Linear Compressed Pattern Matching for Polynomial Rewriting (Extended Abstract)

Manfred Schmidt-Schauss
Institut für Informatik,
Fachbereich Informatik und Mathematik,
Goethe-Universität,
Postfach 11 19 32, D-60054 Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de

This paper is an extended abstract of an analysis of term rewriting where the terms in the rewrite rules as well as the term to be rewritten are compressed by a singleton tree grammar (STG). This form of compression is more general than node sharing or representing terms as dags since also partial trees (contexts) can be shared in the compression. In the first part efficient but complex algorithms for detecting applicability of a rewrite rule under STG-compression are constructed and analyzed. The second part applies these results to term rewriting sequences.

The main result for submatching is that finding a redex of a left-linear rule can be performed in polynomial time under STG-compression.

The main implications for rewriting and (single-position or parallel) rewriting steps are: (i) under STG-compression, n rewriting steps can be performed in nondeterministic polynomial time. (ii) under STG-compression and for left-linear rewrite rules a sequence of n rewriting steps can be performed in polynomial time, and (iii) for compressed rewrite rules where the left hand sides are either DAG-compressed or ground and STG-compressed, and an STG-compressed target term, n rewriting steps can be performed in polynomial time.

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. [2]). The basic and widely used algorithms in these areas are matching, unification, term rewriting, equational deduction, 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) for compression of strings is a general one and allows analyses of correctness and complexity of algorithms [21, 16]. SLPs are polynomially equivalent to the LZ77-variant of Lempel-Ziv compression [25]. SLPs are non-cyclic context free grammars (CFGs), where every nonterminal has exactly one production in the CFG, such that any nonterminal represents exactly one string. Basic algorithms are the equality check of two compressed strings, which requires polynomial time [19] (see [15] for an efficient version and [11] for a proposal of a further improvement), and the compressed pattern match, i.e., given two SLP-compressed strings s,t, the question whether s is a substring of t 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) [22, 13, 7], a specialization of straight line context free tree grammars [4, 5, 17, 18], where linear SLCF tree grammars

R. Echahed and D. Plump (Eds.): 7th International Workshop on
Computing with Terms and Graphs
EPTCS 110, 2013, pp. 29–40, doi:10.4204/EPTCS.110.5

are polynomially equivalent to STGs [17, 18]. Basic notions for tree grammars and tree automata can be found in [6]. 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 to STGs and requires polynomial time [4, 22] in the size of the STG.

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. A generalization of the pattern match is the following submatching problem (also called encompassment): given two (STG-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 in PTIME in [7, 8], including the computation of the (unique) compressed substitution.

In this extended abstract (of [23]) we report informally on progress in finding algorithms operating on STGs for answering the submatching question, and which only operate on the STGs. We show that if s is STG-compressed and linear, then submatching can be solved in polynomial time (Theorem 3.7). If s is ground and compressed or s is DAG-compressed, we describe less complex algorithms that solve the submatching question in polynomial time (Theorem 4.1 and Theorem 4.3). In the general case, we describe a non-deterministic algorithm that runs in polynomial time. The deterministic algorithm runs in time $O(n^{c|FVmult(s)|})$ (Theorem 4.4), where n is the size of the STG and $FVmult(s)$ the set of variables occurring more than once in s. This is an exponential-time algorithm, but in a well-behaved parameter.

As an application and an easy consequence of the submatching algorithms, a (single-position or parallel) deduction step on compressed terms by a compressed left-linear rewriting rule can be performed in polynomial time. We also show that a sequence of n rewrites with a STG-compressed left-linear term rewriting system on an STG-compressed target term can be performed in polynomial time (see Theorem 5.1). Our result confirms results on complexity of rewrite derivations under DAG-compression [1], namely that rewrite systems with a polynomial runtime complexity can be implemented such that the algorithm requires polynomial time.

Example 1.1 *Consider the term rewriting rule $f(x) \to g(x,b)$, and let the term $t_1 = f(f(f(a)))$ be compressed as $C_1 \to f(\cdot)$, $C_2 \to C_1C_1$, $T \to C_2(T'), T' \to f(a)$. A single term rewriting step on the compressed term t_1 by the rule $f(x) \to g(x,b)$ would produce $T' \to g(a,b)$, 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 $C_1 \to f(\cdot)$, $C_2 \to C_1C_1$, $C_3 \to C_2C_2$, $C_4 \to C_3C_3$, $C_5 \to C_4C_4$, $T \to C_5(a)$. A term rewriting step on T using $f(x) \to g(x,b)$ may rewrite the context $f(\cdot)$ and thus would produce $C_1 \to g(\cdot,b)$, and hence reduces the term in one blow to $g(\ldots,(g(\ldots,b)\ldots),b)$, which is a parallel rewriting step, see Section 5.*

The structure of this extended abstract (of [23]) is as follows. First the basic notions, in particular STGs, are introduced in Section 2. An algorithm for linear submatching is explained in Section 3. In Section 4 we explain submatching for some special cases and also a general non-deterministic algorithm for term submatching of compressed patterns and terms. Finally, in Section 5, we illustrate the application in term rewriting and argue that n rewrites for a left-linear TRS can be performed in polynomial time.

