Pattern Matching of Compressed Terms and Contexts and Polynomial Rewriting

Manfred Schmidt-Schauß[1]

Institut für Informatik
Johann Wolfgang Goethe-Universität
Postfach 11 19 32
D-60054 Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de

Technical Report Frank-43

Research group for Artificial Intelligence and Software Technology
Institut für Informatik,
Fachbereich Informatik und Mathematik,
Johann Wolfgang Goethe-Universität,
Postfach 11 19 32, D-60054 Frankfurt, Germany

September 15, 2011

Abstract. A generalization of the compressed string pattern match that applies to terms with variables is investigated: Given terms s and t compressed by singleton tree grammars, the task is to find an instance of s that occurs as a subterm in t. We show that this problem is in NP and that the task can be performed in time $O(n^{c|\,Var(s)|})$, including the construction of the compressed substitution, and a representation of all occurrences. We show that the special case where s is uncompressed can be performed in polynomial time. As a nice application we show that for an equational deduction of t to t' by an equality axiom $l = r$ (a rewrite) a single step can be performed in polynomial time in the size of compression of t and l, r if the number of variables is fixed in l. We also show that n rewriting steps can be performed in polynomial time, if the equational axioms are compressed and assumed to be constant for the rewriting sequence. Another potential application are querying mechanisms on compressed XML-data bases.

1 Introduction

An important concept in various areas of computer science like automated deduction, first order logic, term rewriting, type checking, are terms (ranked trees), and also terms containing variables (see e.g. [BN98]). The basic and widely used algorithms in these areas are matching, unification, equational rewriting, asf. For

example, a term $f(g(a,b),c)$ may be rewritten into $f(g(b,a),c)$ by the commutativity axiom $g(x,y) = g(y,x)$ for g. Since implemented systems often deal with large terms, perhaps generated ones, it is of high interest to look for compression mechanisms for terms, and consequently, also investigate variants of the known algorithms that also perform efficiently on the compressed terms without prior decompression.

The device of straight line programs (SLP) (context free grammars that generate exactly one string) for the compression of strings is a general one and allows clean mathematical proofs [Ryt04] of correctness and complexity of algorithms. SLPs are polynomially equivalent to the variants of Lempel-Ziv compression. These are non-cyclic CFGs where every nonterminal has exactly one rule in the CFG, such that any nonterminal represents exactly one string. Basic algorithms are the equality check of two compressed strings, which is polynomial [Pla94], (see [Lif07] for an efficient version), and the compressed pattern match, i.e., given two SLP-compressed strings s, t, is s a substring of t, which can also be solved in polynomial time in the size of the SLPs.

A generalization of SLPs for the compression of terms are singleton tree grammars (STG) [SS05,LSSV06,LSSV10,GGSS08], a specialization of straight line context free tree grammars [BLM05,BLM08,LMSS09a], where linear SLCF tree grammars are polynomially equivalent to STGs [LMSS09a]. Basic notions for tree grammars and tree automata can be found in [CDG$^+$97]. Besides using the well-known node sharing, also partial subtrees (contexts) can be shared in the compression. The Plandowski-Lifshits equality test of nonterminals can be generalized and requires polynomial time ([BLM05,SS05]).

A naive generalization of the pattern match is to find a compressed ground term in another compressed ground term, which can be solved by translating this problem into a pattern match of compressed preorder traversals of the terms. The interesting generalization of the pattern match is the following submatching problem: given two (compressed) terms s, t, where s may contain variables, is there an occurrence of an instance of s in t? A special case is matching, where the question is whether there is a substitution σ, such that $\sigma(s) = t$, which is shown to be polynomial in [GGSS08,GGSS10], and also computes the (unique) compressed substitution. Other related work are [Com95,Sal92] on term schematizations that investigate a form of compression as well as representing infinite sets of terms, and related algorithms.

In this paper we describe algorithms for answering the submatching question, and which only operate on the grammars. We show that if s is ground and compressed or if s is uncompressed, then the problem can be solved in polynomial time (Theorem 3.3 and Theorem 3.8). In the general case, we describe an algorithm that runs in time $O(n^{c|\,Var(s)|})$ (Theorem 5.21), which is exponential, but in a well-behaved parameter: If the number of variables is bounded by k, then the match algorithm runs in polynomial time. As a subproblem it is shown that there is a polynomial-time algorithm to find all occurrences of a compressed ground context occurring in a compressed term (Theorem 5.20). As an application and an easy consequence of the submatching algorithms, a single (parallel) deduction

step on compressed terms by a compressed axiom with at most k variables can be performed in polynomial time. We also show that a sequence of n rewrites can be performed in polynomial time, where the term rewriting system as well as the to-be-reduced term are compressed by STG. and where we assume that the term rewriting system is assumed to be constant (see Theorem 6.4). Investigations on complexity of deduction sequences under DAG-compression are in [AM10]. Another application is a querying mechanism for compressed XML-data bases ([LMM10]).
An introductory example illustrating the compression and equational deduction is as follows.

Example 1.1. As an example, consider the term rewriting rule $f(x) \to g(x, b)$, and let the term $t_1 = f(f(f(a)))$ be compressed as $C_1 ::= f(\cdot)$, $C_2 = C_1C_1$, $T ::= C_2(T'), T' = f(a)$. A single term rewriting step on the compressed term t_1 by the rule $f(x) \to g(x, a)$ would produce $T' ::= g(a, b)$, and hence the reduced and decompressed term is $f(f(g(a, b)))$. Other rewriting steps on the compressed term that do not decompress the term have to analyze the contexts. Let another term be $t_2 = f^{16}(a)$, compressed as $C_1 ::= f(\cdot)$, $C_2 = C_1C_1$, $C_3 = C_2C_2$, $C_4 = C_3C_3$, $C_5 = C_4C_4$, $T ::= C_5(a)$. A term rewriting step on T using $f(x) \to g(x, b)$ may rewrite the context $f(\cdot)$ and thus would produce $C_1 ::= g(\cdot, b)$, and hence reduces the term in one blow to $g(\ldots, (g(\ldots, b)\ldots), b)$. In fact this is a form of a parallel rewriting step.

The structure of the paper is as follows. First the basic notions, in particular STGs are introduced. Then algorithms for several special cases of the term submatching problem are given. The main part is a general algorithm for term submatching of compressed pattern and term, where it is shown that it can be performed in polynomial time for a fixed number of variables. Finally, we illustrate the application in equational deduction and term rewriting.

2 Preliminaries

We assume some knowledge in signatures, terms, positions, contexts and substitutions (see e.g. [BN98]). The set of free variables in a term t is denoted as $FV(t)$. Terms without occurrences of variables are called *ground*. A *substitution* σ is a mapping on variables, extended homomorphically to terms by $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$. Let *holep*$(c)$ be the position (as a string of numbers) of a hole in a context c. A prefix context of a context c is defined as a context c_1, such that $c = c_1(c_2)$ for some other context c_2.

Definition 2.1. *A term rewriting system (TRS) R is a finite set of pairs $\{(l_i, r_i) \mid i = 1, \ldots, n\}$, usually written $\{l_i \to r_i\}$, where we assume that for all $i : l_i$ is not a variable, and $FV(r_i) \subseteq FV(l_i)$.*
A term rewriting step is $t \xrightarrow{R} t'$, if for some i: $t = t_1[\sigma(l_i)]$ and $t' = t_1[\sigma(r_i)]$.
This can also be seen as an equational deduction step, where the rules in R are seen as equational axioms.

2.1 Tree grammars for compressions

Definition 2.2. *A* singleton context-free grammar (SCFG) *G, also called straight-line program (SLP) is a 3-tuple $\langle \mathcal{N}, \Sigma, R \rangle$, where $\mathcal{N}$ is a set of non-terminals, Σ is a set of symbols (a signature), and R is a set of rules of the form $N \to \alpha$ where $N \in \mathcal{N}$ and $\alpha \in (\mathcal{N} \cup \Sigma)^*$. The sets $\mathcal{N}$ and Σ must be disjoint, each non-terminal X appears as a left-hand side of exactly one rule of R, and $>_G$ on $\mathcal{N}$ is defined as $X >_G Y$ iff $X \to A \in R$ and Y occurs in A. The transitive closure $>_G^+$ must be irreflexive; i.e. there are no $>_G$-cycles. The word generated by a non-terminal N of G, denoted by $val_G(N)$ or $val(N)$ when G is clear from the context, is the word in Σ^* reached from N by successive applications of the rules of G.*

Definition 2.3. *A* singleton tree grammar (STG) *is a 4-tuple $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$, where $\mathcal{TN}$ are tree/term non-terminals of arity 0, $\mathcal{CN}$ are context non-terminals of arity 1, and Σ is a signature of function symbols (the terminals), such that the sets $\mathcal{TN}$, $\mathcal{CN}$, and Σ are pairwise disjoint. The set of non-terminals $\mathcal{N}$ is defined as $\mathcal{N} = \mathcal{TN} \cup \mathcal{CN}$. The rules in R must be of the form:*

- *$A \to \alpha(A_1, \ldots, A_m)$, where $A, A_i \in \mathcal{TN}$, and $\alpha \in \Sigma$ is an m-ary terminal symbol.*
- *$A \to C_1 A_2$ where $A, A_2 \in \mathcal{TN}$, and $C_1 \in \mathcal{CN}$.*
- *$C \to [\cdot]$ where $C \in \mathcal{CN}$.*
- *$C \to C_1 C_2$, where $C, C_1, C_2 \in \mathcal{CN}$.*
- *$C \to \alpha(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_m)$, where $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m \in \mathcal{TN}$, $C \in \mathcal{CN}$, and $\alpha \in \Sigma$ is an m-ary terminal symbol.*
- *$A \to A_1$, (λ-rule) where A and A_1 are term non-terminals.*

Let $N_1 >_G N_2$ for two non-terminals N_1, N_2, iff $(N_1 \to t)$, and N_2 occurs in t. The STG must be non-cyclic, i.e. the transitive closure $>_G^+$ must be terminating. Furthermore, for every non-terminal N of G there is exactly one rule having N as left-hand side. Sometimes we refer to the right-hand side of this rule as the definition *of N in G. Given a term t with occurrences of non-terminals, the derivation of t by G is an exhaustive iterated replacement of the non-terminals by the corresponding right-hand sides. The result is denoted as $val_G(t)$. We will write $val(t)$ when G is clear from the context. In the case of a non-terminal N of G, we also say that N (or G)* generates *$val_G(N)$ or* compresses *$val_G(N)$. The depth of a nonterminal N is the maximal number of $>_G$-steps starting from N, and the depth of G is the maximal depth of all its nonterminals. Let V be a set of nonterminals with λ-rule, then the $Vdepth(N, V)$ is the maximal number of $>_G$-steps starting from N until an element of V or a leaf is reached. The size of an STG is the number of its rules, denoted as $|G|$.*

An application for SLPs are to represent compressed positions in compressed terms.

