t with occurrences of nonterminals, the derivation by G is an exhaustive iterated replacement of the nonterminals by the corresponding right hand sides, using the convention for second-order terms. The result is denoted as $w_{G,t}$. In the case of a nonterminal C (A) we also say that G defines *$w_{G,C}$ (or $w_{G,A}$, resepctively). As a short hand for $\mathtt{hp}(w_C)$ we use $\mathtt{hp}(C)$ also for context nonterminals C.*

For our purposes of context matching we will also allow first-order and context variables Z in the STG. The convention is that in case there is a rule with left hand side Z, then it is a nonterminal, otherwise we treat Z as terminal.

Note that term dags can efficiently be represented in STGs, but terms may have exponential depth in contrast to dags, which only allow for a linear depth in the size of the dags.

A special case of STGs are singleton context free grammars (SCFG), which can only represent strings. They can be obtained from STGs for the case of a monadic signature. An independent definition of SCFGs is as follows: There are string nonterminals $\mathcal{SN}$, and the possible grammar rules format is: $\mathcal{SN} ::= t$ where t is a string of at most length 2, and consists of letters and string nonterminals. These grammars are non-recursive, for every nonterminal there is exactly one rule, and hence every nonterminal can represent exactly one word. The rest is analogous to STGs.

3.2 Encodings and Complexities

There is a polynomial translation of Context Matching with Dags (CMD) and also of Context Matching with STGs into a context unification problem using the following steps: encode a node t in a dag as a first-order variable x and add an equation $x \doteq f(x_1, \ldots, x_n)$, where x_i are the first-order variables corresponding to the successor-nodes of t. The same can be done for the STG-rules, where context nonterminals are encoded as context variables. The encoding of rules is straightforward, the only slight exception is that rules of the form $C ::= C_1C_2$ have to be encoded as two equations $C[a] = C_1C_2[a], C[b] = C_1C_2[b]$ for two different constants a, b. This translation has the disadvantage that context matching problems are turned into context unification problems for which no decision algorithm is known. For first-order matching where left- and right-hand sides are dags, the translation produces an equivalent first-order unification problem with only a linear size increase, that can be solved using a polynomial first-order unification algorithm. Using the naïve translation of Context Matching with dags or with STG-compressed terms into context matching by simply expanding the terms yields a NEXPTIME algorithm. However, we will show that context matching using compressed terms is in NP: For dags the argumentation is easy: guess an instantiation possibility for every context variable: The ground contexts for the instantiation can be represented in linear space. For every particular guess, it is possible to check in polynomial time whether it solves the given CMD-problem, hence the problem is in NP. For the more general case of STG-compression Theorem 4.10 shows that context matching using STGs is also in NP.

The following hardness result holds for the term-representation, and hence also for the compressed variants.

Proposition 3.2 *Context matching where every context variable occurs at most once, but first order variables are unrestricted, is NP-hard.*

Proof. We encode the positive ONE-IN-THREE-SAT problem (see [7]) For every propositional variable p_i of the given positive ONE-IN-THREE-SAT problem, let x_i, x_i' be first order variables, F_i be a context variable, and there is an equation
$F_i(g(x_i, x_i')) = f(g(a,b), g(b,a))$.
For every clause $C_j = p_{j_1} \vee p_{j_2} \vee p_{j_3}$, let H_j be a fresh context variable, and add an equation as follows:
$H_j(c(x_{i_1}, x_{i_2}, x_{i_3})) = h(c(a,b,b), c(b,a,b), c(b,b,a))$.

This encodes every positive ONE-IN-THREE-SAT-problem as a context matching problem where every context variable occurs at most once, such that the positive ONE-IN-THREE-SAT-problem is satisfiable iff the context matching problem is unifiable. □

Full linearity guarantees polynomiality for CMD (a slight generalization of the term case [20]).

Proposition 3.3 *Context matching where the left- and right-hand side is a dag, and where the left hand side is linear in all variables, is polynomial.*

Proof. We apply dynamic programming tabling the pairs of left and right hand matching subdags from bottom-up. Note that every variable occurs at most once, so no partial substitution has to be remembered. This shows that a table of polynomial size can be built in polynomial time. □

A context matching problem instance obeys *Comon's restriction* (CMCR)[4, 5] if for each context variable all its occurrences are applied to the same term. Proposition 3.2 shows that CMCR is NP-hard.

The restriction that the left hand side is a dag linear in all the variables is a slight variation of Comon's restriction, where first-order variables must occur at most once, and where context matching is polynomial. These comments correct a faulty remark in [20] on context matching using Comon's restriction.

4 Context Matching Using Compression by Singleton Tree Grammars

Now we analyze context matching using singleton tree grammars (see 3.1). We will show that context matching with STG-compression is in NP.

Definition 4.1 *The* size $|G|$ *of a grammar (STG) G is the number of its rules. The* depth *of a nonterminal D is defined as the maximal number of $>_G$-steps from D. The* depth of a grammar *is the maximum of the depths of all nonterminals, denoted as* $\mathtt{depth}(G)$.

As a generalization of the theorem in Plandowski [15, 16], the following theorem for STGs holds:

Theorem 4.2 *([3, 18]) Given an STG G, and two tree nonterminals A, B from G, it is decidable in polynomial time depending on $|G|$ whether $w_A = w_B$.*

The following lemmas state how the size and the depth of an STG are increased by extending it with concatenations, exponentiation, prefixes and suffixes of contexts and subterms/subcontexts of terms and contexts. The depth/size bounds for these operations are related to balancing conditions for trees. The development is borrowed from [13], and the proofs of the lemmas can be derived either from the corresponding proofs in the forthcoming journal version of [11] or are in [13]. We denote with $\log$ the logarithm for base 2 and with $\mathtt{ln}$ the logarithm for base e.

