Matching of Compressed Patterns with Character-Variables*

Manfred Schmidt-Schauß[1]

1 **Goethe-Universität, Frankfurt, Germany**
schauss@ki.informatik.uni-frankfurt.de

Abstract

We consider the problem of finding an instance of a string-pattern s in a given string under compression by straight line programs (SLP). The variables of the string pattern can be instantiated by single characters. This is a generalisation of the fully compressed pattern match, which is the task of finding a compressed string in another compressed string, which is known to have a polynomial time algorithm. We mainly investigate patterns s that are linear in the variables, i.e. variables occur at most once in s, also known as partial words. We show that fully compressed pattern matching with linear patterns can be performed in polynomial time. A polynomial-sized representation of all matches and all substitutions is also computed. Also, a related algorithm is given that computes all periods of a compressed linear pattern in polynomial time. A technical key result on the structure of partial words shows that an overlap of $h + 2$ copies of a partial word w with at most h holes implies that w is strongly periodic.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases matching, grammar-based compression, partial words

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.272

Category Regular Research Paper

1 Introduction

Matching is a widely used operation in computer science: Given a string (term, object)-pattern s with variables, and a string (term, object) t, is there a substitution σ, such that $\sigma(s)$ is a substring (subterm, subobject) of t? There are at least two sources of motivation for considering matching problems: (i) Given a term rewrite system R and a term t, in order to rewrite t it is first necessary to find a substitution σ, and a rule $s \to r$ of R, such that $\sigma(s)$ is a subterm of t. (ii) Given a string t or a data base D, and a pattern s, find and retrieve the substrings of t or the records (objects) in D, respectively, that are matched by s. Here the pattern may be a string, a string with variables, with holes, or a regular expression, or otherwise extended patterns. The task is to design efficient algorithms for the different variants of matching. In the following we sometimes speak of *submatching* in order to emphasize that substrings (subterms) are searched.
Often terms, strings or databases are very large, and are thus represented or stored in a compressed format. This is the motivation to propose and analyze efficient variants of matching, term rewriting or other algorithms that can also be run on the compressed strings

* Partially supported by DFG, grant SCHM 986/9-1

23rd International Conference on Rewriting Techniques and Applications (RTA'12).
Editor: A. Tiwari; pp. 272–287

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

or terms. We do not directly tackle the full problem of compressed term-submatching. The reason is that this turns out to be a nontrivial problem with complex algorithms even for specializations if polynomial-time algorithms are requested: The technical report [24] contains an approach to compressed term submatching and compressed rewriting but has to leave open several issues. In particular wether there is a polynomial-time algorithm for finding a redex for rewriting where the rule as well as the to-be-rewritten term are compressed. Even the complexity-question of term-submatching for left-linear rules was left unsolved. Thus this paper is restricted to matching algorithms on compressed strings where the patterns are linear and variables mainly stand for characters. The result in this paper turns out to be a first step in the construction of a polynomial-time algorithm for finding matches of compressed left-linear rules in compressed terms. Note that compressed (exact) term matching, where s should match the whole term t is known to be polynomial [11].

Potential applications of matching are in information retrieval, in optimizing term rewriting systems on large terms, and in application domains where large strings are processed. Citing the book [6], where algorithms on partial words (string patterns which are linear in the variables) are investigated: "The next generation of research on combinatorics of partial words promises to have a substantial impact on molecular biology, nanotechnology, data communication, and DNA computing". Our algorithm has potential applications in finding DNA-fragments under compression taking care of SNPs (single nucleotide polymorphism), which is a special case of a point mutation.

Well-known compression methods and compressed representations of strings are the LZ77, LZ78 schemes [26, 27], where LZ77 may compress a string of length n into a representation of $O(\log(n))$ size. Other compression mechanisms are straight line programs (SLP), which are context free grammars (in Chomsky normal form) that generate exactly one string. It is well-known that there are efficient translations between LZ77 and SLPs. A prominent result is that SLP-compressed strings can be tested for equality in polynomial time in the size of SLPs using the method of Plandowski [22] (with improvements in [17, 12]). Plandowski's algorithm works top-down in the grammar and keeps the space (and time) requirements small by exploiting periodicities to detect and eliminate redundancies, whereas Lifshits' is a dynamic programming algorithm that memorizes periodicities using arithmetic progressions. Fully compressed matching of compressed strings (finding a string as a substring in another string) can also be decided and computed in polynomial time [14]. An efficient algorithm is given by Lifshits [17], who shows that it can be done in time $O(mn^2)$ where m is the size of the SLP for the pattern and n is the size of the SLP of the string. An algorithm with improved complexity is proposed by Jez [12]. Complexities of further algorithms on SLP-compressed strings are in [18]. There are also generalizations of SLP-compression on two-dimensional texts, see [2]. The SLP-compression scheme is also extended to terms and ranked trees: An application and an analysis for XML-compressions is in [9], and an analysis and application to matching and unification is in [11].