2 Preliminaries

We will use standard notation for signatures, terms, positions, and substitutions (see e.g. [2]). A position is a word over positive integers. For two positions p_1, p_2, we write $p_1 \leq p_2$, if p_1 is a prefix of p_2, and

$p_1 < p_2$, if p_1 is a proper prefix of p_2. We call two strings w_1, w_2 *compatible*, if w_1 is a prefix of w_2, or w_2 is a prefix of w_1. We write $p[i]$ for the i^{th} symbol of p, where 0 is the start index, and $p[i,j]$ for the substring of p starting at i ending at j. The set of free variables in a term t is denoted as $FV(t)$. Let $FVmult(s)$ be the set of variables occurring more than once in s. Terms without occurrences of variables are called *ground*. A term where every variable occurs at most once is called *linear*. A *context* is a term with a single hole, denoted as $[\cdot]$. Sometimes it is convenient to view a linear term containing one variable as a context, where the single variable represents the hole. As a generalization, a *multicontext* is a linear term, where the variable occurrences are also called holes. Let $holep(c)$ be the position (as a string of numbers) of a hole in a context c, and let the *hole depth* be the length of $holep(c)$. If $c = c_1[c_2]$ for contexts c, c_1, c_2, then c_1 is a *prefix context* of c and c_2 is a *suffix context* of c. The notation $c[s]$ means the term constructed from the context c by replacing the hole with s. An n-fold iteration of a context c is denoted as c^n; for example c^3 is $c[c[c]]$. A *substitution* σ is a mapping on variables, extended homomorphically to terms by $\sigma(f(t_1,\ldots,t_n)) = f(\sigma(t_1),\ldots,\sigma(t_n))$.

Definition 2.1 *A term rewriting system (TRS) R is a finite set of pairs $\{(l_i, r_i) \mid i = 1,\ldots,n\}$, called* rewrite rules*, 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 by R is $t \xrightarrow{R} t'$, if for some i: $t = c[\sigma(l_i)]$ and $t' = c[\sigma(r_i)]$ for some context c and some substitution σ.*

2.1 Tree Grammars for Compression

First we introduce string compression: A *straight line program* (SLP) is a context-free grammar that generates one word, has no cycles, and for every nonterminal A there is exactly one production of the form $A \to A_1A_2$ or $A \to a$.
An application for SLPs is the representation of compressed positions in compressed terms. We will use the well-known (polynomial-time) algorithms, constructions and their complexities on SLPs like equality check of compressed strings, computing prefixes, suffixes, the common prefix (suffix) of two strings (see [21, 9, 19, 20, 12, 15, 14]).
We consider compression of terms using tree grammars:

Definition 2.2 *A* singleton tree grammar (STG) *is a 4-tuple $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, \mathcal{R})$, where $\mathcal{TN}$ are tree/term nonterminals of arity 0, $\mathcal{CN}$ are context nonterminals of arity 1, and Σ is a signature of function symbols (the terminals), such that the sets $\mathcal{TN}$, $\mathcal{CN}$, and Σ are finite and pairwise disjoint. The set of nonterminals $\mathcal{N}$ is defined as $\mathcal{N} = \mathcal{TN} \cup \mathcal{CN}$. The productions in $\mathcal{R}$ must be of the form:*

- $A \to f(A_1,\ldots,A_m)$*, where $A, A_i \in \mathcal{TN}$, and $f \in \Sigma$ is an m-ary terminal symbol.*
- $A \to C_1A_2$ *where $A, A_2 \in \mathcal{TN}$, and $C_1 \in \mathcal{CN}$.*
- $C \to [\cdot]$ *where $C \in \mathcal{CN}$.*
- $C \to C_1C_2$*, where $C, C_1, C_2 \in \mathcal{CN}$.*
- $C \to f(A_1,\ldots,A_{i-1},[\cdot],A_{i+1},\ldots,A_m)$*, where $A_1,\ldots,A_{i-1},A_{i+1},\ldots,A_m \in \mathcal{TN}$, $C \in \mathcal{CN}$, and $f \in \Sigma$ is an m-ary terminal symbol.*
- $A \to A_1$ *(λ-production), where A and A_1 are term nonterminals.*

Let $N_1 >_G N_2$ for two nonterminals N_1, N_2, iff $(N_1 \to t) \in R$, and N_2 occurs in t. The STG must be non-cyclic, i.e. the transitive closure $>_G^+$ must be irreflexive. Furthermore, for every nonterminal N of G there is exactly one production having N as left-hand side. Given a term t with occurrences of nonterminals, the derivation of t by G is an exhaustive iterated replacement of the nonterminals 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 nonterminal 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. The size of an STG is the number of its productions, denoted as $|G|$.*

Definition 2.3 *Let G be an STG and V be a set of variables. Then (G,V) is an* STG with variables*, where additional production forms are permitted:*

- *$A \to x$, where $A \in \mathcal{TN}$ and $x \in V$.*
- *$x \to A$ (λ-production), where $x \in V$ and $A \in \mathcal{TN}$.*