Lemma 4.3 *Let G be an STG defining the contexts $D_1, \ldots, D_n$ for $n \geq 1$. Then there exists an STG $G' \supseteq G$ that defines the context $D_1 \cdot \cdots \cdot D_n$ and satisfies $|G'| \leq |G| + n - 1$ and $\mathtt{depth}(G') \leq \mathtt{depth}(G) + \lceil \log n \rceil$.*

Lemma 4.4 *Let G be an STG defining the context D. For any $n \geq 1$, there exists an STG $G' \supseteq G$ that defines the context D^n and satisfies $|G'| \leq |G| + 2\lfloor \log n \rfloor$ and $\mathtt{depth}(G') \leq \mathtt{depth}(G) + \lceil \log n \rceil$*

Lemma 4.5 *Let G be an STG defining the context D and the term t, respectively. Let D' be a nontrivial prefix or suffix of the context D, or let t' be a subterm of the term t or context D, respectively. Then there exists an STG $G' \supseteq G$ that defines D' or t', respectively, and satisfies $|G'| \leq |G| + \mathtt{depth}(G)$ and $\mathtt{depth}(G') = \mathtt{depth}(G)$.*

Lemma 4.6 covers the case that the hole path of the desired prefix context of a term t (or subcontext of a context) deviates from the paths as given in the STG.

Lemma 4.6 *Let G be an STG defining the term t. For any nontrivial prefix context D of the term t, there exists an STG $G' \supseteq G$ that defines D and satisfies $|G'| \leq |G| + 2\,\mathtt{depth}(G)\,(\log(\mathtt{depth}(G)) + 1)$ and $\mathtt{depth}(G') \leq \mathtt{depth}(G) + 2 + \log(\mathtt{depth}(G))$,*

We will use Lemma 4.3 only for $n = 2$ and we will make no use of Lemma 4.4, hence we formulate a simplified version of a Proposition in [13].

Definition 4.7 *Let G, G' be STGs. Then we write $G \to_{sd} G'$ for a* grammar extension *by size and depth, iff*

$$\begin{array}{lcl} |G'| & \leq & |G| + 3\mathtt{depth}(G) * \log(\mathtt{depth}(G)) + 2 \\ \mathtt{depth}(G') & \leq & \mathtt{depth}(G) + \log(\mathtt{depth}(G)) + 2 \end{array}$$

Proposition 4.8 *Let G, G' be STGs such that $G \to_{sd} \ldots \to_{sd} G'$ in k steps. For simplicity, we assume that $\mathtt{depth}(G) \geq 7$. Then with $D = \mathtt{depth}(G))$ and $\beta(D,k) := D + 2k + k\log(D) + k^2$:*

$$\begin{array}{lcl} |G'| & \leq & |G| + 3k(\beta(D,k))\log(\beta(D,k)) + 2k \\ \mathtt{depth}(G') & \leq & \beta(D,k) \end{array}$$

Proof. Let $G = G_0, G_1, \ldots G_k = G'$ be a sequence of STGs, such that for every $i = 0, \ldots, k-1$: $G_i \rightarrow_{sd} G_{i+1}$. To verify the bound for $\mathtt{depth}(G_k)$, let $d_i := \mathtt{depth}(G_i), i = 1, \ldots, k$. Then $d_{i+1} = d_i + \log(d_i) + 2$, which implies $\mathtt{depth}(G_k) \leq d_0 + 2k + \sum(\log(d_i))$. Using $\log(d_i + a) \leq \log(d_i) + a/(d_i * \mathtt{ln}(2))$, it follows that $\log(d_{i+1}) - \log(d_i) \leq 1$ for $i \geq 2$, where we used the assumption $D \geq 7$. Then we obtain $\mathtt{depth}(G_k) \leq D + 2k + k\log(D) + k^2$. The bound for $|G_k|$ can be derived by summing the upper bounds: $|G_k| \leq |G_0| + 3k\big(\beta(D,k) * \log(\beta(D,k))\big) + 2k$. □ □

Corollary 4.9 *Let G be an STG, and G' be constructed from G by k grammar extensions according to Lemmas 4.3 with $n = 2$, Lemma 4.5 and Lemma 4.6. Then with $D = \mathtt{depth}(G)$:*

$$\begin{array}{lcl} |G'| & = & O(Dk^3(\log(k) + \log(D)) \\ \mathtt{depth}(G') & = & O(Dk^2) \end{array}$$

Theorem 4.10 *Context matching with STGs is in NP, and hence is NP-complete.*

Proof. The proof is aimed at using Theorem 4.2 after guessing instantiations. First we assume that the right as well as the left-hand sides of the match-equations are defined by an STG G, where we use the convention that variables without instantiation are terminals. Now we guess an instantiation to solve the context matching problem. The guessed contexts are contained in the right-hand sides, hence 3 grammar extension steps per context variable are sufficient to obtain a larger STG that defines the guessed contexts: define subterm, define subcontext of a subterm and add a concatenation to the grammar. Hence $3 * K$ steps, where K is the number of context variables in the problem are sufficient to construct a common STG G' for all the instantiations. The same can be done for the first-order variables, and requires a linear number of extensions. Corollary 4.9 shows that the new STG G' has only a polynomial size increase in comparison with G. Now an application of Theorem 4.2 shows that the complexity of checking whether the guessed instantiations are a solution can also be done in polynomial time. Hence context matching with STGs is in NP. Since context matching is already known as NP-hard, we have NP-completeness. □

For the special case matching of strings we obtain also NP-completeness: An instance of the matching problem for strings is a list of equations $s_1 \doteq t_1, \ldots, s_n \doteq t_n$, where s_i, t_i are strings, only s_i may contain string variables, and a solution σ may replace string variables by strings, and must solve all equations, i.e. $\sigma(s_i) = t_i$ for all i.

Corollary 4.11 *String-matching where left and right hand sides are compressed using an SCFG, is NP-complete.*

Proof. It is well-known that string matching is NP-hard, and using a monadic signature, Theorem 4.10 shows the claim. □

4.1 A PTime Algorithm for First-Order Matching with STGs

The encoding of a first-order matcing problem where the terms are compressed with STGs by expansion or the above translation into uncompressed first-order unification does not lead to a PTime algorithm, since there may be an exponential increase of the size. First we will prepare some further polynomial operations on STGs and SCFGs that allow to construct a polynomial algorithm for first-order matching using STGs.