The matching problem for strings is the question, given a pattern string s, whether it occurs in another given (long) string t. Well-known and folklore algorithms scanning the string t and comparing it with s are for example the Knuth-Morris-Pratt and the Boyer-Moore algorithm. An example for a generalization is searching for an instance of a regular expression in a string, which is possible by a polynomial-time algorithm. There are also results for regular string matching in compressed strings, which is also known to be executable in polynomial time [4, 21], where however, the compression schemes LZ78 or LZW ([26, 27]) are used that have a maximum compression ratio of $O(\sqrt{n})$, in contrast to straight line programs (SLP) which have a potential compression ratio up to $O(\log n)$. In [19] the complexity of several variants

of the matching problem of regular expressions in SLP-compressed texts is investigated: for the usual regular expressions with concatenation, union, and star-operator, the matching problem is P-complete [20], whereas for slightly extended regular expressions including integer exponents, the matching problem in an SLP-compressed string is PSPACE-hard [19].
The string-matching problem when both strings s, t are SLP-compressed is called fully compressed matching. It is well known that the fully compressed pattern matching problem can be decided in polynomial time [17, 12]. It is also well-known that if s contains string-variables and s, t are both uncompressed, then exact matching is NP-complete [1]. The same problem restricted to the case where s is linear, but where the pattern may be compressed, can easily be decided in polynomial time by an eager search (see Lemma 5.1). A related paper is [25], where the pattern s is linear, uncompressed and may contain character- and string-variables and where t is SLP-compressed, matching is shown to be in *PTIME*.
In this paper we consider the following string matching problem: The pattern s is SLP-compressed by G_s, s may have at most $O(|G|)$ holes, and t is also a string SLP-compressed by G_t. We use the view that the holes are character-variables (that may be instantiated with characters from the signature). The question is whether there exists a substitution σ (replacing variables by characters), such that $\sigma(s)$ is a substring of t. The methods as mentioned above lead to exponential time algorithms, so a different algorithm is required. As a technical sub-problem we have to analyze structural properties of strings with character-variables, where every variable occurs at most once. Such strings are more or less equivalent to strings with holes (where a hole means a missing or unknown character). This notion is equivalent to *partial words*. Some results on the structure of partial words and simple unification equations on partial words are in [6, 7].

As result (Algorithm 4.1, Theorem 4.5) we describe an algorithm that runs in polynomial time, decides this problem and moreover outputs a polynomial representation of all occurrences of the pattern. A technical result that is required to show polynomiality of the algorithm is a periodicity-lemma (Theorem 3.10): Given a partial word w with h holes, and an overlap of $h + 2$ copies of w where the occurrences are dense enough, we show that w is strongly periodic, together with a formula for the period. As a corollary, we show that if a partial word has two periods p, q, then it has $\gcd(p, q)$ as periods, provided further conditions on p, q, the length $|w|$ and the number of holes hold. We also show that there is a polynomial time algorithm to compute all periods of a compressed partial word (Proposition 4.6).
Also extensions are considered: The algorithm can be extended to the case where s contains character and string-variables, and s contains every variable at most once (Proposition 5.2). Another extension is that the variables in the pattern s may occur several times, and s is compressed. Then a modification of the algorithm runs in polynomial time in the number of variable occurrences, and the size of the SLPs (Proposition 5.3).
The paper is structured as follows. After some preliminaries on string properties and compression, in Section 3 words with holes, periodicity and overlaps of a word with holes with itself are analyzed. In Section 4 a dynamic programming algorithm is described that computes all matching occurrences of one partial word in a string, both compressed. Finally, some potential extensions are discussed in Section 5.