2.2 Grammar Extensions

We list the main grammar extensions required in this paper and give also their size estimations.

Definition 2.4 (Grammar Extension). *We say that an 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* $\mathcal{R}' \supseteq \mathcal{R}$*.*

We repeat the constructions and their properties (see [LSSV10]).

Lemma 2.5. Term-Construction *Let* f *be an* n*-ary function symbol* f *in its signature and defining* n *terms* $t_1, \ldots t_n$*. Then there exists a grammar extension* $G' \supseteq G$ *defining the context* $f(t_1, \ldots, [\cdot], \ldots, t_{n-1})$ *and also a grammar extension defining the term* $f(t_1, \ldots, t_n)$ *and satisfies*

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

Concatenation *Let the contexts* $c_1, \ldots, c_n$ *for* $n \geq 1$ *be generated by* G*. Then there exists a grammar extension* $G' \supseteq G$ *that generates the context* $c_1 \cdot \cdots \cdot c_n$ *and satisfies*

$$\begin{aligned} |G'| &\leq |G| + n - 1 \\ Vdepth(G', V) &\leq Vdepth(G, V) + \log n + 1 \end{aligned}$$

Exponentiation *Let the context* c *be generated by* G*. For any* $n \geq 1$*, there exists a grammar extension* $G' \supseteq G$ *that generates the context* c^n *and satisfies*

$$\begin{aligned} |G'| &\leq |G| + 2 \log n \\ Vdepth(G', V) &\leq Vdepth(G, V) + \log n + 1 \end{aligned}$$

Prefix and Suffix *Let the context* c *be generated by* G*. For any nontrivial prefix or suffix* c' *of the context* c*, there exists a grammar extension* $G' \supseteq G$ *that defines* c'*, and satisfies*

$$\begin{aligned} |G'| &\leq |G| + depth(G) - 1 \\ Vdepth(G', V) &\leq Vdepth(G, V) + \log(depth(G)) + 1 \end{aligned}$$

Subterm *Let the context* c *or term* t *be generated by* G*. 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*

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

Subcontext *Let the term* t *be generated by* G*. For any nontrivial prefix context* c *of the term* t*, there exists a grammar extension* $G' \supseteq G$ *that generates* c *and satisfies*

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

Instantiation *Let the term t be generated by G, and let $x \in V$ be a terminal and let A be a nonterminal. Then the grammar extension $G' \supseteq G$ with the additional rule $x \mapsto N$ satisfies*

$$\begin{array}{ll} |G'| & \leq |G| + 1 \\ Vdepth(G') & = Vdepth(G) \end{array}$$

Lemma 2.6. *For an STG, we have $depth(G) \leq (Vdepth(G, V) + 1) \cdot (|V| + 1)$.*

Definition 2.7 (Grammar Extension Step). *We say that the pair (G', V') is constructed from the pair (G, V) using an* α-bounded grammar extension step *if it can be constructed by term-construction, concatenation, exponentiation, prefix, suffix, subterm, subcontext, or instantiation, where the exponent used for exponentiation is bounded by 2^α, and the number of concatenated contexts is bounded by α.*

Lemma 2.8. *If G' is an α-bounded extension of G according to Definition 2.7 then the following inequations hold:*

$$\begin{array}{ll} |G'| & \leq |G| + \mathcal{O}(depth(G)^2) + \mathcal{O}(\alpha) \\ Vdepth(G') & \leq Vdepth(G) + \mathcal{O}(\log(depth(G))) + \mathcal{O}(\log \alpha) \end{array}$$

In [LSSV10] it is shown, that a polynomial number of grammar extension as above under certain restrictions for concatenation and exponentiation leads to a polynomial-sized grammar:

Theorem 2.9. *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*

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

The extensions steps above can be performed in polynomial time, where proofs can be found in [GGSS10], and missing ones can be easily derived from these proofs.

3 Term Submatching

Given two first-order terms s, t, where s (the pattern) may contain variables, the submatching problem is to identify an instance of s as a subterm of t. The most general variant that we will consider in this paper is when s, t are compressed by an STG. This section contains several polynomial algorithms for special cases of the term submatching problem.

3.1 Term Submatching – First Observations

Algorithms for term matching and term submatching of compressed terms are useful for locating positions where a term rewriting step may occur, depending on whether the term that has to be rewritten is compressed and/or the term rewriting system, and also for performing a single rewrite step or a sequence of rewrite steps.
In deduction systems using equational deduction, like term rewriting system, the rewriting process may generate very large terms in a few steps, thereby slowing down the memory access as well as further deduction steps. It is standard to use dag representation as optimization, which is a form of compression. Here we investigate the more general case that an STG is used to represent terms. We will show that this representation can also be used to perform equational deduction.

Definition 3.1. *The* term submatching problem *is:*
Given a term s which may contain first-order variables, and a (ground) term t, both compressed with an STG G, i.e., $val(T) = t$ and $val(S) = s$ for term nonterminals S, T of G. The task is to compute a (compressed) substitution σ such that $\sigma(s)$ is a subterm of t; also the (compressed) position (all positions) p of the match in t should be computed.
We consider several specializations of the term submatching problem, depending on the properties and the representation of s, where we also treat combinations of the properties:

uncompressed *If s is given as a plain term without any compression.*
compressed *If s is represented by a term nonterminal S with $val(S) = s$.*
ground *If s is ground.*
linear *If s is a linear term, i.e. every variable occurs at most once in s.* □

Note that (compressed) term matching is a special case: it asks whether $\sigma(s) = t$. We are interested in efficient algorithms for submatching. We will prove that the general case can be performed in time $\mathcal{O}(n^{c \cdot k})$ where c is a constant, and k is the number of different variables in s. Thus the general case has an exponential worst-case upper bound for the required time, but the number of variables k is a well-behaved parameter, since it often can be assumed as fixed for the rewriting process. This upper bound already implies that the problem can be performed in polynomial time for special cases like ground terms and terms containing occurrences only of a single variable. Nevertheless, we will describe algorithms for several specializations, since these are easier to describe and implement than the general algorithm, and permit more flexibility for optimizations.

3.2 Ground term submatching

The investigation in [GGSS08] shows that (exact) term matching, also in the fully compressed version and also the computation of a compressed substitution, is polynomial. I.e. given two nonterminals S, T, where S may contain first-order variables, there is a polynomial time algorithm for answering the question

whether there is some substitution σ such that $\sigma(val(S)) = val_G(val(T))$, and also for computing the substitution, where the representation is as a list of variable-nonterminal pairs, where the nonterminals are w.r.t. an extension of the input STG.

Algorithm 3.2 (ground compressed term submatching). The special case of submatching where s is ground and compressed by a nonterminal S can be solved in polynomial time by translating both compressed terms into their compressed preorder traversals (i.e. strings) [BLM05,BLM08] and then applying string pattern matching (see [Ryt04,Lif07] for further references on the subject). The translation efficiently computes an SLP for the preorder traversal of $val(S)$ and $val(T)$ and asks whether one is a substring of the other (see [GGSS10] for more information on the preorder traversal). The string match algorithm in [Lif07] computes a polynomial representation of all occurrences. Note that in our case, the structure of ground terms is very special as a string matching problem. Thus the complete output of the algorithm is a follows: (i) a list of term nonterminals N of the input STG G, where $val(\sigma(S)) = val(N)$, and (ii) a list of pairs (N, p), where the rule for N is of the form $N ::= C[N']$, p is a compressed position, and $val(C)_{|val(p)}[val(N')] = val(S)$. Moreover, every nonterminal N appears at most once in the list.

There are efficient algorithms for the compressed string pattern match [Lif07]. The required time is $O(n^2m)$ where n is the size of SLP of T and m is the size of the SLP of S (the pattern). Since the preorder traversal can be computed in linear time (see [GGSS10]), we have:

Theorem 3.3. *The ground compressed term submatching can be computed in time $O(|G_T|^2|G_S|)$, and the output is a list of linear size.*

Note that there may be exponentially many matching positions, even if the output list has only single element N, since N may occur at an exponential number of positions.

3.3 Uncompressed Linear Submatching

We only give a sketch for this case, since it will be solved in a more general case in the next subsection.
Given an uncompressed and linear term s (i.e. every variable occurs at most once in s) and an STG G together with a nonterminal T, the following algorithm solves the submatching decision problem: Construct a non-deterministic tree automaton T_s that recognizes whether s is a subterm of another term ignoring the variables. This automaton can be easily constructed and has a linear number of states, where T_s has an accepting run on $val(T)$ if and only if s is a subterm of $val(T)$. It is known that acceptance of a compressed term by NTAs can be decided in polynomial time in the size of the STG and the automaton (see [LMSS09b,LM05]).
This method does not answer further questions like: What is the substitution, all matching positions or matching positions according to further constraints.