Lemma 4.12 *Given an STG G, a term t that is defined by G, and a terminal symbol x in the STG, it is possible in polynomial time to construct an SCFG that defines the position of the left-most occurrence of x in t.*

Proof. First of all, it is easy to see that given an STG, it is possible in polynomial time to compute a mirror-SCFG that defines the hole-positions of every context non-terminal of G. We denote the position non-terminals for a context non-terminal C as pos_C. It is efficiently checkable which non-terminals contain the terminal x (after expansion). Now the construction of non-terminals $q_{A,x}, q_{C.x}$ is as follows: If $x = A_i$ in the rule $A ::= f(A_1, \ldots, A_n)$, then $q_{A,x} ::= i$. For the rule $C ::= C_1[A]$, there are two cases: if x occurs in w_C, then $q_C ::= q_{C_1}$, otherwise if $x = w_A$, then $q_C ::= pos_{C_1}$. For the rule $C ::= C_1C_2$, there are two cases: if x occurs in w_{C_1}, then $q_C ::= q_{C_1}$, and otherwise, if x occurs in w_{C_2}, then $q_C ::= pos_{C_1}q_{C_2}$. Finally, f T is the nonterminal that defines T, the nonterminal q_T defines the position of the leftmost x in the constructed SCFG. □

Lemma 4.13 *Given an STG G, a nonterminal t, an SCFG P encoding positions wth $|P| \leq |G|$, and a nonterminal p in P. Then it is possible in polynomial time to extend G to G' such that $(w_t)_{|w_p}$ is defined in G', where the size increase is as in Lemma 4.5.*

Proof. It is sufficient to show that we can navigate to the corresponding position in G, since the construction and size increase is already treated in Lemma 4.5.
It is possible in polynomial time to compute the length of all defined positions in P. The same can be done for the positions in the SCFG Q defining the hole-positions of all context nonterminals of G. Now we have to walk down the structure of both grammars. A single step has as state a pair (C, p), or (A, p), such that the position w_p is in w_C or w_A, respectively. Let $p ::= p_1p_2$ be the rule in G. We have to look for the cases of rules of C, or A, respectively. If

$C ::= C_1C_2$, then we can construct the prefix p' of p of length $|w_{\mathtt{hp}(C_1)}|$ in P in polynomial time, and then compare it with $\mathtt{hp}(w_{C_1})$, also in PTime. There are two cases:

1. If $w_{p'} = \mathtt{hp}(w_{C_1})$, then we have to construct a nonterminal p'' in P for the suffix of w_p with $w_{p'}w_{p''} = w_p$ and proceed with the pair (C_2, p''). There is a size increase of P.

2. Otherwise, w_p is a posiiton within w_{C_1}. Then proceed with the pair (C_1, p).

In a similar way, we proceed for the other rules. This may be a polynomial number of steps and a corresponding size increase of P, which is polynomial by Lemma 4.5 and Corollary 4.9 Finally we obtain in polynomial time an extension of G that defines $(w_t)_{|w_p}$. □

Theorem 4.14 *First-order matching where terms are compressed using an STG can be done in polynomial time, where the term and the grammar count as input.*

Proof. Given a match equation $s \doteq t$ and an STG G, we proceed as follows: For every first-order variable x, we first construct a grammar for the leftmost position p of x, and then extend G by constructing a non-terminal r_x that represents t at position p. This can be done in polynomial time according to Lemmas 4.12 and 4.13. Then add the rule $x ::= r_x$ to the grammar G. This can be done for all variables in s. The size increase of the final grammar is polynomial due to the Lemmas 4.3, Lemma 4.5 and Lemma 4.6. Finally, we compare the non-terminals for s and t w.r.t, to the final STG using Theorem 4.2. As a summary, all operations can be done in polynomial time, hence the claim holds. □

Using the same constructions and a slight extension, it is possible to show that first-order unification using STGs is also polynomial. This does not follow from the results in [12, 13], since the algorithm in these papers is nondeterministic.

Theorem 4.15 *First-order unification where terms are compressed using an STG G can be done in polynomial time and where the terms and the grammar G count as input.*

Proof. The only two additions to the algorithm given for first-order matching in the proof of Theorem 4.14 are:

1. It may not be possible to construct a corresponding term r_x for x, since the other term contains a first-order variable y on the path. In this case, switch the role of s, t, and compute the corresponding term r_y. All the checks and computations can be done in polynomial time, using similar arguments as in the proofs above.

2. Check for an occurs-check-situation, which can also be done in polynomial time.

□

5 A PTime algorithm for k-Context Matching with Dags

The context matching problem is NP-complete. In this section we reconsider this problem by introducing the additional restriction stating that the maximum number k of different context variables of a given instance is fixed for the problem. We call k-context matching to this problem. Hence, we have a family of problems indexed by k.

A polynomial time algorithm for k-context matching (with uncompressed terms) can be easily obtained. Suppose we are given an instance $\{s \doteq t\}$ of the problem, where t is a ground term and s contains at most k different context variables. Any solution of $\{s \doteq t\}$ instantiates any context variable by a context occurring in t, and the number of different contexts in t is bounded by $|\mathtt{Pos}(t)|^2$. This is because any context occurring in t can be defined by two positions of $\mathtt{Pos}(t)$: the root position and the hole position of the context. Hence, it suffices to do at most k guessings of contexts for the context variables, every one along $|\mathtt{Pos}(t)|^2$ possibilities, apply this partial substitution, and check if the resulting first-order matching problem has a solution.

When the input is compressed with dags, the problem becomes more difficult.