This means that variables may be terminals or nonterminals, depending on the existing productions. The measure $Vdepth(N,V)$ is defined as the maximal number of $>_G$-steps starting from N until an element of V or a terminal is reached, and $Vdepth(G,V)$ the maximum.
In the following we always mean STG with variables if variables are present.
An STG G is called a DAG*, if there are no context nonterminals.* □

The compression rate may be exponential in the best case, but not larger: The size of terms represented with an STG G is at most $O(2^{|G|})$. Note that the term depth of DAG-compressed terms is at most the size of the DAG, whereas the term depth of STG-compressed terms may be exponential in the size of the STG. Note also that every subterm in a DAG-compressed term is represented by a nonterminal, whereas in STG-compressed terms, there may be subterms that are only implicitly represented. It is known that several computations in SLPs and STG, for example length computations, can be done in polynomial time. Several forms of extensions of STGs are well-behaved, such that even a sequence of n such extensions will lead to only polynomial size growth.

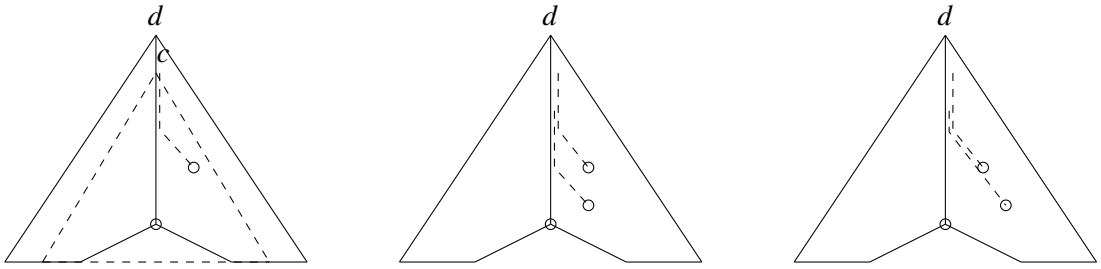
Compressed Matching. The investigation in [7] shows that (exact) term matching, also in the fully compressed version including the computation of a compressed substitution, is polynomial. I.e. given two nonterminals S,T, where S may contain variables, there is a polynomial time algorithm for answering the question whether there is some substitution σ such that $\sigma(val(S)) = val(T)$, and also for computing the substitution, where the representation is a list of variable-nonterminal pairs, and the nonterminals belong to an extension of the input STG.

Compressed 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. Submatching (also called encompassment relation) is a prerequisite for term rewriting.

Definition 2.4 *The* compressed term submatching problem *is:*
*Assume given a term s which may contain variables, and a (ground) term t, both compressed with an STG $G = G_S \cup G_T$, such that $val(T) = t$ and $val(S) = s$ for term nonterminals $S \in G_S$, $T \in G_T$. 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. Specializations are:*uncompressed *if s is given as a plain term without any compression;* ground *if s is ground;* DAG-compressed*, if s is DAG-compressed; and* linear*, if s is a linear term, i.e. every variable occurs at most once in s.*

Lemma 2.5 *Given an STG G, a term s and a nonterminal T, 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 $val_G(B) = \sigma(s)$.*



(a) non-compatible overlap (b) parallel (c) sequential
Subfigures (b) and (c) only show the hole path of two occurrences of the context c.

Figure 1: Non-compatible, parallel and sequential overlap of c with d

2. *There is a production $B \to CB'$ in G, such that $\sigma(s) = c[val_G(B')]$, where c is a nontrivial 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 algorithms below have to distinguish the* subterm case *where x occurs more than once in s and the* subcontext case *where x occurs exactly once in s.*

3 Term Submatching with Linear Terms

Overlaps of Linear Terms and Contexts. An important concept and technique used is periodicity of contexts. This is a generalization of periodicity of strings: for example the string "bcabcabc" is periodic with period length 3. A context c is called periodic if $c = d^n d'$ for some contexts d, d' and a positive integer n, where d' is a prefix of d. This is even generalized to multicontexts c (linear terms, where the variables are the holes), and where periodicity means that c can be overlapped with itself at periodic positions without conflicts.
We consider overlapping multicontexts $c, c_1, c_2, \ldots$ and a context d. In particular special variants of overlaps have to be analyzed: Overlaps where the hole of d is not compatible with any hole of c. The overlaps where a hole of c is compatible with a hole of d can be dealt with generalizing results from words (or words with character-holes). If there are non-compatible overlaps of copies of c with d, then only two configurations are possible: parallel and sequential (see Proposition 3.2 and Fig. 1), and there are no mixed configurations. Thus, periodicities in linear terms are not only possible along the hole-path of d but also along other paths, and there are two different kinds of such periodicities: the parallel and the sequential variant. A helpful technical result is a periodicity theorem that tells us that a multi-context c is periodic, if there is a multiple overlap of $h+2$ copies of c where h is the number of holes, and the overlap is sufficiently dense. This will be used in the submatching algorithm for linear terms.