3.4 Uncompressed Submatching

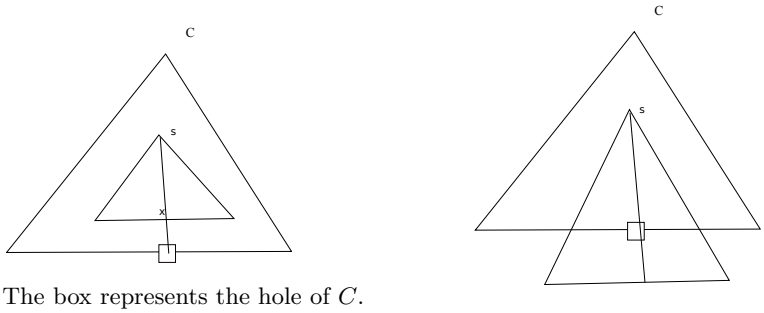
Now we look for the case of uncompressed s, but where variables may occur several times in s. We show that also in this case, there is an algorithm that requires polynomial time. The algorithm outputs enough information to determine all the positions of a submatch. We derive the following by an easy case analysis:

Lemma 3.4. *Given an STG G, a term s and a nonterminal T, the start symbol, with $val_G(T) = t$, where t is ground. If there is some substitution σ, such that $\sigma(s)$ is a subterm of t, then there are the following possibilities:*

1. *There is a term nonterminal B of G such that $val_G(B) = \sigma(s)$.*
2. *There is a rule $B ::= C[B']$ in G, such that $\sigma(s) = c[val_G(B')]$, where c is a suffix context of $val_G(C)$. There are subcases for the hole position p of c.*
   (a) *(overlap case) p is a position in s.*
   (b) *$p = p_1p_2$, where p_1 is the maximal prefix of p that is also a position in s. Then $s_{|p_1} = x$ is a variable. The algorithm has to distinguish further subcases*
      i. *(subterm case) x occurs more than once in s.*
      ii. *(subcontext case) x occurs exactly once in s.*

Example 3.5. The number of possible substitutions for a submatch may be exponential: Let the rules be $S ::= f(x)$, and $T ::= C_n[a], C_0 ::= f([.]), C_1 ::= C_0C_0, \ldots C_i ::= C_{i-1}C_{i-1}$. Then $val(T) = f^{2^n}(a)$, and every substitution $\sigma(x) = f^i(a)$ with $0 \leq i \leq 2^n - 1$ corresponds to a submatch. However, distinguishing the subterm (Lemma 3.4, case 2a and subcontext case (Lemma 3.4,2b), we see that the exponentially many substitutions correspond to one substitution for the subcontext case.

We will see that this also holds for the uncompressed submatch.



The box represents the hole of C.

Fig. 1. Possible relative positions of C and s

Algorithm 3.6 (uncompressed submatching). The following task is solved: Given an STG G, a term nonterminal T of G, and a term s. Find one (or all) possibilities of a substitution σ and a position p such that $\sigma(s)$ occurs in $val(T)$ at position p. We assume that the substitutions and the positions are represented in a compressed form.
There are the following kinds of output according to the cases in Lemma 3.4: (i) a nonterminal N such that s matches N (case (1)), (ii) a nonterminal N with rule $N = C[B]$, such that s overlaps C and B (case (2a)), (iii) a substitution ρ such that $\rho(s)$ is ground, where the positions can be determined using Algorithm 3.2 (case (2b) with multiple occurrences of the variable x) (iv) a context nonterminal C, a substitution ρ, and a compressed position of an occurrence of s in C, where $\sigma(s)$ occurs in $val(C)$, and a variable x with a single occurrence in s where x matches a superterm of the hole (case (2b)).

s-in-C-table For every context nonterminal C and for every position p of s there is at most one entry ρ in the s-in-C-table, representing the possibilities where s starts in $val(C)$ (see Fig. 1). The substitution ρ instantiates certain variables of s with ground terms. We assume that ρ is represented as a list with entries of the form $x \mapsto D_x$, where D_x are nonterminals in an extension of G. The semantics is that $\rho(s)$ has an overlap with $val(C)$ at some position q in $val(C)$, and p is a position in s, such that $q.p$ is the hole position of $val(C)$.
We use dynamic programming for a bottom-up (w.r.t. G) computation of the s-in-C-table. In a precomputation several attributes and further information can be computed like an SLP for the the position of the holes in $val(C)$ for every context nonterminal C.

- The rule is $C ::= f(A_1, \ldots, A_{j-1}, [\cdot], A_{j+1}, \ldots A_n)$. Then compute a compressed substitution ρ, such that for $s = f(s_1, \ldots, s_n)$, $val(\rho(s_i)) = val(A_i)$ for all $i = 1, \ldots, n$, where $i \neq j$. If there is no such ρ, then there is nothing to do. Let ρ be the computed substitution. If $s_j = x$ and x occurs only once in s, then there is no table entry, and we are in case (2(b)ii) of Lemma 3.4. The output will be (C, ρ) and the top position ε. Otherwise, there will be a table entry ρ.
- The rule is $C ::= C_1 C_2$, and an entry in the table is to be computed from the entries in the tables for C_1, C_2. There are several cases:
  1. One case is that there is an entry in the table C_2. This entry is simply inherited.
  2. The other case is that there is an entry ρ for (C_1, p) in the s-in-C-table. Then we have to match $s_{|p}$ against C_2.
     (a) If the hole position h of C_2 is a position in $s_{|p}$, then we compute a compressed substitution ρ' in an extension of G, such that every variable x in $s_{|p}$ is instantiated, with the possible exception of variables for which $p.h$ is a prefix of their position. If this is not possible, then do nothing. If ρ' and ρ are compatible, then the combined ρ'' is the new entry for (C, p) Otherwise, there is no entry.

(b) If the hole position h of C_2 is not a position in $s_{|p}$, then let q be the maximal prefix of $p.h$ that is also a position in s. Compute ρ' for the match of $s_{|p}$ with C_2 taking into account all variable positions p' with $pp' \neq q$ in s. If ρ' does not exist, or if it is computable and not compatible with ρ, then do nothing. Let ρ'' be the combination of ρ and ρ'. Clearly, $s_{|q}$ is a variable. If $\rho(s)_{|q} = s_{|q}$ is a variable, then this variable has only one occurrence in s, and we are in case (2(b)ii) of Lemma 3.4. It is sufficient to have C, p, ρ in the output.
If $\rho(s)_{|q}$ is not a variable, then it is a ground term, and also $\rho(s)$ is a ground term, since all variables of s are instantiated. We are in case (2(b)i) of Lemma 3.4. The substitution ρ will be outputted, and the possible positions be computed using the algorithm for matching ground compressed terms in Algorithm 3.2.

Now there are two further steps for detecting and locating matches:

1. For all term nonterminals N, check whether there is a substitution σ, such that $\sigma(s) = val(N)$. If this is true, then a match is found.
2. For all nonterminals N with rule $N ::= C[B]$, and every position p of s, compute a compressed match ρ' for $s_{|p}$ with B. If ρ' and the s-in-C-table entry ρ for p are compatible, then a match is found.

Proposition 3.7. *The algorithm 3.6 for uncompressed pattern matching has polynomial running time in $|s|$ and the size of G. Moreover, a polynomial sized representation of all matching possibilities can be computed.*
The number of necessary matching substitutions is polynomial, if the subcontext case i.e. case (2(b)ii) *of Lemma 3.4 is represented as a partial substitution, as done in algorithm 3.6.*

Proof. The table is of at most quadratic size and the entries are at most linear. However, the nonterminals used in the substitutions ρ have to be constructed during the construction. This constructions can be done independently of each other. Hence the size of the table, and of the output is polynomial. The output represents all the substitutions and positions. The number of substitutions is therefore polynomial.

Note that the number of positions may be exponential, since for example a matching nonterminal N may have an exponential number of occurrences in the input term $val(T)$, but they can be represented in polynomial space. Also the number of substitutions may be exponential, if plainly all ρ are requested such that $\rho(s)$ is a subterm of t.

Theorem 3.8. *The uncompressed submatch computation problem can be solved in polynomial time. Also an explicit polynomial representation of all matching possibilities can be computed in polynomial time.*

4 Compressed Pattern Match for Terms

We consider the compressed pattern match problem for terms.
Given an STG G, two term nonterminals S, T, where $val(S)$ may contain variables, compute an extension G' of G and a compressed substitution σ such that $\sigma(val_{G'}(S))$ is a subterm of $val_G(T)$. Also a representation of the position(s) of the match in $val(T)$ have to be computed.

Algorithm 4.1. The (nondeterministic) algorithm for matching S against a subterm of $val(T)$ proceeds in several steps: The algorithm consists of an iteration that in every step (non-deterministically) generates instantiations for at least one variable of $val(S)$: In a single iteration step there are three cases:

1. If $s := val(S)$ contains more than two occurrences of variables, then do the following (nondeterministically):
   Construct an extension G' of G and nonterminals S', C, A such that $S' ::= C[A]$, $val(S') = s$, $val(C)$ is a ground context, $val(A) = f(r_1, \ldots, r_n)$, $n \geq 2$ and at least two r_i contain occurrences of variables. (The algorithm for this prefix computation is in Subsection 4.1). Then (nondeterministically) select a right hand side $f(B_1, \ldots, B_n)$ of a rule that contributes to T. The first case is that $val(f(B_1, \ldots, B_n))$ is a term. Then compute σ as matching A against $f(B_1, \ldots, B_n)$: This means to construct an extension G'' of G' such that $val_{G''}(A) = val_G(f(B_1, \ldots, B_n))$. Now $val_{G''}(S')$ is ground and we can use another item in the next iteration to check the matching.
   The second case is that $f(B_1, \ldots, B_n)$ is a context. Then compute σ as matching A against $f(B_1, \ldots, B_{i-1}, [.], B_{i+1}, \ldots, B_n)$, where we ignore the index i, which is the hole position. This means to construct an extension G'' of G', such that A can be replaced by $f(A_1, \ldots, A_n)$ and to match A_j against B_j for all $j \neq i$. The extension is such that $val_{G''}(A_j) = val_G(B_j)$. Now $val_{G''}(S')$ contains less variables than s and we can go to the next iteration.
2. If s is ground, then use compressed string matching. The position is an index of the string pattern of s in the preorder traversal of t, which can easily and efficiently be translated into compressed position p (see Algorithm 3.2).
3. If s contains exactly one occurrence of a variable (say x), then another sub-algorithm is required (see Section 5). First construct a fresh context nonterminal C, such that $val(S) = val(C[x])$. Note that C represents a ground context. The algorithm in Section 5 uses dynamic programming to compute a table such that for every context- and term nonterminal D that contributes to T a (polynomial) representation of the occurrences of C in D is stored.