5.1 Inferring the context

One of the key points for obtaining a polynomial time algorithm is the fact that, in some cases, the context solution for a context variable can be inferred. Consider the simple case where we have two matching equations of the form $F(s) \doteq u$ and $F(t) \doteq v$, and suppose that u and v are different. Suppose also that we know the existence of a solution σ for these equations, but the only known information for σ is $|\mathtt{hp}(\sigma(F))|$, i.e. just the length to the hole position of $\sigma(F)$ and nothing else. It can be proved that this information suffices to obtain the whole $\sigma(F)$. With this aim we define below $\mathtt{InfCon}(u, v, l)$ for any rooted dags u and v, and natural number l, which intuitively corresponds to the supposed $|\mathtt{hp}(\sigma(F))|$.

Definition 5.1 *Let u and v be two different ground rooted dags with $u \neq_t v$, and let l be a natural. We define $\mathtt{InfCon}(u, v, 0)$ to be the empty context $[\cdot]$. We also define $\mathtt{InfCon}(f(u_1, \ldots, u_m), g(v_1, \ldots, v_m), l + 1) = f(u_1, \ldots, u_{i-1}, \mathtt{InfCon}(u_i, v_i, l), u_{i+1}, \ldots, u_m)$ in the case where $f = g$ and there exists a natural number $i \in [m]$ such that $u_j =_t v_j$ for all $j \neq i$. Otherwise, $\mathtt{InfCon}(f(u_1, \ldots, u_m), f(v_1, \ldots, v_m), l + 1)$ is undefined.*

Note that in the second case of the previous definition, if $f = g$ and such an i exists, then it is unique. This is because $f(u_1, \ldots, u_m)$ and $g(v_1, \ldots, v_m)$ are different, and hence, $u_j =_t v_j$ for all $j \neq i$ implies that $u_i \neq_t v_i$.

Example 5.2 *Let* u, v, w *be* $f(a, g(h(a, a), c), b)$, $f(a, g(h(b, b), c), b)$, $g(f(a, b, c), b)$, *respectively. Then,* $\mathtt{InfCon}(u, v, 0) = \mathtt{InfCon}(u, w, 0) = [\cdot]$, $\mathtt{InfCon}(u, v, 1) = f(a, [\cdot], b)$, $\mathtt{InfCon}(u, w, 1)$ *is undefined,* $\mathtt{InfCon}(u, v, 2) = f(a, g([\cdot], c), b)$, *and* $\mathtt{InfCon}(u, v, 3)$ *is undefined.*

Lemma 5.3 *Let* u, v *be ground rooted dags with* $u \neq_t v$. *Let* s, t *be dags. Let* σ *be a solution of* $\{F(s) \doteq u, F(t) \doteq v\}$. *Then* $\sigma(F) = \mathtt{InfCon}(u, v, |\mathtt{hp}(\sigma(F))|)$.

Proof. We prove the claim by induction on $|\mathtt{hp}(\sigma(F))|$. If $|\mathtt{hp}(\sigma(F))|$ is 0, then $\sigma(F)$ is $[\cdot]$, which coincides with $\mathtt{InfCon}(u, v, |\mathtt{hp}(\sigma(F))|)$. Now, suppose that $|\mathtt{hp}(\sigma(F))|$ is $l + 1$ for some natural number l. This implies that $\sigma(F)$ is of the form $f(w_1, \ldots, w_{i-1}, C[\cdot], w_{i+1}, \ldots, w_m)$ for some function symbol f and some $i \in [m]$. Since σ is a solution of $\{F(s) \doteq u, F(t) \doteq v\}$, the dags u and v are of the form $f(u_1, \ldots, u_m)$ and $f(v_1, \ldots, v_m)$, respectively. For the same reason, $w_j =_t u_j =_t v_j$ for all $j \neq i$, and moreover, $\sigma(C[s]) =_t u_i$ and $\sigma(C[t]) =_t v_i$, and $u_i \neq_t v_i$. Consider a new context variable F' and the extension of σ as $\sigma(F') = C[\cdot]$. Then, σ is also a solution of $\{F'(s) \doteq u_i, F'(t) \doteq v_i\}$. Note that $|\mathtt{hp}(\sigma(F'))|$ is l, which is smaller than $|\mathtt{hp}(\sigma(F))|$. By induction hypothesis, $\sigma(F') = \mathtt{InfCon}(u_i, v_i, |\mathtt{hp}(\sigma(F'))|)$. Now, we can conclude that $\sigma(F)$ is equal to:

$$
\begin{array}{ll}
=_t & f(w_1, \ldots, w_{i-1}, C[\cdot], w_{i+1}, \ldots, w_m) \\
=_t & f(w_1, \ldots, w_{i-1}, F', w_{i+1}, \ldots, w_m) \\
=_t & f(w_1, \ldots, w_{i-1}, \\
 & \quad \mathtt{InfCon}(u_i, v_i, |\mathtt{hp}(\sigma(F'))|), w_{i+1}, \ldots, w_m) \\
=_t & \mathtt{InfCon}(u, v, |\mathtt{hp}(\sigma(F))|) \qquad \square
\end{array}
$$

5.2 The k-CMD Algorithm

Definition 5.4 *The rules of the matching algorithm for dags are in figure 1.*

Initially, we compact the right hand sides, such that equal subterms are represented by the same dag. L *is the maximum height of the initial right-hand-sides* $t_1, \ldots, t_n$. *We write the match-equations as* $s_i \doteq t_i$, *assuming that the right-sides* t_i *are a ground rooted dags. As notation for several instantiations of context variables we also allow* $F \mapsto \mathtt{AllCon}(s_1, s_2)$, *if* s_1, s_2 *are dags and* s_2 *is a subdag of* s_1. *The semantics is that* $\mathtt{AllCon}(s_1, s_2)$ *is the set of all contexts* C, *such that* $C[s_2] =_t s_1$.

Note that after applying any rule on a set Δ producing a set Δ', any right-hand side t'_i of an equation in Δ' is a subdag of some right-hand side t_j of an equation in Δ. Hence, L is also a bound for the height of the dags at the right-hand sides of equations of Δ'.