Example 3.1 *Let $d = f(a_1, f([\cdot], a_1))$ and let $c = f(a_1, [\cdot])$. Then c overlaps d at position ε, which is a compatible overlap, since the start as well as the hole position of c is on the hole path of d. The overlap of c with d at position 2 (in d) is a non-compatible overlap, since the hole of c is at 2.2, which is not a prefix or suffix of the hole path of d, which is 2.1.*

Proposition 3.2 *Let c be a multicontext with at least one hole, and let d be a context with exactly one hole, and let $p_1 < p_2$ be two positions of non-compatible overlaps of c in d. Let q_i be the maximal common hole path (mchp) of c at p_i for $i = 1,2$. Then there are the following two cases (see Figure 1):*

1. *$q_1 = q_2$ (the* parallel overlap *case). Then for p' such that $p_1 p' = p_2$ the path $p_1(p')^n$ is compatible with $holep(d)$ for all n. Also, this is a multiple overlap of c' with itself at positions $(p')^i$, where c' is constructed from c with an extra hole at p'', where $p_1 p'' = holep(d)$.*
2. *$q_2 < q_1$ (the* sequential overlap *case). Then $p_2 q_2 = p_1 q_1$. I.e., there is a fixed position on the hole path of d, where the hole paths of occurrences of c deviate.*

Example 3.3 *Let $c' = f(f(a_1,a_2),[\cdot])$ be a context, $c = f(f(x,y),(c')^{100}[.])$, and let $d = (c')^{100}[\cdot]$. Then there is an overlap of c with d at positions $\varepsilon, 2, 2.2, \ldots$. It is an overlap of the first kind, i.e. a parallel overlap. A sequential overlap is the following: Let $c = f(a_1, f(a_1, f(a_1, [\cdot])))$ and let $d = f(a_1, f(a_1, f(a_1, f(a_1, f([\cdot], f(a_1, f(a_1, a_1)))))))$. Then the overlap positions are $\varepsilon, 2, 2.2, 2.2.2$.*

Theorem 3.4 (Periodicity-Theorem) *Let c be a multi-context with $h \geq 1$ holes. Let p be the position of a fixed hole of c, and let $p_i, i = 1, \ldots, n$ be prefixes of p such that $i < j$ implies $p_i < p_j$ with $n \geq h+2$. Assume that there is a (right-cut) overlap of n copies of c starting at position p_i such that p is a prefix of $p_i p$, i.e., the hole position of c starting at p_i is compatible with p for all i, and only positions in c at p_1 are relevant for the overlap. Let p_{max} be $\max\{|p_{i+1}| - |p_i| \mid i = 1, \ldots, n-1\}$. Assume $|p| - |p_n| \geq 2h \cdot p_{max}$; this means there are $2h \cdot p_{max}$ common positions on the path p of all occurrences of c.*
Then the multicontext c is periodic (in the direction p), and a period length is $p_{all} := gcd(|p_2| - |p_1|, |p_3| - |p_2|, \ldots, |p_n| - |p_{n-1}|)$. Moreover, the overlap is consistent with using the same substitution for the variables for every occurrence of c.

Tabling Prefixes of Multicontexts in Contexts.
The core of the algorithm for finding submatches of a linear term s in other terms (under STG-compression) is the construction of a table in dynamic-programming style. The table contains overlaps of s with contexts that are explicitly represented in the STG G by a context nonterminal. In fact the table is split into several tables: There is a table per context nonterminal A of G and per variable (hole) of s for the compatible overlaps. In addition there is an extra table for non-compatible overlaps. This makes $h+1$ tables where h is the number of variables of s.
The entries in the tables are pairs of a position and a substitution necessary for the overlap. Since terms of exponential size and depth may be represented in the STG G, a compact representation of a large number of entries is necessary in order to keep the tables of polynomial size. Indeed this is possible exploiting periodicity. If the number of entries in a table are sufficiently dense, then the periodicity theorem implies that a large subset of the entries enjoys regularities, and a series of periodic overlaps can be represented in one entry, consisting of: a start position, a period (a position, respectively a context nonterminal), and the number of successive entries.
In more detail, the construction of the prefix tables is bottom-up w.r.t. the grammar where the productions $A \to A_1 A_2$ for context nonterminals permit to construct the A-tables from the A_1,A_2-tables, and where the start are the contexts with hole-depth 1. This construction must take into account the compact representation of the entries: single ones and periodic ones, which makes the description of the algorithm rather complex due to lots of cases. The construction of the prefix table in the case $A \to A_1 A_2$ and the periodic cases is depicted in Figure 2 where (a) shows the case where A has a periodic suffix, (b) shows the case where A has an inner part that is periodic, (c) shows a case where the periodicity goes into a direction that is not compatible with the hole of A_2, which leads to the sequential overlap case; and (d) is a case of a sequential overlap already in the table for A_1. The generation of the periodic entries is done in

an extra step: compaction, where the periodic overlaps are detected by searching for sufficiently dense entries. This is the only place where periodic entries are generated.
In addition to the prefix tables there is a result table, which contains the detected submatchings, and which is maintained during construction of the prefix tables.
Since it is necessary to also have submatchings in terms, i.e. for term nonterminals, we keep things simple and assume that every production for a term nonterminal is of the form $A \to CA_1$, where A_1 is a term nonterminal with production $A_1 \to a$, i.e. a constant. This rearrangement of G can be done efficiently, and thus does not restrict generality. For these nonterminals the extraction of the submatchings can be done using the already constructed prefix-tables.
Note that during construction of the tables, the STG G may have to be extended in every step.