4.1 Computing a Ground Prefix Context

We consider the subtask to split $s = val_G(S)$ into $D[S']$, such that C is the maximal prefix context that is ground. This can be done along the structure of

the STG G as follows, where the algorithm extends the grammar. We start with the rule for S, and two lists L_p, L_t: one is the list for context-parts of the prefix context and the other is required if the algorithm looks for contexts and has to remember a list for context-parts and a term part for constructing the term in the hole. Initially, these lists are empty

- $A ::= C[A']$. If C does not contain variables, then proceed with A' and $L'_p := L_p; C$. If C contains variables, then proceed with C and $L_t := [A']$.
- If $A ::= f(A_1, \ldots, A_n)$ then check which of the terms $val_G(A_i)$ contain variables. If there is only one, say A_i, then proceed with A_i, and $L'_p := L_p, f(A_1, \ldots, [.], \ldots, A_n)$. Otherwise, the algorithm stops and constructs D from the list L_p as a balanced tree.
- If $C = C_1C_2$, then there are two cases: If $val(C_1)$ does not contain variables, then proceed with C_2, and $L'_p := L_p; C_1$. Otherwise, if $val(C_1)$ contains variables, then proceed with C_1 and $L'_t := C_2; L_t$.
- If $C = f(A_1, \ldots, [\cdot]_j, \ldots, A_n)$, then there are two cases: If more than two of the terms $val(A_i)$ contain variables or one $val(A_i)$ contains variables as well as some expanded term or context in L_t, then we are finished. We construct D from the list L_p as a balanced tree, and the term nonterminal from the list $L_t = B_1; \ldots; B_m; A$ also as a balanced tree such that it is equivalent to $B_1 \ldots B_m[A]$).
  If only one $val(A_i)$ contains variables and all expanded terms or contexts in L_t are ground, then let $k \neq j$ be this index. Now we construct A_j from L_t as a balanced tree. Then we proceed with A_k, $L_t := []$, and $L'_p := L_p; f(A_1, \ldots, [\cdot]_k, \ldots, A_n)$.

5 Matching a Context as a Subcontext

This section contains a description of an algorithm for a subproblem of the fully compressed pattern matching problem for terms: checking whether a compressed ground context appears in a compressed term. The problem could be called the *fully compressed context-in-term problem.*

> Given an STG G, a context nonterminal C, and a term nonterminal A, check whether $val_G(C)$ occurs in $val_G(A)$.

The goal is to describe a polynomial algorithm for this problem
The algorithm will rely on the efficient algorithm for the fully compressed pattern matching problem for strings. We only have to take care of some special cases that are not covered by a translation into strings. It will also use knowledge on periodicity of strings.

5.1 Overlappings of Contexts

First we have to clarify the possibilities of overlapping two ground contexts. There are two different variants of the question:

(i) Given a ground context c, we ask for an occurrence of another ground context d or perhaps parts of d in c.
(ii) Given a ground term t and two ground contexts c, d, find the occurrences and relative positions of c, d in t.

Note that there are subtle differences between these two scenarios, since the overlap question is relative to the given context or term. Though (ii) is finally required, we investigate the case (i) since it answers the question also for all embeddings in terms t and permits a polynomial representation of all possibilities. Similar considerations are in [SSS02], but only for the second case: overlapping contexts embedded in terms.

Definition 5.1 (overlap and occurrence of contexts). *We assume given a (ground) context c and another context d and a position p within c. If there is a term r, and a position p within c that is a prefix of $holep(c)$ but $p.holep(d)$ is not a prefix of $holep(c)$ and there is an occurrence of d in $c[r]$ starting at p, then we say there is an* overlap *of d in c at p. The overlap is called* completely within *c, if p is not a prefix of $holep(c)$. The overlap is* aligned *if p is a prefix of $holep(c)$ and $holep(c)$ is a prefix of $p.holep(d)$. The overlap is called* partially aligned *if p is a prefix of $holep(c)$ and $holep(c)$ is not a prefix of $p.holep(d)$.*
The case that d occurs in c at a position p that is not a prefix of $holep(c)$ is called an side area occurrence*, and the case that c at a position p that is a prefix of $holep(c)$ and $p.holep(d)$ is a prefix of $holep(c)$ is called an* aligned occurrence*.*

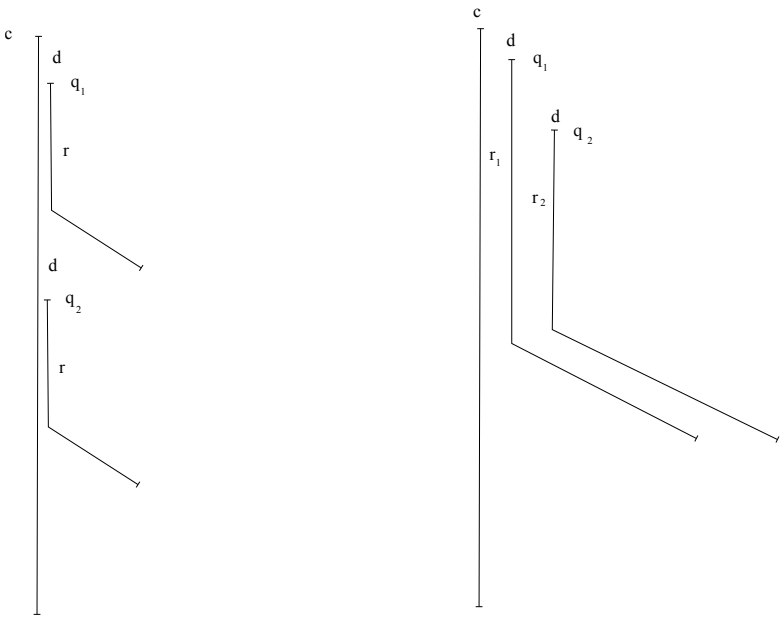
Lemma 5.2. *Let c be a context. Then every overlap of c with itself is aligned.*

Proof. Let $p \neq \varepsilon$ be the overlap-position of c. It is easy to see that p must be a prefix of $holep(c)$, since otherwise c is properly contained within itself, which is impossible. Assume that $holep(c)$ is not a prefix of $p.holep(c)$. Then by induction on n we see that for every n, the position p^n is a position in c, which is not possible since contexts are finite. Hence every occurrence of c in c is aligned.

Definition 5.3. *Let c be a context. Let d be another context such that there are 2 overlaps of d in c at the occurrences $q_1 \neq q_2$, and assume the overlaps are partially aligned. We distinguish two possibilities (see figure 2):*

- *The occurrences q_1, q_2 are called* parallel *partially aligned occurrences of d in c, iff there is a path r that is a prefix of $holep(d)$ such that $q_i.r$ is the maximal common prefix of $holep(c)$ and $q_i.holep(d)$ for $i = 1, 2$.*
- *The occurrences q_1, q_2 are called* sequential *partially aligned occurrences, if the following holds: there are two positions r_1, r_2 in d which are both prefixes of $holep(d)$, with $q_1.r_1 = q_2.r_2$ and such that $q_1.r_1$ is the maximal common prefix of $q_1.holep(d)$ and $holep(c)$, and $q_1.r_1$ is also the maximal common prefix of $q_2.holep(d)$ and $holep(c)$. Moreover, $q_1.holep(d)$ is a prefix of $q_2.holep(d)$.*

Lemma 5.4. *Let c be a context. Let d be another context and $q_1 \neq q_2$ be positions, such that q_1 is a prefix of q_2 and d is partially aligned at q_1 and q_2. Then the occurrences are either parallel or sequential according to Definition 5.3.*



parallel partial overlap sequential partial overlap

Fig. 2. Two possibilities of overlap

Proof. For $i = 1, 2$ let p_i be the maximal common prefix of $holep(c)$ and $q_i.holep(d)$.
First assume that $p_1 \neq p_2$. Let r_i be such that $q_i.r_i = p_i$. We have to show that $r_1 = r_2$. If r_1 is a proper prefix of r_2. let a be a path such that $q_1.r_1.a = q_2.r_1$. It is easy to see that r_1a^n is a position in d for every n, which is impossible. If r_2 is a proper prefix of r_1, let a be a path such that $q_1.r_1.a = q_2.r_2$. Again it is easy to see that r_1a^n is a position in d for every n, which is impossible. The only case left is that $r_1 = r_2$. If the embedding is in $c[t]$, then this is impossible, since then the term at position $q_1.r_1$ contains itself as a proper subterm.
Now assume that $p_1 = p_2$. Let $p := p_1$ and r_i be such that $q_ir_i = p$. Assume that $q_1.holep(d)$ is not a prefix of $q_2.holep(d)$. Let b be the path such that $q_1.b = q_2$ and a be such that $q_1.r_1.a$ is the maximal common prefix of $q_1holep(d)$ and $q_2.holep(d)$. Then the position $r_1.a.b^n$ is within d for every n, which is impossible. Hence $q_1.holep(d)$ is a prefix of $q_2.holep(d)$.

Example 5.5. There are examples for the parallel and sequential case of partial overlaps:

1. Let $c = f(f(f([.], a), a), a)$ and let $d = f(f(f(a, a), a), [.])$. Then d has parallel partially aligned overlaps at positions ε, 1 and 1.1. Note that if we in c plug in a term t at position 1.1.1, then at most one these overlaps of d in $c[t]$ is possible at the same positions.

2. We give an example for case 2 of Lemma 5.4, i.e. for sequential partially aligned overlaps. The partially aligned occurrences of d within c are not unique: Let $t = f(a, f(a, f(a, f(a, f(a, b)))))$, d be the context $f(a, f(a, f(a, f([.], f(a.b)))))$ and let $c = f(a, f(a, f(a, [.])))$, and we look for occurrences of c in $d[a]$. Then there are two partially aligned occurrences of c in d: at position 2 and at position 2.2 of t. Note that the two occurrences of c are aligned.

Corollary 5.6. *Let c be a context and let q_1, q_2 be two different occurrences of d that are parallel partial overlaps. For every term t, at most one occurrence q_1, q_2 can be an occurrence of d in $c[t]$.*

Proof. Otherwise, Lemma 5.2 would imply that the occurrences of d are aligned, which is impossible.

Now we determine the possible relative positions of several partial overlaps of a context d with a context c.