Lemma 5.5 *The set of rules is sound.*

Proof. Let Δ be a set of equations, and let Δ' be a resulting set from applying an inference step on Δ. We show that every solution of Δ' is also a solution of Δ'.

This is easy for all the rules. Note that rules `Var-ElimF1`, `Var-ElimF3` and `Var-ElimF4` give an instantiation $\sigma(\Delta)$ of the original set Δ as result. Hence, from any solution θ of $\sigma(\Delta)$ we derive a solution $\theta(\sigma)$ for Δ. This argument is analogous for the rule `Var-Elimx`. The rule `Decompose` gives a set Δ' as result satisfying that if σ is a solution of Δ' then σ is also a solution of Δ. For rule `Fail`, it is obvious that the assumption of a solution σ for the resulting set Δ' can not be satisfied. $\square$

Our set of rules represents, in fact, a family of algorithms, parameterized by the strategy for deciding the applied rule at every step. On the other side, any of the possible strategies produces a non-deterministic algorithm, since some rules require guessings.

Lemma 5.6 *The set of rules is complete. Moreover, any sequence of rule applications computes a representation of all solutions, by gathering all guesses in the rules.*

Proof. Let ∇ be a set of equations with a solution σ. It suffices to show that after applying any applicable rule to ∇, one of the resulting sets of equations among the possible guesses also has σ as solution. We distinguish the cases depending on which inference step is applied.

- For the `Decompose`- and `Var-Elimx`-rules this is trivial, and for the `Fail` rule it is clear that the assumption on the existence of a solution is not possible. We treat the rest of cases as follows.
- Suppose that we apply `Var-ElimF1` rule. We assume the existence of a solution σ for the initial set of equations $\Delta \cup \{F(s_1) \doteq t_1, F(s_2) \doteq t_2\}$. By the conditions for this rule application, $t_1 \neq_t t_2$. By Lemma 5.3, $\sigma(F)$ is $\mathtt{InfCon}(t_1, t_2, |\mathtt{hp}(\sigma(F))|)$. It is obvious that $|\mathtt{hp}(\sigma(F))|$ is smaller than or equal to L. Hence, $|\mathtt{hp}(\sigma(F))| \in [0, L]$, and we can consider the case where l is guessed to $|\mathtt{hp}(\sigma(F))|$ in this rule application. Since σ is a ground substitution, $\sigma(u) =_t \sigma(\{F \mapsto \sigma(F)\}(u)) =_t \sigma(\{F \mapsto \mathtt{InfCon}(t_1, t_2, |\mathtt{hp}(\sigma(F))|)\}(u))$ holds for any dag u, and hence, σ is also a solution of $\{F \mapsto \mathtt{InfCon}(t_1, t_2, l)\}(\Delta \cup \{F(s_1) \doteq t_1, F(s_2) \doteq t_2\})$.
- Suppose that we apply the `Var-ElimF2` rule. We assume the existence of a solution σ for the set of equations $\Delta \cup \{F(s_1) \doteq t, F(s_2) \doteq t, \ldots, F(s_n) \doteq t\}$. In particular, $\sigma(F(s_1)) =_t t, \sigma(F(s_2)) =_t t, \ldots, \sigma(F(s_n)) =_t t$, and hence, there exists a subdag t' of t with $t' =_t \sigma(s_1) =_t \ldots =_t \sigma(s_n)$. Therefore, σ is also a solution of $\Delta \cup \{s_1 \doteq t', \ldots, s_n \doteq t'\}$. Also, $\sigma(F)$ is represented by $\mathtt{AllCon}(t, t')$, since it is one of the contexts in $\mathtt{AllCon}(t, t')$.

Decompose: $$\frac{\Delta \cup \{f(s_1,\ldots,s_m) \doteq f(t_1,\ldots,t_m)\}}{\Delta \cup \{s_1 \doteq t_1,\ldots,s_m \doteq t_m\}}$$
where f is a function symbol ($n = \mathtt{arity}(f)$).

Fail: $$\frac{\Delta \cup \{f(t_1,\ldots,t_n) \doteq g(u_1,\ldots,u_m)\}}{\bot}$$
where $f \neq g$.

Var-Elimx: $$\frac{\Delta \cup \{x \doteq t\}}{\{x \mapsto t\}(\Delta)}$$
where x is a first-order variable.

Var-ElimF1: $$\frac{\Delta \cup \{F(s_1) \doteq t_1, F(s_2) \doteq t_2\}}{\{F \mapsto \mathtt{InfCon}(t_1,t_2,l)\}(\Delta \cup \{F(s_1) \doteq t_1, F(s_2) \doteq t_2\})}$$
if $t_1 \neq_t t_2$. The number l is guessed over $[0, L]$, where $\mathtt{InfCon}(t_1,t_2,l)$ must be defined.

Var-ElimF2: $$\frac{\Delta \cup \{F(s_1) \doteq t, F(s_2) \doteq t, \ldots, F(s_m) \doteq t\}}{\Delta \cup \{s_1 \doteq t', \ldots, s_m \doteq t'\}}$$
where F is a context variable not occurring in $\Delta \cup \{s_1 \doteq t', \ldots, s_m \doteq t'\}$. The dag t' is guessed over $\mathtt{Subdags}(t)$. The instantiation for F is $F \mapsto \mathtt{AllCon}(t,t')$.

Var-ElimF3: $$\frac{\Delta \cup \{F(s) \doteq t\}}{\{F \mapsto [\cdot]\}(\Delta \cup \{F(s) \doteq t\}) \mid \{F \mapsto \mathtt{InfCon}(t,t',l)\}(\Delta \cup \{F(s) \doteq t\})}$$
where F occurs in s. The dag t' is guessed over $\mathtt{Subdags}(t) \setminus \{t\}$, l is guessed over $[0, L]$, where $\mathtt{InfCon}(t,t',l)$ must be defined.