Example 3.5 *We describe several small examples for compatible entries in a prefix table. Therefore we slightly extend Example 3.3. Let the STG be* $S \to A; A \to A_1A_1; A_1 \to A_2A_2, A_2 \to f(a_1,[\cdot])$.

1. *Then* (C,A_2,∞) *for* $C \to [\cdot]$ *is a potential entry in a result table for A.*
2. *Let* $A_4 \to g([\cdot]), B \to A_4A, C' \to A_4$. *Then* (C',A_2,∞) *is an entry in the result table for B.*
3. *Let* $B' \to BA_4$, *then* $(A_4,A_2,2)$ *is a potential entry in the result table for* B'.
4. *The tuple* $(A_4,A_2,3)$ *is an entry in the prefix table for B.*
5. *Let* $B'' \to A_6A_4, A_6 \to A_4A_1$. *The context* A_6 *is then a potential entry in the result and prefix tables of* B''.

Note that item 4 cannot be used as a result, since composing B as in $B' \to BA_4$ *in item 3, may render an overlap invalid.*

Example 3.6 *We describe an example for a non-compatible entry in a prefix table. Therefore we slightly modify Example 3.3. Assume there is an STG G. Let* $c = f(a_1,f(a_1,f(a_1,f(a_1,[\cdot]))))$, $d = f(a_1,f(a_1,f(a_1,f([\cdot],f(a_1,f(a_1,a_1))))))$, *and let* P,D,C_0,S *be a nonterminals such that* $val(P) = f(a_1,[\cdot])$, $val(D) = d, val(S) = c$, $val(C_0) = [\cdot]$. *Then an entry in the non-compatible prefix table for D could be* $(C_0,P,3)$.

Theorem 3.7 (Linear Submatching) *Let G be an STG, and* S,T *be two term nonterminals such that* $val(S)$ *is a linear term, and the submatching positions of* $val(S)$ *in* $val(T)$ *are to be determined. Then the algorithm for linear submatchings computes an* $O(|G|^5)$*-sized representation of all submatchings of* $val(S)$ *in* $val(T)$ *in polynomial time dependent on the size of G.*

4 Submatching Algorithms for Other Cases

We consider several specialized situations: ground terms, uncompressed patterns, DAG-compressed terms, and also non-linear terms.

4.1 Ground Term Submatching

If s is ground and compressed by a nonterminal S then submatching can be solved in polynomial time by translating both compressed terms into their compressed preorder traversals (i.e. strings) [4, 5] and then applying string pattern matching [21, 15]. The string matching algorithm in [15, 11] computes a polynomial representation of all occurrences. Note that in our case, the structure of ground terms is

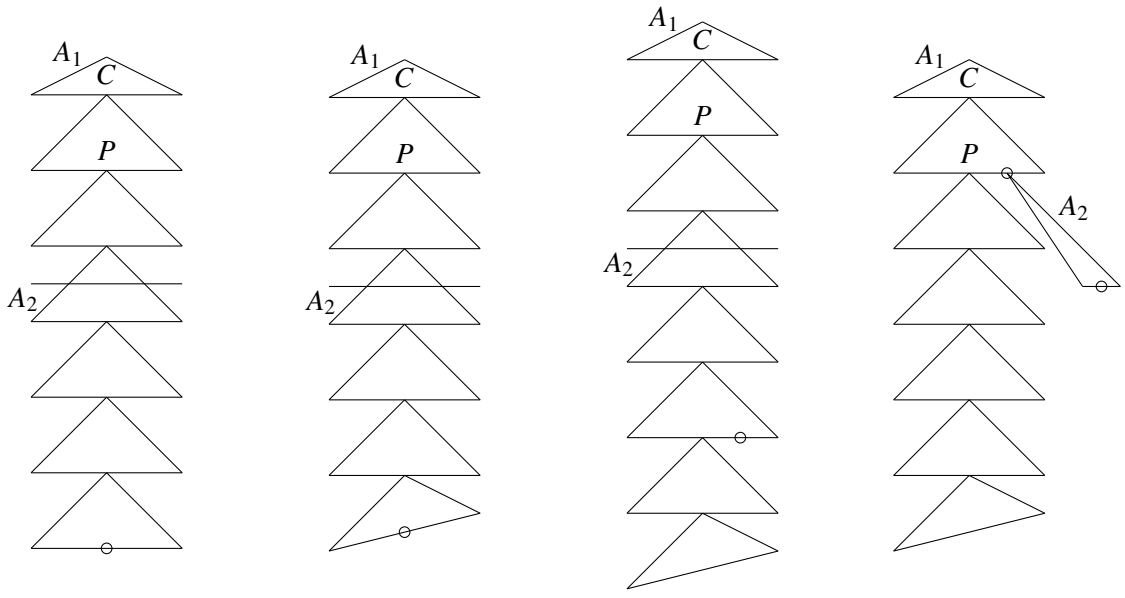


Figure 2: Cases in the construction of the prefix tables for periodic entries

very special as a string matching problem: periodic overlaps of the preorder traversal as strings are not possible. Thus the complete output of the algorithm is as 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 production for N is of the form $N \to CN'$, 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.
The required time for string matching is $O(n^2m)$ where n is the size of the SLP of T and m is the size of the SLP of S. Since the preorder traversal can be computed in linear time (see [8]), we have:

Theorem 4.1 *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.*

4.2 DAG-Compressed Non-Linear Submatching

Now we look for the case of DAG-compressed s, which is slightly more general than the uncompressed case, and where variables may occur several times in s. Also for this case, there is an algorithm for submatching that requires polynomial time. The algorithm outputs enough information to determine all the positions and substitutions of a submatch.

Example 4.2 *The number of possible substitutions for a submatch in a DAG-compressed term may be exponential: Let the productions be $S \to f(x,y)$, and $T \to f(A_1,A_1), A_1 \to f(A_2,A_2), \ldots, A_{n-1} \to f(A_n,A_n), A_n \to a$. Then $val(T)$ is a complete binary tree of depth n and there is a submatch at every non-leaf node. Clearly, it is sufficient to have all A_i as submatchings in the output, which is of linear size.*

In the case of a DAG-compressed or uncompressed pattern-term (not necessarily linear) s and STG-compressed target term t, the algorithm for computing all submatchings is designed in dynamic programming style. It constructs a table of possible submatchings of s in the context nonterminals corresponding

(a) (b)

Figure 3: Cases in the construction of the s-in-C-table for DAG-compression

to t. The key of the table is (C,p), where C is a context nonterminal, and p a position that is a suffix of $val(C)$ as well as a position in s. The number of these positions is linear in $|G_s|+|G_t|$ for every context. The entries are substitutions into the variables of s, i.e. a list of pairs (x_i,A_i), where A_i is a term nonterminal representing a ground term. There is also a result list of found submatchings in contexts C contributing to T, and term nonterminals for ground terms that are instances of s. The construction proceeds again bottom-up in the STG G_t for context nonterminals, and for $A \to A_1A_2$, constructs the table for A from the tables for A_1,A_2, and in case a full submatching is found, inserts a result into the result list.
Finally, from these information, a representation of all submatchings can be constructed by looking at the right hand sides of the productions $A \to CB$ for term nonterminals, and using the table entries for C, and also constructing the occurrences of the ground terms.

Theorem 4.3 *Let G be an STG, and S,T be two term nonterminals such that S is DAG-compressed. Then the submatch computation problem can be solved in polynomial time. Also an explicit polynomial representation of all matching possibilities can be computed in polynomial time.*

4.3 A Non-Deterministic Algorithm for Sub-Matching in the General Case

The submatching problem for STG-compressed pattern terms that may be nonlinear can be solved by a relatively easy search that leads to a non-deterministic polynomial time algorithm: Given S, with nonlinear $s = val(S)$, extract and construct a nonterminal B representing a subterm $f(r_1,\ldots,r_n)$ of s such that two terms r_i,r_j contain a common variable. Then non-deterministically choose a right hand side r of a production of G_t of the form $f(\ldots)$, then compute the usual match of B with r using [7] which will produce an instantiation of at least one variable of $val(B)$, and hence of s. Then iterate this until all variables with double occurrences are instantiated. For the resulting linear term we know how to find all matching positions.

Theorem 4.4 (Nondeterministic General Submatch) *Let G be an STG and S,T be two nonterminals of G where $val(S)$ may contain variables. Then the algorithm for fully compressed submatching for compressed terms s,t requires at most searching in $|G|^{|FVmult(s)|}$ alternatives for the substitution and the computation for one alternative can be done in polynomial time. Thus the submatching problem is in NP.*

There remains a gap in the knowledge of the complexity of the fully compressed submatching problem for terms, which for the decision problem is between PTIME and NP.

Remark 4.5 *The non-linear submatching problem can be computed in polynomial time if there are few variable occurrences ($\leq |G|$) in s: First linearize s, then use the linear compressed submatch and then perform a postprocessing checking equality enforced by the variables of s.*

5 Polynomial Compressed Term Rewriting

For our compressed representation the natural approach to rewriting is to use parallel rewriting of the same subterm at several positions and by the same rewriting rule. Note, however, that the set of redexes that are rewritten in parallel will depend on the structure of the STG G_t, and not on the structure of the rewritten term t.
Let R be a compressed TRS, let t be a ground term with $val_G(T) = t$, let R be compressed by the STG G_R as $\{L_i \to R_i \mid i = 1,\ldots,n\}$ where L_i, R_i are term nonterminals.
A (parallel) term rewriting step is performed as follows:
First select $L_i \to R_i$ as the rule. There is an oracle, which is one of our submatching algorithms applied to L_i, for finding the redex for $val(L_i)$ or the set of redexes that provides the following:

1. An extension G' of G, i.e. additional nonterminals and productions.
2. A substitution σ as a list of pairs: $\{x_1 \mapsto A_1,\ldots,x_m \mapsto A_m\}$, where $FV(val(L_i)) = \{x_1,\ldots,x_m\}$, A_i are term nonterminals in G', and $val(A_i)$ is a subterm of t. It is also assumed that the instantiation is integrated in the grammar G' as productions $x_i \to A_i$ for $i = 1,\ldots,m$.
3. A term nonterminal A (corresponding to L_i) in G' which contributes to $val(T)$, and a compressed position p.