Definition 5.7. *Let c, d be contexts. Then a* periodic *sequence of partially aligned overlaps of d in c is a sequence of positions $q_1, \ldots, q_n$ for $n \geq 2$ of partially aligned overlaps of d in c, such that $q_i = q_1.q^{i-1}$ for some q and $i = 1, \ldots, n$. The path q is the* period*, and n the length of the sequence. If the partially aligned occurrences are pairwise parallel, then we say it is a parallel sequence, and if all partially aligned occurrences are pairwise sequential, then we call it a sequential sequence. Note that there are no other possibilities due to Lemma 5.4.*

Example 5.8. This example shows that there may be three partially aligned occurrences of d within c that do not form a periodic sequence. Let $c = c_1^3 c_2 c_1^3$ where $c_i = f([\cdot], A_i)$, for $i = 1, 2$ and $d = f(c_1^2 c_2 c_1^3(A_3), [\cdot])$. Then there are (parallel) partial alignments of d in c: at the top, i.e. at position ε, at 1.1.1.1 and at 1.1.1.1.1. However, there is no partial alignment at position 1, since $A_1 \neq A_2$. The reason that there is no sequence is that the common part of the first two overlaps of d is not large enough (see also Lemma 5.11). Note that there is no term t, such that $c[t]$ contains all these occurrences of d.

The goal is now a polynomial representation of all the possible partial alignments that may have a potential embedding in a term.
We have to examine some special cases. First we examine parallel partially aligned overlaps and show some completeness properties.

5.2 Overlapping of Strings with Holes

The relative positions of partially aligned overlaps can be determined by investigating overlaps of strings that may have a single hole. We represent the hole by special symbol #. In overlaps, the symbol # holds the place of a single symbol. Also, # will not be compared with other symbols. Positions in strings s are

numbers $1, 2, 3, \ldots$ and we use $s[i]$ to denote a symbol at position i and $s[i, i']$ to denote the substring from position i to i' including the last one, The notation $s[i, j]$ is the empty string for $i > j$, and $s[i, i] = s[i]$. For example, the string $s_1 =$ "$babacabab$" overlaps with $s_2 = a\#ababd$" at position 3 in s_1, and s_2 is said to have the hole at position 2.

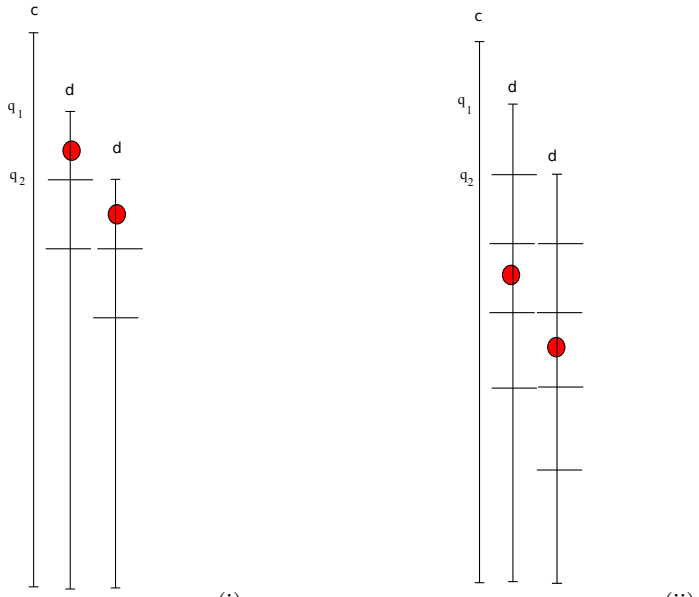


Fig. 3. Overlapping possibilities for two strings with a hole

Structure of Parallel Partially Aligned Overlaps We describe the situation for the parallel overlap in a general way as overlapping of several strings with or without a hole. In the case of a parallel partial overlap, the strings without the hole represent the context c, and the strings with holes are the context d, but along the path that is overlaid with the hole path of c; the string-hole is the position, where the hole paths of the c, d-contexts fork.

Lemma 5.9. *Let c be a string and let d be another string with a hole that has two overlaps with c at q_1, q_2, where $q_1 < q_2$ and $|d| \geq |c|$. Assume also that there are no further overlaps of d for $q_1 < q < q_2$. Let $p = q_2 - q_1$ and let p_1 be the position of the hole of d. Assume further that $|c| \geq q_2 + p$.*
Then there are two cases:

(i) There is a periodic sequence $q_1, \ldots, q_n$ of d-occurrences with $n \geq 2$ and $|c| - q_n < p$, and for every overlap of d with c at q with $q \notin \{q_1, \ldots, q_n\}$, we have $q - |c| < p$.

(ii) For every overlap of d with c where $q \notin \{q_1, q_2\}$ and $q > q_2$, we have $q - |c| < p$.

Proof. We look at two occurrences of d in c. There are several cases, depending on the position of the hole (see Fig. 3).

1. Let the hole position be $1 \leq p_1 < p$. Then the overlap generates the following equalities: $d[p+1, p+p_1-1] = d[1, p_1-1]$, and $d[p+p_1+1, 2p] = d[p_1+1, p]$, but there is no relation between $d[p+p_1]$ and $d[p_1]$. Furthermore there are the equalities $d[ip+p_1] = d[(i+1)p+p_1]$ for $i \geq 1$, provided $q_2+ip+p_1 \leq |c|$. This implies that $d[ip+1, (i+1)p] = [(i+1)p+1, (i+2)p]$, provided $q_2+(i+1)p \leq |c|$, and also $d[ip+1, k] = [(i+1)p+1, k]$ for the last partial period. This means that d is periodic after the hole with period p. Hence there is a periodic sequence as claimed in the lemma. If there is a further overlap q of d in c with $q \notin \{q_1, \ldots, q_m\}$, then there is an overlap of $s = d[p+1, 2p]$ within ss. Hence the period of d after the hole is strictly smaller than p. So we will find also a further overlap of d with c between q_1 and q_2, which is not possible. Note that there may be further overlaps q of d with c with $|c| - q < p$.
2. Let the hole position be $p_1' > p$. Let p_1, p_2 be such that $p = p_1 + 1 + p_2$ and $p_1 = p_1' \mod p$. Using a similar reasoning as above, it is easy to see that there are two cases:
   (i) d is periodic with period p (ignoring the hole), and there is a periodic sequence as in the previous item and the same reasoning for other overlaps as in the previous item is applicable.
   (ii) The equality reasoning is disconnected at the hole overlappings and c is long enough: i.e. $d[p_1' - p] \neq d[p_1' + p]$, and $q_2 + p_1' > |c|$.
   Then d is periodic with period p up to the positions $ip+p_1$, since the overlap is large enough. Let $a := d[p_1' - p]$ and $b := d[p_1' + p]$, where $a \neq b$. Then $d[ip + p_1] = a$ for all $i \geq 0$ such that $ip + p_1 < p_1'$ and $d[ip + p_1] = b$ for all i such that $ip + p_1 > p_1'$. This clarifies the periodic structure of d w.r.t. the period d and up to the index $|c| - q_1$, i.e., in the part of d at q_1 that overlaps c. It remains to show that there are no further long overlaps of d with c other than q_1, q_2: First suppose there is an overlap at a position of the form $q_1 + ip$ with $i > 1$. This is not possible, since there will be a conflict at position $d[p_1' + p]$, which is b for the occurrence of d at position q_1 and a for the occurrence of d starting at position $q_1 + ip$.
   Now suppose there is a further overlap of d with c at position q not at a position of the form $q_1 p^i$, with $q > q_2$ and with a significant overlap with the c, i.e., $q - |c| > p$. Then the first period of length p of d at q is contained either in d at q_1 or d at q_2 without covering a hole. Several case have to be considered separately:
   - If $q + p > q_2 + p_1'$, i.e. the first period of d at q_2 is before the hole, then $d[1, p_1']$ is periodic with a period that is a divisor of p. If the hole of d at q_2 is not within c, i.e. $q_1 + p_1' > |c|$, then there will be another overlap of d between q_1, q_2, which is impossible. Hence the hole of d at q_2 is within c, i.e. $q_1 + p_1' \leq |c|$. Looking at d at q_1, we obtain that $d[p_1' + p] = a$, which contradicts $a \neq b$.

– If $q + p \leq q_2 + p'_1$, i.e. the first period of d at q_2 may contain the hole or is after the hole, and it is completely after the hole of d at q_1. Then the $d[1, p]$ is periodic with a period that is a divisor of p. However, since the containment is completely with the b-part of d, this is impossible, since the multiset of symbols of $d[1, p]$ (the d at q) and $d[q-q_1+1, q-q_1+1+p]$ (the d at q_1) are different. Hence this is also impossible.

It is easy to see, based on the structural analysis in the previous proof, that the following holds:

Lemma 5.10. *In the situation of Lemma 5.9, if the periodic sequence consists of at least 3 overlaps, at positions q_1, q_2, q_3 with $q_2 - q_1 = q_3 - q_2 = p$ and $|c| \geq q_3 + p$, then the d has a regular structure: Replacing the hole of d by the corresponding symbol of c makes d periodic, where p is a period of d.*

Lemma 5.11. *Let c be a context and let d be a context that has two occurrences of parallel partially aligned overlaps with d at q_1, q_2, where $q_1 \preceq q_2$ and such that there are no further partially aligned overlaps at occurrences q with $q_1 \preceq q \preceq q_2$. Then there are two cases:*

(i) There is a periodic sequence $q_1, \ldots, q_n$ of d-occurrences that are partially aligned with $n \geq 2$ and $|holep(c)| - |q_n| < |q_2| - |q_1|$.
(ii) Every occurrence q of d starting properly below q_2 and which is a partially aligned overlap satisfies $|holep(c)| - |q| < |q_2| - |q_1|$.

In addition, if the periodic sequence has length at least 3, then there is a term r, such that the term $d[r]$ is the embedding of d in c, and along the hole path of c, this term is periodic with period p, where $q_1.p = q_2$.