2 On Strings and Compression

We provide some preliminaries for our algorithm and results: strings, overlaps and periodicity, straight line programs (SLP) for compressing strings and several algorithms and operations on SLPs.

2.1 Properties of Strings

If s is a string (also called a word) over a finite alphabet Σ of characters, we write $s[i]$ for the symbol of s at position i, where we assume that $s = s[0] \dots s[n-1]$, if $n = |s|$, where $|s|$ denotes the length of s. The substring of s from positions i to j is denoted as $s[i..j]$. We also use t^k for a k-fold concatenation of t with $k \geq 0$, where $t^0 = \varepsilon$ is the empty string.
If for a string s there is a number $1 \leq p < |s|$ such that $s[i] = s[i+p]$ for all i, provided $s[i], s[i+p]$ are defined, then we say that s is *periodic with period* p. For strings this is equivalent to: for all i, j: $i \equiv j \bmod p$ implies $s[i] = s[j]$ for all $0 \leq i, j \leq |s| - 1$.

▶ **Theorem 2.1.** *(Fine and Wilf [10]) Let s be a word with periods p and q. If $|s| \geq p + q - gcd(p, q)$, then s has a period $gcd(p, q)$.*

A string w is a *cyclic permutation* of another string w', iff there are substrings w_1, w_2 of w, such that $w = w_1 w_2$, and $w' = w_2 w_1$.

▶ **Lemma 2.2.** *If w is a p-periodic string, then any two substrings v, v' of w of length $|v| = |v'| = p$ are cyclic permutations of each other.*

The following is easy and can be proved by induction using an Euclidean algorithm.

▶ **Fact 2.3.** *If for two nontrivial strings s_1, s_2, the equation $s_1 s_2 = s_2 s_1$ holds, then there is a nontrivial string t, and natural numbers $m, n \geq 1$, such that $s_1 = t^n, s_2 = t^m$. Thus, $|t|$ is a period of $s_1 s_2$ and also a divisor of $gcd(|s_1|, |s_2|)$.*

▶ **Fact 2.4.** *Let w be a string that overlaps with itself, i.e. $w = w_1 w_2$, and w_2 is a prefix of w. Then w is periodic with a period $|w_1|$.*

We will also use strings with variables. Let us assume that there are two kinds of variables: (i) variables standing for characters and (ii) variables standing for strings. We assume that substitutions respect the kind of variables. In the following, variables are character-variables, if not stated otherwise.

2.2 Compression of Strings and Straight Line Programs

▶ **Definition 2.5.** A *straight-line program (SLP)* G (see [22, 23, 13]) is a context free grammar with the following restrictions: Every nonterminal generates exactly one string, for every nonterminal there is exactly one rule of one of the forms $A \to a$ for $a \in \Sigma$ or $A \to A_1 A_2$, where A, A_1, A_2 are nonterminals, and the grammar is not recursive. The generated string for nonterminal A is denoted as $val(A)$, and the string generated by the start nonterminal is denoted as $val(G)$. The *size* of G is the number of its rules, and the *depth* of G is the depth of the derivation tree of the start nonterminal.

Note that $|val(G)|$ may be as large as $2^{|G|}$, but not larger. Lempel-Ziv compression schemes can be efficiently translated into SLPs [23]. However, SLP are more amenable to general analyses and proofs about efficient algorithms for compressed strings.

▶ **Lemma 2.6.** *(Plandowski [22]) Given two nonterminals A, B of an SLP G, it is decidable in polynomially time in $|G|$, whether $val(A) = val(B)$, where an efficient $O(|G|^3)$-algorithm was proposed in [17], and the recently proposed matching algorithm by Jez [12] further improves the complexity.*

The following operations, extensions and modifications of an SLP can be done efficiently (see [15, 16, 11]).