Then the rewriting step is performed by modifying the grammar such that somewhere in the part of the grammar contributing to t: L_i is replaced by R_i. This will also generate an extension of G_t on the fly and also a copy of the STG G_R is made.
A single-position rewriting step under STG-compression is performed in a similar way.

Theorem 5.1 *Let R be a TRS compressed with G_R and t be a term compressed with an STG G. Then a sequence of n term rewriting steps where submatching is a non-deterministic oracle that is not counted, can be performed in polynomial time. The size increase by n term rewriting steps is $\mathcal{O}\big(|G_R|^2 n^7\big(|G|^2 + |G|(\log n + 2|G_R|) + (\log n + |G_R|)^2\big)\big)$.*

The complexity bound is $\mathcal{O}(n^7 \log^2(n))$ depending on the number n of rewrites; $\mathcal{O}(|G_0|^2)$ depending on the size of G_T; and $\mathcal{O}(|G_R|^4)$ depending on the size of G_R. 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 deterministic tests [15, 11] and even faster randomized equality checks [10, 3, 24].
Single-position rewriting requires a partial decompression of the redex position (similar to the parallel), which leads to an extra increase in the size of the STG, but to the same, still polynomial, complexity.
Combining the results on submatching and sequences of rewriting, we obtain the following corollaries:

Corollary 5.2 *Let R be an STG-compressed TRS and t be an STG-compressed term. Then a sequence of n term rewriting steps using the submatching algorithm in Subsection 4.3 can be performed in non-deterministic polynomial time.*

Proof. This follows from Theorems 5.1 and 4.4. □

Corollary 5.3 *Let R be a left-linear STG-compressed TRS and t be an STG-compressed term. Then n term rewriting steps where the submatching algorithms in Subsection 4.3 are used can be performed in polynomial time.*

Proof. This follows from Theorems 5.1 and 3.7. □

Corollary 5.4 *Let R be a TRS with DAG-compressed left-hand sides and STG-compressed right hand sides and let t be an STG-compressed term. Then n term rewriting steps where the submatching algorithm in Subsection 4.2 is used can be performed in polynomial time in n.*

Proof. This follows from Theorems 5.1 and 4.3. □

Corollary 5.5 *Let R be an STG-compressed TRS and t be an STG-compressed term, such that the left hand sides of every rule has at most $|G|$ occurrences of variables. Then n term rewriting steps (see Remark 4.5) can be performed in polynomial time in n.*

6 Conclusion

We have constructed several polynomial algorithms for finding a submatch under STG-compression, or restrictions thereof. It is also shown that n rewrite steps can be performed in polynomial time under STG-compression in several cases: left-linear and STG-compressed TRS, DAG-compressed or ground left hand sides of rules. Also in the general case of non-linear left hand sides n rewrites can be performed non-deterministically in polynomial time, where a search for a redex is required. This is connected to the open problem of the exact complexity of computing submatches also for non-linear terms.
A connection to the results in [1] on polynomial runtime complexity is that our results also imply that for TRSs with polynomial runtime complexity the (single-position and parallel) rewriting can be implemented such that n rewrite steps can be performed in polynomial time.
A remaining open question is whether the general STG-compressed submatching (of nonlinear terms s in t) can be solved in polynomial time or not.

References

[1] Martin Avanzini & Georg Moser (2010): *Closing the Gap Between Runtime Complexity and Polytime Computability*. In Christopher Lynch, editor: *21st RTA*, *LIPIcs* 6, Schloss Dagstuhl, Germany, pp. 33–48, doi:10.4230/LIPIcs.RTA.2010.33.

[2] Franz Baader & Tobias Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA.

[3] Piotr Berman, Marek Karpinski, Lawrence L. Larmore, Wojciech Plandowski & Wojciech Rytter (2002): *On the Complexity of Pattern Matching for Highly Compressed Two-Dimensional Texts*. *J. Comput. Syst. Sci.* 65(2), pp. 332–350, doi:10.1006/jcss.2002.1852.

[4] Giorgio Busatto, Markus Lohrey & Sebastian Maneth (2005): *Efficient Memory Representation of XML Documents*. In: *Proceedings of DBPL 2005*, *LNCS* 3774, pp. 199–216, doi:10.1007/11601524_13.

[5] Giorgio Busatto, Markus Lohrey & Sebastian Maneth (2008): *Efficient Memory Representation of XML Document Trees*. *Information Systems* 33(4–5), pp. 456–474, doi:10.1016/j.is.2008.01.004.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison & M. Tommasi (1997): *Tree Automata Techniques and Applications*. Available at http://www.grappa.univ-lille3.fr/tata. Release October 2002.