Proof. It is already known (see Lemma 5.4) that two different parallel partially aligned overlaps have the same maximal common hole path with the context c. Let p be the maximal path such that $q_1.p$ is a prefix of $holep(c)$ and let r be the path such that $p.r = holep(d)$. Due to the overlap, the contexts d are periodic with period p up to the positions corresponding to the hole, i.e up to the positions $p^i.r$. We can apply Lemma 5.9 to the labels at $p^i.r$, and obtain that also all the occurrences of contexts are either periodic or that q_1, q_2 are the only occurrences, ignoring occurrences q of d that $|holep(c)| - |q| < |q_2| - |q_1|$. If the periodic sequence has length at least 3, then the claim follows from Lemma 5.10. □

Structure of Sequential Partially Aligned Overlaps Similarly as for the parallel case, we describe the situation for the sequential partial overlap in a general way as overlapping of one string with a hole with two usual strings. Here the interpretation as strings is as follows: the strings for d represent the contexts along their hole path, and the string for c is the context seen along the path used by the overlapping d-contexts. The string-hole is the position, where c and d fork. Note that Lemma 5.12 about strings does not talk about the structure of c below the forking position, which has to be deferred to Lemma 5.14.

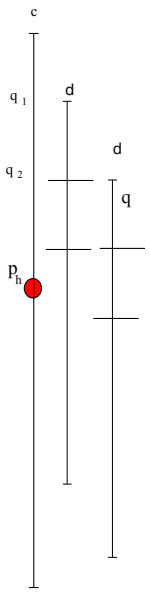


Fig. 4. Overlapping possibilities for two strings in a string with a hole

Lemma 5.12. *Let c be a string with a hole at position p_h, and let d be another string that has two overlaps with c at q_1, q_2, where $q_1 < q_2$ and both occurrences of d are completely within c and overlap the hole of c. Assume also that there are no further overlaps of d for $q_1 < q < q_2$. Let $p = q_2 - q_1$ and let p_h be the position of the hole of c. Also, the overlap should be sufficiently large, i.e. $|d| \geq 2|q_2 - q_1|$.*
Then the following holds:
There is a periodic sequence $q_1, \ldots, q_n$ of d-occurrences with $n \geq 2$, where the d-occurrences are completely within c, and the d-occurrences overlap the hole of c, such that there are no other overlaps of d with c that are completely within c and overlap the hole of c.

Proof. First analyze the structure of d. Let $p = q_1 - q_2$ be the period. Then let $p'_h = p_h - q_1 \mod p$. The structure of d is as follows: it consists of a repetition of k strings $d[1..p]$, followed by a prefix of the string b^j, where b is the same as $d[1..p]$ up to the position p'_h. The reason is that the hole of c disconnects the symbols of the lower and upper half of d.
Now assume that there is another occurrence q' of d completely within c and overlapping the hole of c. The preconditions show that if $q' - p_h \geq p$, and that then $b = d[1..p]$, and it must be synchronous with the period structure of d. A similar argument applies if $q' - p_h < p$, since in this case the overlap of the two occurrences below the hole is large enough to enforce $b = d[1..p]$.
We obtain that in the case of a third occurrence, all the d-occurrences that are completely within c and overlap the hole form a periodic sequence $q_1, \ldots, q_n$. But

note that the last occurrence, q_n, might be far from the hole, i.e. $q_n - p_h > p$ is possible.

It is easy to see that for more occurrences of d, we obtain periodicity:

Lemma 5.13. *In the situation of Lemma 5.12, if the periodic sequence consists of at least 3 overlaps at q_1, q_2, q_3, then d is periodic with period $p = q_2 - q_1$.*

Lemma 5.14. *Let c be a context and let d be a context that has two occurrences of sequentially partially aligned overlaps with d at q_1, q_2, where $q_1 \preceq q_2$ and such that there are no further partially aligned overlaps at occurrences q with $q_1 \preceq q \preceq q_2$. Then there is a periodic sequence $q_1, \ldots, q_n$ of d-occurrences that are sequentially partially aligned with $n \geq 2$ and there are no further sequentially partially aligned overlaps of d in c.*
Moreover, if $n \geq 3$, then the context d is periodic with period p, where $q_2 = q_1.p$. Also, the suffix context of c starting at the position where c and d fork is periodic with a period p', where $|p'| = |p|$.

Proof. This follows from Lemma 5.12, and the obtained structure of d. Note that d is completely contained in c in every case up to the position where the two hole paths fork, which corresponds to the hole in the string. Hence there can be no other partially aligned occurrences in c.

Structure of Fully Aligned Overlaps Now we look for fully aligned overlaps, which correspond to plain overlapping strings:

Lemma 5.15. *Let c be a context and let d be a context that has two occurrences of aligned overlaps with d at q_1, q_2, where $q_1 \preceq q_2$ and such that there are no further aligned overlaps at occurrences q with $q_1 \preceq q \preceq q_2$. Then the following holds:*

(i) There is a periodic sequence $q_1, \ldots, q_n$ of d-occurrences that are aligned with $n \geq 2$ and $|holep(c)| - |q_n| < |q_2| - |q_1|$.
(ii) Every other occurrence q of d starting properly below q_2 and which is a aligned overlap satisfies $|holep(c)| - |q| < |q_2| - |q_1|$.

Proof. This follows similarly as for overlapping strings. If there are other occurrences than mentioned in the claim of the lemma, then we would obtain a finer period of d, and hence another periodic sequence.

Definition 5.16. *Let G be an STG and let C, D be two context nonterminals of G. A representation of all overlaps (partially aligned and aligned) of $val(C)$ with $val(D)$ consists of three sequences, one for parallel partially aligned, one for sequential partially aligned and one for aligned overlaps, of the following components:*

1. *q, a position of an overlap.*

2. (q, P, n) *for a periodic sequence starting at a position* q *with period context nonterminal* P *and* n *occurrences. We assume that* $n \geq 3$.

The semantics is that $val(C)$ *has an overlap with* $val(D)$ *at position* $val(q)$ *in* $val(D)$*, and in case of a periodic sequence, also for all positions* $val(q)holep(val(P))^i$ *with* $0 \leq i < n$.

Now we can prove that there is a polynomial representation of all overlaps of a compressed context in another compressed context.

Proposition 5.17. *Let* C, D *be two context nonterminals of an STG* G*. Then there is a polynomial-sized representation of all overlaps (partially aligned and aligned) of* $val(D)$ *with* $val(C)$.

Proof. The lemmas above show that all the overlap positions of d in the upper half of c can be represented by one or two single entries or by a single entry for a periodic sequence. This covers already $|holep(c)|/2$ potential positions. t Since the length $|holep(c)|$ is bounded by $2^{|G|}$, the complete sequence has linear length in $|G|$. The required numbers can also be represented in linear space. There are at most three such sequences, hence the claim holds.

5.3 Tabling the Occurrences

We describe a dynamic programming algorithm for computing the occurrences of a context c in a term t under compression, which computes the representation in Definition 5.16, and also prints detected occurrences.
For a context nonterminal P and a natural number n we write P^n for the context nonterminal that represents the n-fold iteration of $val(P)$; and $prefix(P, k)$ for the context nonterminal representing the prefix of $val(P)$ of hole depth k. Given a number k, the rotation of P by k, denoted as $rot(P, k)$, is a new nonterminal P' in an extension of G, such that $val(P) = d_1 d_2$, $val(P') = d_2 d_1$, and $|holep(d_1)| = k$.
The following polynomial subalgorithms are used in the construction:

1. (Finding the maximal period parameters) Given a context nonterminal P, and a term (or context) nonterminal N, compute the maximal numbers m, k with $k < m$, such that $val(P)^m prefix(P, k)$ is a prefix context of $val(N)$.
   This can be performed by known grammar constructions, using equality check, and binary search.
2. (Rotating a context) Given a context nonterminal P, and a number k, compute the rotation $rot(P, k)$, i.e. construct the extension and the context nonterminals.

Algorithm 5.18 (Checking occurrences of contexts in contexts). The following dynamic programming algorithm computes a table indexed by (C, B, w), where C is a context nonterminal, B is a context nonterminal, w is one of the three texts "aligned", "parallel", "sequential", and the entry represents the overlaps of $val(C)$ in $val(B)$. A single entry in the table is a list consisting of single positions p and triples (p, P, n) for periodic sequences.
The computation of the entries is bottom-up on the second argument in the grammar as follows:

- If during the computation of the table entries below, it is detected, that a position or periodic sequence represents a complete containment of $val(C)$ in $val(B)$, then these occurrences are printed.
- If $B ::= f(B_1, \ldots, B_{i-1}, [.], B_{i+1}, \ldots, B_n)$ is a context nonterminal, then we test whether the context $val(C)$ overlaps (or occurs) at the top of $val(B)$. This can be done in the usual way by constructing term nonterminals such that $val(C) = val(f(C_1, \ldots, C_{k-1}, [.], C_{k+1}, \ldots, C_n))$ and testing whether $val(C_j) = val(B_j)$ for all $j \neq i$ and $j \neq k$. If $val(C) = val(B)$, this is printed, otherwise if the test succeeds and $k = i$, then ε is an entry in the aligned table, and if $k \neq i$, then an entry in each partial overlap list is ε.
- If the rule for B is $B ::= B_1B_2$, then we combine the table entries for (C, B_1) and (C, B_2) as follows: The table entries for B_2 are retained, where the positions are prefixed by a nonterminal representing the hole path of B_1.
  The table entries from B_1 have to be checked how they can be inherited and adapted:
  Single occurrences can easily be checked by testing whether there is an overlap or an occurrence of $val(C)$ in $val(B_1B_2)$ at the position p. The test may shift entries between tables: an aligned occurrence w.r.t. B_1 may be turned into a partially aligned occurrence w.r.t. B_1B_2.
  Let (p, P, n) be the entry of a periodic sequence, and let $n_1 := |holep(val(P))|$, $h = |holep(val(B_1))| - |p|$ be the length of the path from p to the hole of $val(B_1)$, and $h_1 := h \bmod n_1$ with $h_1 < n_1$. We distinguish between the kinds of sequences:
  - For a sequence of aligned positions: Compute the maximal m_1, k_1 with $m_1 > k_1$ such that $val(P^{m_1} prefix(P, k_1))$ is a prefix of $val(C)$. Let $P' = rot(P, h_1)$, and compute the maximal m_2, k_2 with $m_2 > k_2$ such that $R := val(P'^{m_1} prefix(P', k_2))$ is a prefix of $val(B_2)$. Now we have to distinguish different cases:
    1. Case $val(R) = val(B_2)$, i.e. B_2 is completely covered by iterations of P. We can now easily compute the following, on the basis of the number of periods and the phase-shift, and take the corresponding actions: Compute the new p', n' for the new entry (p', P, n') for C. If C has an aligned occurrence in $val(B_1B_2)$, then the information is outputted, which can only be the case, when C is completely covered by the periods.
    2. Case $val(R) \neq val(B_2)$, i.e., the period P does not cover $val(B_2)$ completely. Then the following cases are possible: (i) Let $val(C)$ be completely covered by P. If $val(C)$ has occurrences in the periodic sequence, which can be computed by comparing the parameters, then this is outputted, but does not lead to an entry in the C-table. A second case is that R' has an overlap with $rot(P, k_2)$. Let R' be equivalent to $f(A_1, \ldots, [.]_i, \ldots, A_n)$, which may have to be constructed, and let j be the direction of the hole at the top of R'. Then we have to compute the periodic parameters for $rot(P, k_2+1)$ in $val(A_j)$. The computed parameters now allow to determine the periodic sequence entry in the sequential partially aligned list of the table.