▶ **Lemma 2.7.** *Let G be an SLP. Then a sequence of m operations can be done in polynomial time $O(poly(|G|, m))$, where poly is a polynomial and the operations may be the following:*

1. *Given a nonterminal A, and a prefix (or suffix) w of $val(A)$, extend the SLP to G' such that there is a nonterminal B with $val(B) = w$. In this case the depth of the SPL G' has the same depth as G.*
2. *Given nonterminals A_1, A_2, extend the SLP to G' by $B \to A_1A_2$, where B is a fresh nonterminal.*
3. *Given a nonterminal A, and a number $n \leq 2^{|G|}$, extend the SLP to G' such that there is a nonterminal B with $val(B) = val(A)^n$.*

Note that the complexity is not just $m * poly'(|G|)$, since the extensions iteratively increase the size of intermediate grammars G.
Mainly based on Plandowski's result [22], the following holds:

▶ **Lemma 2.8.** *Let G be an SLP. Then the following can be performed in polynomial time:*

1. *Given a nonterminal A, compute the length $|val(A)|$.*
2. *Given two nonterminals A, B, check whether $val(A)$ is a prefix (suffix) of $val(B)$.*

2.3 Equality of Compressed Strings

▶ **Lemma 2.9.** *Equality up to renaming of variables of two compressed strings with variables can be tested in polynomial time.*

Proof. Let A, B be the nonterminals that represent the two strings that are to be compared, where we assume for simplicity that there are no nonterminals shared in the derivation of A and B. First we normalize the variable names and then we apply a compressed-equality test. The normalization is top down in the grammar: For every variable x identify its leftmost occurrence in $val(A)$, and then replace it by y_i, where i is the number of different variables occurring to the left of this occurrence in $val(A)$, and where y_i is a unique name that is used by the normalization. The y_i can be seen as variables or as constants not occurring in $val(A) \cup val(B)$. Then do the same for B. The number of variables is at most $|G|$, and for every variable the normalization can be done in polynomial time. Note that there is no size-change of G. Then use the equality-test for compressed strings. ◀

▶ **Lemma 2.10.** *Let s be a string with character variables and t be a string, both compressed with an SLP G. Then it is decidable in polynomial time whether there exists σ, such that $\sigma(s) = t$.*

Proof. Let there be $n < |G|$ variables in s. It is sufficient to compute a single position p_i (say the leftmost one) for every variable x_i in s for all $i = 1, \ldots, n$. Then compute the character a_i at position p_i of t. and perform the equality test of $[a_1/x_1, \ldots, a_n/x_n]s = t$. ◀

2.4 Submatching

▶ **Definition 2.11.** The *compressed character-submatch (CCSM)* problem for strings is defined as follows. Let Σ be an alphabet, let G be an SLP over Σ, let $t = val(T)$ where T is a nonterminal of G, let $s = val(G')$ where G' is an SLP over $\Sigma \cup V$, where V is a set of (character-)variable symbols.
Question: is there a substitution $\sigma : V \to \Sigma$ such that $\sigma(s)$ is a substring of t? If every variable occurs at most once in s, then it is called the *linear compressed character submatch (LCCSM)* problem.

Note that if σ can replace variables by strings, then even the following question is NP-complete: Given uncompressed s, t, does there exists σ, such that $\sigma(s) = t$?
The following observations for special cases are easy to verify:

- CSSM is in NP: guess the substitution σ and then use fully compressed pattern matching.
- If there is only one variable x, then CSSM is in *PTIME*: try all possibilities for σ.
- If there is a given number k that is an upper bound for $|val(s)|$, then CSSM is solvable in polynomial time where k occurs in the exponent of the polynomial.

2.5 Remarks on Comparison with Other Approaches and Result

approximate matching with edit-distance. This is the question whether a variant s' of the string s occurs in t, where s' and s differ by a form of edit-distance. This is also called approximate matching. The investigations in the literature usually take a weaker form of compression: LZ78, LZW, and the pattern s is not compressed. If s is not compressed, then the approximate matching will also produce the matching solutions in case s has character holes, but perhaps more, depending on the exact definition of edit-distance used [4].

regular expression matching. Compression by SLP cannot be represented using a regular expression. If the pattern has the form $s = s_1 x_1 \dots x_n s_{n+1}$ and x_i are character-variables or string variables, and s_i are uncompressed and ground (no variables), then this can be translated into a regular expression match question. If the regular expression syntax also allows exponents w^k for an integer k and a single string w, then it can also cover special forms of compression, but not the general form. However, this syntax extension is not discussed in [19]. It is not hard to see that in this case matching in SLP-compressed strings is NP-hard (using NP-hardness of SUBSETSUM and exploiting the union-operator).