Var-ElimF4: $$\frac{\Delta}{\{F_2 \mapsto [\cdot]\}(\Delta) \mid \{F_1 \mapsto \mathtt{InfCon}(t_1,t_2',l)\}(\Delta)}$$
if no other rule can be applied. Select (don't care) two equations $F_1(s_1) \doteq t_1, F_2(s_2) \doteq t_2$ in Δ with $F_1 \neq F_2$, F_1 occurs in s_2 and $\mathtt{height}(t_1) \geq \mathtt{height}(t_2)$. The dag t_2' is guessed over $\mathtt{Subdags}(t_2) \setminus \{t_1\}$, and l is guessed over $[1, L]$, where $\mathtt{InfCon}(t_1,t_2',l)$ must be defined.

Figure 1. Rules of the Dag-Context-Matching Algorithm

- Suppose that we apply the Var-ElimF3 rule. We assume the existence of a solution σ for the set of equations $\Delta \cup \{F(s) \doteq t\}$. Since F occurs in s, there exists a subdag of $F(s)$ of the form $F(s')$. Since $\sigma(F(s)) = t$ holds, there exists a proper subdag t' of t such that $\sigma(F(s')) = t'$. The case $\sigma(F) = [\cdot]$ is covered by the first alternative of the rule. Now assume that $\sigma(F) \neq [\cdot]$. Then $t' \neq_t t$. Hence, by Lemma 5.3, $\sigma(F)$ is $\mathtt{InfCon}(t,t',|\mathtt{hp}(\sigma(F))|)$. In this case $|\mathtt{hp}(\sigma(F))| \geq 1$, and also $|\mathtt{hp}(\sigma(F))| \leq L$. Hence, $|\mathtt{hp}(\sigma(F))| \in [1, L]$, and we can consider the case where l is guessed to $|\mathtt{hp}(\sigma(F))|$ in this rule application. Since σ is a ground substitution, $\sigma(u) = \sigma(\{F \mapsto \sigma(F)\}(u)) = \sigma(\{F \mapsto \mathtt{InfCon}(t,t',|\mathtt{hp}(\sigma(F))|)\}(u))$ holds for any dag u, and hence, σ is also a solution of $\{F \mapsto \mathtt{InfCon}(t,t',l)\}(\Delta \cup \{F(s) \doteq t\})$.

- Suppose that we apply the Var-ElimF4 rule. We assume the existence of a solution σ for the set Δ of equations. By the conditions for this rule application, no other rule can be applied. Since rules Decompose, Fail, Var-Elimx, can not be applied, every equation in Δ is of the form $F(s) \doteq t$ for some context variable F. Moreover, since rule Var-ElimF2 and Var-ElimF3 can not be applied, every context variable F occurring in Δ satisfies that there exists an equation $G(s) \doteq t$ in Δ, such that G is different from F, and F occurs in s. Since the set Δ is finite, there exist equations $F_1(s_1) \doteq t_1, F_2(s_2) \doteq t_2, \ldots, F_n(s_n) \doteq t_n$ with $n \geq 2$ satisfying that F_1 occurs in s_2, F_2 occurs in s_3, ..., F_{n-1} occurs in s_n, and F_n occurs in s_1, and where the F_i's are pairwise different. Therefore, there are dags of the form $F_n(s_1'), F_1(s_2') \ldots, F_{n-1}(s_n')$ which are subdags of $s_1, s_2, \ldots, s_n$, respectively. Since σ is a solution of Δ, it holds that $\sigma(F_1(s_1)) =_t t_1, \sigma(F_2(s_2)) =_t t_2, \ldots, \sigma(F_n(s_n)) =_t t_n$. We select the index i such that t_i has maximal height among the $t_1, \ldots, t_n$. Without loss of generality we can assume $i = 1$. The

rule now focusses on the two equations $F_1(s_1)) \doteq t_1, F_2(s_2)) \doteq t_2$ If $\sigma(F_2) = [\cdot]$, we apply the first alternative of the rule, and then it is easy to see that σ is also a solution of the resulting set of equations.
Now assume that $\sigma(F_2) \neq [\cdot]$. Then there exists a proper subdag t_2' of t_2, such that $\sigma(F_1(s_2')) = t_2'$. Since $\mathtt{height}(t_1) \geq \mathtt{height}(t_2) > \mathtt{height}(t_2')$, it holds that $t_1 \neq_t t_2'$. Hence by Lemma 5.3, $\sigma(F_1)$ is the same context as $\mathtt{InfCon}(t_1, t_2', |\mathtt{hp}(\sigma(F_1))|)$. Also, $0 \leq |\mathtt{hp}(\sigma(F_1))| \leq L$. Hence we can consider the case where l is guessed to $|\mathtt{hp}(\sigma(F_1))|$ in this rule application. Since σ is a ground substitution, $\sigma(u) = \sigma(\{F \mapsto \sigma(F)\}(u)) = \sigma(\{F \mapsto \mathtt{InfCon}(t_1, t_2', |\mathtt{hp}(\sigma(F))|)\}(u))$ holds for any dag u, and hence, σ is also a solution of $\{F \mapsto \mathtt{InfCon}(t_1, t_2', l)\}(\Delta)$.

□ □

6 Complexity of the k-CMD Algorithm

First we summarize our assumptions on the implementation of dags and the algorithm. For the right hand sides we assumed that in a first compaction step, different nodes represent also different terms. For the left hand sides, we leave the dags as they are inputted. Let K be the number of context variables in the input, let N be the size of the input, and let L be as defined above, the maximal height or right-hand dags.

The algorithm may change the left hand side dags by two operations:

- A substitution $\{x \mapsto t\}$ in the rule `Var-Elimx`. This is implemented as redirecting the edges that point to x to edges that point to the dag t. Assuming an appropriate data structure, this can be performed in linear time. In particular, it does not increase the number of nodes.
- a substitution $F \mapsto C$ in the `VarElimF`-rules, where C is a ground context. For the ground context C, it holds that $|\mathtt{hp}(C)| \leq L$. Since the arity of function symbols is $O(1)$, this will add at most L new nodes in the global dag. The rules can perform at most K times.