(ii) $val(C)$ is not completely covered by P. Then let R' be a context nonterminal for the suffix of $val(B_2)$, such that $val(RR') = val(B_2)$. Also let C' be a context nonterminal for the suffix of C with $C = val(P^{m_1} prefix(P, k_1)C')$. There are four possibilities: (1) C' is a prefix of R', (2) R' is a proper prefix of C', (3) R' and C' have an overlap at the top but are not prefixes of each other, or (4) there is no overlap. In case (1) and if the period sequence of B_1B_2 is sufficiently long, there is an occurrence of $val(C)$ in $val(B)$, which is outputted. In case (2), and if the period sequence of B_1B_2 is sufficiently long, a single position for the occurrence of C in B_1B_2 is the new entry. In case (3) of an overlap in the non-prefix case, we have to check whether a partially aligned overlap of C with B is detected. In this case it can only be a single one, and the position p' can be determined using the already computed parameters.

- For a periodic sequence of parallel partially aligned positions:
  Similarly as in the aligned case, let P be the period, and m_2, k_2 be the parameters of the coverage for $val(B_2)$. If B_2 is completely covered, then there will be a new periodic sequence.
  If B_2 is not completely covered, then at most one entry may be in the table for (C, B). The computation is similar as for the aligned case: Compute a context nonterminal R' for the suffix of $val(B_2)$ below the periodic sequence. Also compute a term nonterminal D' for $val(C)$ that is below the periodic sequence. Note that in the case of parallel partial alignment, the periodicity is in a subterm of $val(C)$. If $val(R')$ is a prefix of $val(D')$, then a single entry is in the table, where the position p can be determined by using the already computed numbers.
- For a periodic sequence of sequential partially aligned positions:
  The computation is simple. Let P be the period. Let D' be the term nonterminal that is the side term that overlaps the hole path of $val(B_1)$, and let q be the path in $val(D')$ to the hole of $val(B_1)$ in the overlap. Let D'' be the term nonterminal defining $val(D')_{|q}$. If $val(B_2)$ is not a prefix of $val(D'')$, then there will be no entry in the table and this case is finished. Otherwise, the entry is inherited to B_1B_2.

Finally, a merge of the computed entries has to be done, where single as well as periodic occurrences have to be checked, within every list. This can be done on the basis of the positions and without further extending the STG.

Finally, we have to test whether C occurs in term nonterminals. This is done using the table as follows:

Algorithm 5.19 (Checking occurrences of contexts in terms).
Given a context nonterminal C, for all term nonterminals B the following test is performed:

- If $B ::= f(B_1, \ldots, B_n)$ is a term nonterminal: then we only have to check whether $val(C)$ is a prefix of $val(B)$. If yes, then the result will be printed.

– If $B ::= D[B']$, then we use the table entries for (C, D), which is a list of single and periodic occurrences.
For single occurrences p, the suffix context D' of $val(D)$ has to be computed and then we have to test whether C occurs at the top of $D'[B']$.
For periodic sequences (p, P, n), the test depends on whether the periodic occurrence belongs to aligned, a sequential or parallel partial aligned overlap.
For aligned occurrences, do the following: Let m, k be the parameters of the period occurrences of $val(P)$ in $val(D)$. Use binary search to find the maximal occurrence m_1, k_1 of $rot(P, k)$ in $val(B')$, and also the parameters m_2, k_2 for the periodic occurrences of $val(P)$ in $val(C)$. Also let B'' and C' be the suffixes that are not covered by periods in B' and C, respectively. If C is completely covered by periods, then on the basis of these numbers and the length of the period, all occurrences can be computed and printed. If C is not completely covered, then the suffixes of C' and B''' have to be checked for an overlap, and then a single occurrence can be detected.
For sequential partially aligned occurrences, a single equality test is sufficient to check whether the complete periodic sequence represents occurrences of $val(C)$ or not. The equality check has to test whether $D'[B']$ completes the gap in the periodic sequence in the C-overlap in D, which means an equality check.
For parallel partially aligned occurrences, at most one occurrence of C can be detected. It can be found by first comparing the size of $val(B')$ with the sizes of the suffix contexts of $val(D)$ that start at the forking positions between $val(D)$ and $val(C)$ (as occurring in $val(D)$). Then constructing the necessary nonterminals and an equality check are sufficient for the detection of the occurrence.

5.4 Properties of Compressed Pattern Match

Theorem 5.20. *Assume given an STG G and a context nonterminal C. The algorithm 5.18 computes the occurrence table and the outputs in polynomial time, such that for all context nonterminals B of G, all occurrences of $val(C)$ in $val(B)$ are represented.*
The algorithm 5.19, given the context-in-context occurrence table constructed from algorithm 5.18, and a term nonterminal B, computes a representation of all occurrences of C in B in polynomial time.

Proof. (sketch) We mention the main arguments for correctness and the complexity. The structure of the overlaps of contexts is analyzed in Subsection 5.1. The algorithm for the context-in-context table is based on this analysis. It only memorizes the proper overlaps and prints the detected occurrences.
The number of positions is represented in polynomial space due to Proposition 5.17, and since a compacting step is included. The intermediate size increase of G is only by constructing a prefix of a context, and then by constructing an exponent. There is no iterated size increase, since the exponents can be removed after they are no longer used for equality tests.

Complexity: The table is of polynomial size. The size of the used STG G is only moderately increased (see Theorem 2.9). Main contributions to the time complexity are the binary searches in the periodic sequences, where in every step an equality check in $O(n^3)$ is performed.

Hence the following holds:

Theorem 5.21. *Given an STG G and two nonterminals S, T, where $val(S)$ may contain variables. Then the algorithm for fully compressed pattern match for compressed terms s, t requires at most searching in $|G|^{|Var(s)|}$ alternatives for the substitution.*
A single computation path can be completed in polynomial time.
Thus the submatching problem is in NP.

6 Polynomial Compressed Term Rewriting

In this section we apply the results on the compressed submatch to sequences of reductions by a term rewriting system, which also comprises a single equational deduction step. Given a TRS R and a term t, where we assume that the rules of R as well as t are compressed by STGs, we investigate upper bounds for execution time and the growth of the compressed representation. There are several options for a rewriting strategy, i.e. for choosing the position(s) for a deduction step. There are single-position rewritings at leftmost innermost and outermost-leftmost positions, and also parallel rewritings of the same subterm at several positions using the same axiom. For our compressed representation the natural approach is to use parallel rewriting of the same subterm at several positions and by the same rewriting rule.
First we define how a (parallel) rewriting step on a compressed term t is performed:

Algorithm 6.1. Given a compressed TRS R and a ground term t compressed with G with $val(T) = t$, where we assume that R is compressed by the STG G_R as $\{L_i \to R_i \mid i = 1, \ldots, n\}$ and L_i, R_i are term nonterminals. A term rewriting step is performed as follows:
First we compute a submatch of some left hand side L_i of a rule $L_i \to R_i$ of R in $val(T)$. This will lead to a grammar that is an extension of $G \cup G_R$ and the position of the submatch. The different cases according to Algorithm 4.1 are:

1. The submatch has produced a compressed σ and a compressed position p of $\sigma(val(L_i))$ in t. If there is a term nonterminal A that contributes to t and such that $val(A) = \sigma(val(L_i))$, then replace the rule for A by $A ::= R_i$. and also add the nonterminals and rules representing the compressed substitution σ in the form $x ::= A_x$ to the result STG.
   The other case is that the rule for $A ::= D[A']$ and the match of L_i is in the middle between A and A'. Based on the returned position, we construct context nonterminals D', D'', (and the corresponding rules) such that $val(D'D'') = val(D)$ and $val(D''[A]) = \sigma(val(L_i))$. The rewrite step

is then performed by replacing the rule $A ::= D[A']$ by $A ::= D'[R_i]$, and also adding to the resulting STG the compressed substitution σ in the form $x ::= A_x$ for all the substituted variables.

2. This is an alternative treatment of the deduction step in the exceptional case that the rule can rewrite a context into another context, which is potentially a very efficient deduction in our the representation
Let the submatch have produced a substitution $\sigma := \{x_1 \mapsto A_1, \ldots, x_m \mapsto A_m\}$, such that the term $val(L_i)$ has exactly one occurrence of the variable x_1, and $val(R_i)$ has at most one occurrence of the variable x_1. Let $\sigma_1 := \{x_2 \mapsto A_1, \ldots x_m \mapsto A_m\}$. Assume that $val(\sigma_1(val(L_i))) = val(C[x_1])$, where C is a context nonterminal representing a ground context, and there is a context nonterminal D that contributes to t, and $val(D)$ contains an occurrence of $val(C)$ that is aligned and completely contained in $val(D)$. Then the exceptional rewriting can take place:
The first step is to construct the rule for D as $D ::= D_1D_2; D_2 ::= CD_3$, with fresh context nonterminals D_1, D_2, D_3 and the corresponding rules. If x_1 is not contained in $val(R_i)$, then the rewriting step turns D into a term nonterminal with rule $D ::= D_1[R_i]$, and also adjusts the grammar bottom-up by turning several context nonterminals into term nonterminals. If x_1 is contained in $val(R_i)$, then the rewriting step consists of constructing a nonterminal C' such that $val(R_i) = val(C'[x_1])$ and then by replacing the rule $D_2 ::= CD_3$ with $D_2 ::= C'D_3$. Of course, also the substitution σ_1 has to be added to the grammar.