3 The Linear Case: Partial Words

We mainly consider the linear compressed character submatch problem (LCCSM). A string where every variable occurs at most once can also be seen as a string with several holes where the holes represent single missing or unknown characters. We denote the hole with $\circ$, or with variable names x, y, z, depending on the context. In the literature these are also called *partial words*, see for example [8, 5, 6].
If for a partial word s there is a number $p < |s|$ such that $s[i] = s[i+p]$ for all i, and $s[i], s[i+p]$ are defined and are not holes, then we say that s is *p-locally periodic* with period p. For example, the partial word $ababa\circ acac$ is 2-locally periodic. If there is a number $p < |s|$ such that $i \equiv j \bmod p$ implies $s[i] = s[j]$ for all $0 \le i, j \le |s| - 1$ if $s[i], s[j]$ are defined and not holes, then s is called *periodic* (also strongly periodic), and p is a *period* of s. The partial word $ab\circ bababa$ is 2-periodic. Note that p-periodic implies p-locally periodic, but the converse might not hold, for example $ab\circ bb$ is 2-locally periodic, but not 2-periodic. Note that p-periodic partial words can be made p-periodic strings by substituting characters into the holes, whereas this is not the case for p-locally periodic partial words. We write $s =_h s'$ for two partial words s, s' if $|s| = |s'|$ and s, s' are equal up to the occurrences of holes. Note that $=_h$ is not transitive, for example $a\circ =_h \circ b$, $\circ b =_h b\circ$, but $a\circ \neq_h b\circ$.
Equality $=_h$ of two partial words w, w' that are SLP-compressed, and if holes are represented as variables, can be checked polynomially in the size of the SLP and the number of holes of w, w'. If holes were an extra symbol in the SLP, and there is no restriction on the number of occurrences, then we could not find a polynomial algorithm to decide $=_h$: adapting the

Plandowski-style algorithms is not possible, since periodicity lemmas fail in this case. The only trivial information is that the problem is in co-NP.

3.1 Partial Words with One Hole

First we investigate some properties of (uncompressed) partial words with one hole. In the following we fix an alphabet Σ. The inner structure of a p-locally periodic string is clarified:

▶ **Lemma 3.1.** *Let s be a partial word with one hole at position r, and let s be p-locally periodic. Then s is either p-periodic, or the following holds:*

- *For all $i, j : i \equiv j \bmod p$ and $i \not\equiv r \bmod p$ it holds that $s[i] = s[j]$, if $s[i], s[j]$ is defined;*
- *$s[0..r]$ as well as $s[r..|s|-1]$ are p-periodic.*

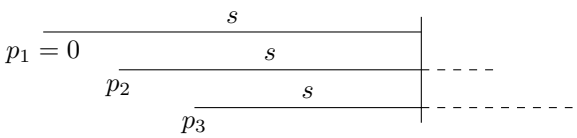
An example for such a partial word is $ababa\circ acac$, which is 2-locally periodic, but not 2-periodic, where $\circ$ represents the hole.
There are generalizations of the Fine and Wilf-theorem to partial words:

▶ **Theorem 3.2.** *([3, 8]) Let s be a partial word with one hole. If s is p-locally periodic and q-locally periodic, $p \neq q$, and $|s| \geq p + q$, then s is also $gcd(p, q)$-periodic.*

The bound $|s| \geq p + q$ is sharp [3, 8].

▶ **Definition 3.3.** Let s be a partial word and $n \geq 2$. An n-fold *overlap* of s (with itself) is given by starting positions $0 \leq p_1 < p_2 < p_3 < \ldots < p_n \leq |s| - 1$, such that for all $i, j = 1, \ldots, n$ and $0 \leq k \leq |s| - 1$: if $0 \leq k - p_i, 0 \leq k - p_j$, then $s[k - p_i] = s[k - p_j]$, provided neither $s[k - p_i]$ nor $s[k - p_j]$ is a hole.
In the following we let $p_1 = 0$, if not stated otherwise.

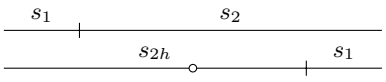
The overlap of s with itself intuitively is only meant in the range of the left most occurrence, left of the vertical line. There are no overlap conditions right of the vertical line. The reason for this choice is our application to overlapping prefixes below. For example, for a 3-overlap, equality of characters must hold on every position of the topmost occurrence of s.