Hence we have argued that the following holds:

Lemma 6.1 *The algorithm increases the size of the input by at most $K * L$ nodes, which is less than $K * N$.*

Lemma 6.2 *There are at most K applications of `VarElimF`-rules, and at most N applications of the `Decompose` and of the `Var-Elimx`-rules.*

Proposition 6.3 *The number of final representations of solutions is at most $(K * N * L)^K$, which is smaller than $(K * N * N)^K$,*

Proof. The maximal contribution of the different rules are

`Var-ElimF1`	`Var-ElimF2`
L	$K * N$
`Var-ElimF3`	`Var-ElimF4`
$K * N * L$	$K * N * L$

This holds, since the possibilities for guessing a dag are at most $K * N$, and the possibilities for guessing a height is bounded by L. □

All the computations during the application of rules, like computing the height, the subdags, or the `InfCon` are polynomial, hence:

Theorem 6.4 *If the number of context variables is a constant K at the start of the algorithm, then an explicit representation of all solutions of size at most $O(N^{2K})$ can be computed in polynomial time. In particular, solvability can be decided in polynomial time.*

Example 6.5 *An example for an exponential number of matches is the dag corresponding to a complete binary tree t_n of depth n, built with binary function symbol f, and the constant a, and the single equation $F(a) \doteq t_n$. There are exponentially many solutions, but they are represented as* $\mathtt{AllCon}(t_n, a)$.

7 Conclusion

We analyzed the complexity of context matching under different compression techniques like dags and STGs. We showed that context matching using STGs is NP-complete and constructed a polynomial context matching algorithm with dags if the number of context variables is fixed. We left open the complexity of the linear context matching with STGs.

References

[1] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science and MIT Press, 2001.

[2] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon, editors. *XML Path Language (XPath) Version 2.0*. W3C, 2007. http://www.w3.org/TR/2007/REC-xpath20-20070123/.

[3] G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML documents. In *Proceedings of DBPL 2005*, volume 3774 of *LNCS*, pages 199–216, 2005.

[4] H. Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *Journal of Symbolic Computation*, 25(4):397–419, 1998.

[5] H. Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *Journal of Symbolic Computation*, 25(4):421–453, 1998.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997. release 1.10.2002.

[7] M. Garey and D. Johnson. *Computers and Intractability*. W.H. Freeman, 1979.

[8] G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.

[9] S. Gulwani and A. Tiwari. Computing procedure summaries for interprocedural analysis. In *Proc. European Symp. on Programming, ESOP 2007*, volume 4421 of *LNCS*, pages 253–267. Springer, 2007.

[10] J. Levy, M. Schmidt-Schauß, and M. Villaret. Monadic second-order unification is NP-complete. In *Rewriting Techniques and Applications (RTA-15)*, volume 3091 of *LNCS*, pages 55–69. Springer, 2004.

[11] J. Levy, M. Schmidt-Schauß, and M. Villaret. Monadic second-order unification is NP-complete. In *RTA-15*, volume 3091 of *LNCS*, pages 55–69. Springer, 2004.

[12] J. Levy, M. Schmidt-Schauß, and M. Villaret. Bounded second-order unification is NP-complete. In *Proc. RTA-17*, volume 4098 of *LNCS*, pages 400–414. Springer, 2006.

[13] J. Levy, M. Schmidt-Schauß, and M. Villaret. Stratified context unification is NP-complete. In *Proc. Third Intl. Joint Conf. on Automated Reasoning, IJCAR 2006*, volume 4130 of *LNCS*, pages 82–96. Springer, 2006.

[14] J. Niehren, M. Pinkal, and P. Ruhrberg. A uniform approach to underspecification and parallelism. In *Proceedings of 35th ACL'97*, pages 410–417, Madrid, Spain, 1997.

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

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

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

[18] 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, November 2005.

[19] M. Schmidt-Schauß and K. U. Schulz. Solvability of context equations with two context variables is decidable. *J. Symb. Comput.*, 33(1):77–122, 2002.

[20] M. Schmidt-Schauß and J. Stuber. On the complexity of linear and stratified context matching problems. *Theory of Computing Systems*, 37:717–740, 2004.

A Proofs for STG-Extensions

The proofs are borrowed from a forthcoming journal version of [11] and from [13].

Lemma A.1 *Let G be an STG defining the contexts $D_1, \ldots, D_n$ for $n \geq 1$. Then there exists an STG $G' \supseteq G$ that defines the context $D_1 \cdot \cdots \cdot D_n$ and satisfies $|G'| \leq |G| + n - 1$ and $\mathtt{depth}(G') \leq \mathtt{depth}(G) + \lceil \log n \rceil$.*

Proof. Let A_i be the non-terminal symbol generating the contexts D_i, for any $i = 1, \ldots, n$. 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 $b_{i,i}$ is A_i. Then, $b_{1,n}$ generates $D_1 \ldots D_n$, and to generate it we only need to add $n-1$ of such rules. The depth is increased by at most $\lceil \log n \rceil$. □

Lemma A.2 *Let G be an STG defining the context D. For any $n \geq 1$, there exists an STG $G' \supseteq G$ that defines the context D^n and satisfies $|G'| \leq |G| + 2\lfloor \log n \rfloor$ and $\mathtt{depth}(G') \leq \mathtt{depth}(G) + \lceil \log n \rceil$*

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

$$\begin{array}{lcll}
a_1 & \to & a\, a & \\
a_2 & \to & a_1\, a_1 & \\
 & \cdots & & \\
a_m & \to & a_{m-1}\, a_{m-1} & \\
b_0 & \to & \left\{ \begin{array}{ll} a & \text{if } k_0 = 1 \\ [\cdot] & \text{if } k_0 = 0 \end{array} \right. & \\
b_1 & \to & \left\{ \begin{array}{ll} a_1\, b_0 & \text{if } k_1 = 1 \\ b_0 & \text{if } k_1 = 0 \end{array} \right. & \\
 & \cdots & & \\
b_m & \to & \left\{ \begin{array}{ll} a_m\, b_{m-1} & \text{if } k_m = 1 \\ b_{m-1} & \text{if } k_m = 0 \end{array} \right. &
\end{array}$$