Now we estimate the size increase of the STG that is used to represent t_n which is the final term after n rewrite steps, i.e., $t \to_R \ldots \to_R t_n$. We distinguish between the STG G_R for the equations and the STG G_n for compression of t_n. Repeating the term rewriting step is done by using a fresh copy of G_R, the STG compressing the TRS R.
In order to complement the estimations in Lemma 2.5, we have to check the size increase of the substitution, and the estimations for the rewrite step itself.

Lemma 6.2. *Let* $M = \max_i(|Var(r_i, l_i)|)$.
A rewrite step consists of M *instantiation steps, which can be seen as independent, since we make a new copy of* G_R *for instantiating every variable. The replacement of* L_i *by* R_i *may increase the depth and Vdepth at most by* $depth(G_R)$. *and for every such step transforming* G *into* G'*, the following estimation holds:*

$$\begin{array}{ll} Vdepth(G', V') & \leq Vdepth(G, V) + \log(depth(G)) + depth(R) \\ |V'| & \leq |V| + M \\ |G'| & \leq |G| + |M|depth(G) + M + depth(G)^2 + |R| \end{array}$$

Proof. The contributions are: (i) for every variable $x \in Var(val(S))$: its substitution construction, i.e. constructing $Var(s)$ times a subterm and then making an instantiation. (ii) Rearranging G such that the position of $val(\sigma(L_i))$ is explicit, and (iii) Modifying the grammar rule by replacing L_i by R_i.

The construction (i) is independent from (ii) and (iii). Constructing the substitution consists in $|\mathit{Var}(s)|$ times independently constructing a subterm of t, which increases the size by $M\,\dot{\mathit{depth}}(G)$, the Vdepth only by $\log(\mathit{depth}(G))$ (since we can take the maximum) and by instantiating, which adds M rules, adds M variables to V, but does not change the Vdepth. Rearranging requires to construct a prefix-context of a term and changing a grammar rule. This may add $\mathit{depth}(G)^2$ nonterminals for the prefix and the nonterminal L_i plus its definition, which means to add $|R|$ to the size. The depth increase is at most $\mathit{depth}(R)$.

Lemma 6.3. *Let there be a sequence $G_i, i = 0, \ldots, n$ of grammars generated by extension or transformation and sets $V_i, i = 0, \ldots, n$ of instantiated nonterminals, such that the following holds:*

$$\begin{array}{ll} \mathit{Vdepth}(G_i, V_i) & \leq \mathit{Vdepth}(G_{i-1}, V_{i-1}) + \mathcal{O}(\log(\mathit{depth}(G_{i-1}))) \\ |V_i| & \leq \mathcal{O}(i) \\ |G_i| & \leq |G_{i-1}| + \mathcal{O}(\mathit{depth}(G_{i-1})^2) \end{array}$$

Then $|G_n|$ is bounded by $O(|G_0| * n^7)$.

Proof. Since $|V_i|$ is the sum of the number of variables in the used term rewrite rule, we have $|V_i| \leq i * M$, where $M = \max_i(|\mathit{Var}(r_i)|)$.
From this bound and Lemma 2.6, we have $\mathit{depth}(G_i) = \mathcal{O}(i)$ $\mathit{Vdepth}(G_i, V_i)$. Therefore, the recurrence for $\mathit{Vdepth}(G_i, V_i)$ may be replaced by

$$\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq \mathit{Vdepth}(G_i, V_i) + \mathcal{O}(\log \mathit{Vdepth}(G_i, V_i)) + \mathcal{O}(\log i). \quad (6.1)$$

A first bound for these recurrence can be computed relaxing the inequality as $\mathit{Vdepth}(G_{i+1}, V_{i+1}) \leq 3\,\mathit{Vdepth}(G_i, V_i) + \mathcal{O}(\log i)$ that has as solution $\mathit{Vdepth}(G_i, V_i) \leq 3^i \left(\mathit{Vdepth}(G_0, V_0) + \mathcal{O}(\log i)\right) = 3^i\,\mathcal{O}(\log i)$. Replacing this approximate solution in (6.1) results in

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

Now we get the approximate solution $\mathit{Vdepth}(G_i, V_i) = \mathcal{O}(i^2)$. Therefore, $\mathit{depth}(G_i) = \mathcal{O}(i)$ $\mathit{Vdepth}(G_i, V_i) = \mathcal{O}(i^3)$. Replacing this in the recursion for $|G_i|$ we get $|G_{i+1}| = |G_i| + \mathcal{O}(i^6)$. Hence, $|G_n| \leq \mathcal{O}(n^7)$.

Theorem 6.4. *Let R be a TRS and t be a term compressed with an STG. Then a sequence of n term rewriting steps according to Algorithm 6.1 can be performed in polynomial time. The size increase by n term rewriting steps is $O(n^7)$, where we assume that the R is $O(1)$*

Proof. The size increase follows from Lemma 6.3, and since we assume that R is constant during the iterated rewriting. The time estimation holds also, since every construction step can be performed in polynomial time and the total size of the final grammar is polynomial.

Note that the degree of the polynomial for the estimation of the worst case running time is worse than the space bound. The term rewriting sequence has to be constructed (+ 1) and Plandowski equality check has to be used in every construction step, which contributes a factor of 3 in the exponent. But note that there are faster randomized equality checks.

6.1 Possible Extensions

A further example shows that more expressive formalisms than STGs may have better compression properties, and may be a subject of future research.

Main theorem 6.4 tells us that the term rewriting algorithm on STG-compressed terms will produce a polynomial-sized grammar depending on the number of steps. This example will show that even for well-behaved equational rules an exponential number of rewrites may be necessary to obtain an irreducible term. This example also shows that an extension of STGs to linear SLCF tree grammars as in [BLM08,LMSS09a], i.e. tree grammars using contexts with multiple holes, where every hole occurs once, does not help. Let the term rewriting system be $f(x) \to g(x,x)$, and let the term $f^n(a)$ be represented as $C_1 ::= f(\cdot)$, $C_2 = C_1C_1$, $C_3 = C_2C_2$, $\ldots C_{n+1} = C_nC_n$, $T ::= C_{n+1}(x)$. A term rewriting step on T using $f(x) \to g(x,x)$ that is applied to C_1 would produce $C_1 := g(.,.)$, which is syntactically not permitted, since there would be two holes. A complete rewriting sequence of $f^n(a)$ would produce a binary tree t' of depth 2^n with 2^{2^n} leaves. Since an STG G can produce only terms of size $2^{|G|}$, the STG for t' would be of exponential size. The result of the rewrite could be easily expressed in a non-linear SLCF tree grammar [BLM08], however, the complexity of the equality check is currently only known to be in PSPACE, and also the complexity of all other operations like matching, extensions, rewriting etc. has not been investigated yet.

7 Future Research and Conclusion

We have shown that finding an instance of a term s as a subterm of t under grammar compression with STGs can be done in time $O(n^{O(|\,Var(s)|)})$, and that there are several cases where the algorithm or variants of the algorithm run in polynomial time. Also, if we assume that the STG for a compressed term rewriting system is constant, then a sequence of n (parallel) rewrites of a compressed term can be computed in polynomial time.

Further research is to find improved (polynomial) algorithms for the submatching problem in the further cases, and to determine the complexity of the general case of submatching.

Another line of research is to investigate the submatching in generalizations of STGs as in linear and nonlinear SLCF tree grammars.

References

[AM10] Martin Avanzini and Georg Moser. Closing the gap between runtime complexity and polytime computability. In Christopher Lynch, editor, *21st RTA*, volume 6 of *LIPIcs*, pages 33–48, Germany, 2010. Schloss Dagstuhl.

[BLM05] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML documents. In *Proceedings of DBPL 2005*, number 3774 in LNCS, pages 199–216, 2005.

[BLM08] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998.

[CDG+97] 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 October 2002.

[Com95] Hubert Comon. On unification of terms with integer exponents. *Mathematical Systems Theory*, 28(1):67–883, 1995.

[GGSS08] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Context matching for compressed terms. In *23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*, pages 93–102. IEEE Computer Society, June 2008.

[GGSS10] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Unification and matching on compressed terms. http://arxiv.org/abs/1003.1632v1, March 2010.

[Lif07] Yury Lifshits. Processing compressed texts: A tractability border. In *CPM 2007*, pages 228–240, 2007.

[LM05] M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. In *Proc. of the 10th CIAA '05*, 2005.

[LMM10] Markus Lohrey, Sebastian Maneth, and Roy Mennicke. Tree structure compression with RePair. *CoRR*, abs/1007.5406, 2010.

[LMSS09a] Markus Lohrey, Sebastian Maneth, and Manfred Schmidt-Schauß. Parameter reduction in grammar-compressed trees. In *FOSSACS*, volume 5504 of *LNCS*, pages 212–226. Springer, 2009.

[LMSS09b] Markus Lohrey, Sebastian Maneth, and Manfred Schmidt-Schauß. Parameter reduction in grammar-compressed trees. In *12th FoSSaCS*, volume 5504 of *LNCS*, pages 212–226. Springer, 2009.

[LSSV06] Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. Bounded second-order unification is NP-complete. In *Term Rewriting and Applications (RTA-17)*, volume 4098 of *LNCS*, pages 400–414. Springer, 2006.

[LSSV10] Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal IGPL*, 2010. to be published.

[Pla94] Wojciech Plandowski. Testing equivalence of morphisms in context-free languages. In *ESA 94*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470, 1994.

[Ryt04] Wojciech Rytter. Grammar Compression, LZ-encodings, and string algorithms with implicit input. In J. Diaz et. al., editor, *ICALP 2004*, volume 3142 of *LNCS*, pages 15–27. Springer-Verlag, 2004.

[Sal92] Gernot Salzer. The unification of infinite sets of terms and its applications. In *LPAR 1992*, volume 624 of *LNCS*, pages 409–420, 1992.

[SS05] Manfred 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.

[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.