[7] Adrià Gascón, Guillem Godoy & Manfred Schmidt-Schauß (2008): *Context Matching for Compressed Terms*. In: *23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*, IEEE Computer Society, pp. 93–102, doi:10.1109/LICS.2008.17.

[8] Adrià Gascón, Guillem Godoy & Manfred Schmidt-Schauß (2011): *Unification and matching on compressed terms*. *ACM Trans. Comput. Log.* 12(4), pp. 26:1–26:37. Available at http://doi.acm.org/10.1145/1970398.1970402.

[9] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski & Wojciech Rytter (1996): *Efficient Algorithms for Lempel-Ziv Encoding (Extended Abstract)*. In Rolf G. Karlsson & Andrzej Lingas, editors: *SWAT*, *Lecture Notes in Computer Science* 1097, Springer, pp. 392–403, doi:10.1007/3-540-61422-2_148.

[10] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski & Wojciech Rytter (1996): *Randomized Efficient Algorithms for Compressed Strings: The Finger-Print Approach (Extended Abstract)*. In: *7th CPM 96*, *Lecture Notes in Computer Science* 1075, Springer, pp. 39–49, doi:10.1007/3-540-61258-0_3.

[11] Artur Jez (2012): *Faster Fully Compressed Pattern Matching by Recompression*. In: *ICALP (1)*, *Lecture Notes in Computer Science* 7391, Springer, pp. 533–544, doi:10.1007/978-3-642-31594-7_45.

[12] Marek Karpinski, Wojciech Rytter & Ayumi Shinohara (1995): *Pattern-matching for strings with short description*. In: *CPM '95*, *LNCS* 937, Springer-Verlag, pp. 205–214, doi:10.1007/3-540-60044-2_44.

[13] Jordi Levy, Manfred Schmidt-Schauß & Mateu Villaret (2006): *Bounded Second-Order Unification is NP-complete*. In: *Term Rewriting and Applications (RTA-17)*, *LNCS* 4098, Springer, pp. 400–414, doi:10.1007/11805618_30.

[14] Jordi Levy, Manfred Schmidt-Schauß & Mateu Villaret (2008): *The Complexity of Monadic Second-Order Unification*. *SIAM J. of Computing* 38(3), pp. 1113–1140, doi:10.1137/050645403.

[15] Yury Lifshits (2007): *Processing Compressed Texts: A Tractability Border*. In: *CPM 2007*, *LNCS* 4580, Springer, pp. 228–240. Available at http://dx.doi.org/10.1007/978-3-540-73437-6_24.

[16] Markus Lohrey (2012): *Algorithmics on SLP-compressed strings. A survey*. *Groups Complexity Cryptology* 4(2), pp. 241–299, doi:10.1515/gcc-2012-0016.

[17] Markus Lohrey, Sebastian Maneth & Manfred Schmidt-Schauß (2009): *Parameter Reduction in Grammar-Compressed Trees*. In: *12th FoSSaCS*, *LNCS* 5504, Springer, pp. 212–226, doi:10.1007/978-3-642-00596-1_16.

[18] Markus Lohrey, Sebastian Maneth & Manfred Schmidt-Schauß (2012): *Parameter reduction and automata evaluation for grammar-compressed trees*. *J. Comput. Syst. Sci.* 78(5), pp. 1651–1669, doi:10.1016/j.jcss.2012.03.003.

[19] Wojciech Plandowski (1994): *Testing equivalence of morphisms in context-free languages*. In: *ESA 94*, *Lecture Notes in Computer Science* 855, pp. 460–470, doi:10.1007/BFb0049431.

[20] Wojciech Plandowski & Wojciech Rytter (1999): *Complexity of Language Recognition Problems for Compressed Words*. In: *Jewels are Forever*, Springer, pp. 262–272, doi:10.1007/978-3-642-60207-8_23.

[21] Wojciech Rytter (2004): *Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input*. In J. Diaz et. al., editor: *ICALP 2004*, *LNCS* 3142, Springer-Verlag, pp. 15–27, doi:10.1007/978-3-540-27836-8_5.

[22] Manfred Schmidt-Schauß (2005): *Polynomial Equality Testing for Terms with Shared Substructures*. Frank report 21, Institut für Informatik. FB Informatik und Mathematik. Goethe-Universität Frankfurt.

[23] Manfred Schmidt-Schauss (2013): *Linear Pattern Matching of Compressed Terms and Polynomial Rewriting*. Accepted for publication, 2013.

[24] Manfred Schmidt-Schauss & Georg Schnitger (2012): *Fast Equality Test for Straight-Line Compressed Strings*. *Information processing letters*, doi:10.1016/j.ipl.2012.01.008.

[25] Jacob Ziv & Abraham Lempel (1977): *A Universal Algorithm for Sequential Data Compression*. *IEEE Transactions on Information Theory* 23(3), pp. 337–343, doi:10.1109/TIT.1977.1055714.