Then, the non-terminal symbol b_m generates D^n, and it is easy to see that this grammar satisfies the bounds stated by the lemma. □

Lemma A.3 *Let G be an STG defining the context D and the term t, respectively. Let D' be a nontrivial prefix or suffix of the context D, or let t' be a subterm of the term t or context D, respectively. Then there exists an STG $G' \supseteq G$ that defines D' or t', respectively, and satisfies $|G'| \leq |G| + \mathtt{depth}(G)$ and $\mathtt{depth}(G') = \mathtt{depth}(G)$.*

Proof. Let C be the non-terminal symbol defining the context $D = w_C$. By induction on $\texttt{depth}_G(C)$, we will prove a stronger result: for any nontrivial prefix D' of w_C, there exists a grammar $G' \supseteq G$ and a nonterminal C' defining D', and satisfying $\texttt{depth}(C') \leq \texttt{depth}(C)$, $|G'| \leq |G| + \texttt{depth}_G(C)$ and $\texttt{depth}(G') = \texttt{depth}(G)$.

The base case is trivial since $\texttt{depth}_G(C) = 1$ implies that the rule for C is of the form $C ::= (A_1, \ldots, A_n)$. For the induction case, assume that $D' \neq D$ and that D' is not trivial, otherwise we are done. Let $C ::= C_1 C_2$ be the rule for C. If D' is a prefix of w_{C_1}, then we can use induction and obtain $|G'| \leq |G| + \texttt{depth}_G(C_1) < |G| + \texttt{depth}_G(C)$. Let w_{C_1} be a proper prefix of D'. Then $D' = w_{C_1} D''$ and D'' is a prefix of w_{C_2}. By induction hypothesis, there exists a grammar G'' deriving D'' from some C'' with $\texttt{depth}(C'') \leq \texttt{depth}(C_2)$, $\texttt{depth}(G') = \texttt{depth}(G)$ and size $|G''| \leq |G| + \texttt{depth}(C'') \leq |G| + \texttt{depth}(C) - 1$. We add $C' ::= C_1 C''$ to get the grammar G' from G'' such that $w_{C'} = D'$. Notice that $|G'| = |G''| + 1$, and $\texttt{depth}(G') = \texttt{depth}(G'')$ because $\texttt{depth}(C'') \leq \texttt{depth}(C_2)$ implies $\texttt{depth}(C') \leq \texttt{depth}(C)$.
For suffixes the proof is very similar. □ □

Lemma A.4 covers the case that the hole path of the desired prefix context of a term t (or subcontext of a context) deviates from the paths as given in the STG.

Lemma A.4 *Let G be an STG defining the term t. For any nontrivial prefix context D of the term t, there exists an STG $G' \supseteq G$ that defines D and satisfies $|G'| \leq |G| + 2\,\texttt{depth}(G)\,(\log(\texttt{depth}(G)) + 1)$ and $\texttt{depth}(G') \leq \texttt{depth}(G) + 2 + \log(\texttt{depth}(G))$,*

Proof.

Let A be the non-terminal symbol defining the term $t = w_A$ and let p be a position in w_A that is the position of the hole of the desired context D. First we show by induction that we can generate a list of context nonterminals that can be concatenated to construct D. The induction is on $\texttt{depth}(A)$.

The base case is that $|p| = 0$ at some depth. In this case the empty context is the result, which is omitted in the list. For the induction step we consider the different possibilities for rules:

1. The rule is $A ::= f(A_1, \ldots, A_n)$ and $p = kp'$. Then we return the context defined by the rule $C_1 ::= f(A_1, \ldots, [\cdot]_k, \ldots, A_n)$, and the list for A_k, p'.

2. The rule is $A ::= C[A_2]$. There are some subcases:
   If p is a prefix of $\texttt{mp}(C)$, then return C_1, constructed such that $p = \texttt{mp}(C_1)$ using Lemma A.3.
   If p is within A_2, and $p = p_1 p_2$, where $p_1 = \texttt{mp}(C)$, then we return C, and the list of contexts generated for A_2, p_2.
   The position p is within C. Then let $p = p_1 p_2 p_3$, where p_1 is the maximal common prefix of p and $\texttt{mp}(C)$, and $|p_2| = 1$. Then construct C_1 for the prefix of w_C with $p_1 = \texttt{mp}(C_1)$ by Lemma A.3. Let $p_1 k$ with $k \in \mathbb{N}$ be a prefix of $\texttt{mp}(C)$. Let C_3 be a new symbol defining the subcontext of w_C starting at position $p_1 k$ using Lemma A.3. Moreover, there is a defined rule $C_2 ::= f(B_1, \ldots, [\cdot]_k, \ldots B_n)$, corresponding to the subcontext of w_C for position p_1, whose existence can be verified by induction. Since $p_2 \neq k$, we have to define the following new symbols and rules: $A_3 ::= C_3[A_2]$, $C_4 ::= f(B_1, \ldots, [\cdot]_{p_2}, \ldots, B_{k-1}, A_3, B_{k+1}, \ldots, B_n)$. Then return C_1, C_4 and the list generated for B_{p_2}, p_3.

Summarizing, we obtain a list of contexts of length at most $2\texttt{depth}(G)$, which can be concatenated defining a new symbol C_D. An upper bound on the total number of new rules is $(2\log(\texttt{depth}(G)) + 2) * \texttt{depth}(G)$, since the induction hypothesis in case 2 is called for $\texttt{depth}(A) - 2$. Notice that the depth of all the contexts that we build up is bounded by $\texttt{depth}(G) + 1$ because of the construction of C_4, hence the depth of C_D is at most $\texttt{depth}(G) + 2 + \log(\texttt{depth}(G))$, which is the depth contribution of the final concatenation. □ □