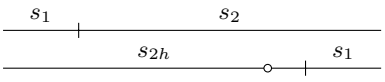


A simple example of an overlap is $s = ab\circ baba$, $n = 2$ and $p_2 = 2$. Another overlap is $s = a\circ bbb$, where $n = 2$ and $p_2 = 1$.
The following shows that the periodicity claim in Fact 2.3 also holds if one of s_1, s_2 is a partial word with one hole, and the other one is a word without holes.

▶ **Lemma 3.4.** *Let s_1, s_2 be nontrivial strings, and let $s_{i,h}, i \in \{1, 2\}$ be partial words with a single hole such that $s_{i,h} =_h s_i, i \in \{1, 2\}$. If $s_1 s_2 =_h s_{2,h} s_1$, or if $s_1 s_2 =_h s_2 s_{1,h}$, then $s_1 s_2$ is periodic with a period $gcd(|s_1|, |s_2|)$.*

Proof. We use induction on $|s_1 s_2|$. The base case is $|s_1| = |s_2|$. In this case each equation implies that $s_1 = s_2$, and the period has length $|s_1| = gcd(|s_1|, |s_2|)$.
Let $|s_1| \neq |s_2|$. We first consider the case $s_1 s_2 =_h s_{2,h} s_1$, the other case is symmetric. We distinguish several cases:





- $|s_2| > |s_1|$ and the hole is at a position $\leq |s_2| - |s_1|$.
  Then let s_2' be such that $s_2's_1 = s_2$. Let $s'_{2,h}$ such that $s'_{2,h}s_1 := s_{2,h}$. The induction hypothesis can be applied to $s_1s_2' =_h s'_{2,h}s_1$, and shows that a period has length $gcd(|s_1|, |s_2|) = gcd(|s_1|, |s_2'|)$, and it is also the period of s_1s_2.
- $|s_2| > |s_1|$ and the hole is at a position $\geq |s_2| - |s_1|$.
  Then let s_2' be such that $s_2's_1 = s_2$. Let $s'_{1,h}$ be such that $s_{2,h} =_h s_2's'_{1,h}$. The induction hypothesis can be applied to $s_1s_2' =_h s_2's'_{1,h}$, and shows that a period has length $gcd(|s_1|, |s_2|) = gcd(|s_1|, |s_2'|)$, and it is also the period of s_1s_2.
- $|s_1| > |s_2|$. This can be proved similarly as above where only the occurrence of the hole is in different partial words. ◀

Lemma 3.4 and Theorem 3.2 imply:

▶ **Corollary 3.5.** *Let s_1, s_2 be nontrivial strings, such that s_1s_2 is periodic with period p, $p \leq 0.5|s_1s_2|$, and let $s_{i,h}, i \in \{1, 2\}$ be partial words with a single hole such that $s_{i,h} =_h s_i, i = 1, 2$. If $s_1s_2 =_h s_{2,h}s_1$ or $s_1s_2 =_h s_2s_{1,h}$, then s_1s_2 is periodic with a period $gcd(|s_1|, |s_2|, p)$.*

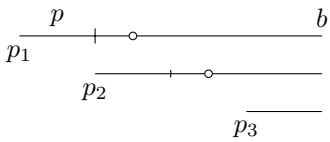
▶ **Lemma 3.6.** *Let w be a partial word with one hole, and assume that there is a 2-overlap of w, starting at p_1, p_2. Then w is locally periodic with period $p_2 - p_1$.*

Proof. The overlap immediately implies that w is locally periodic with period $p_2 - p_1$. Lemma 3.1 provides information about the exact structure of w. ◀

The following lemma shows a part of Lemma 3.9, which is the the base case of the induction proof in Theorem 3.10.

▶ **Lemma 3.7.** *Let w be a partial word with one hole, and assume that we have a 3-overlap of w, starting at $0 = p_1 < p_2 < p_3$. Let $p = p_2 - p_1$ and assume that $|w| - p_3 \geq p$, i.e., the last p positions are common to all the three occurrences of w. Then the partial word w is periodic, and a period is $gcd(p_2 - p_1, p_3 - p_2)$. Moreover, the overlap is consistent with using the same substitution for the single variable for every occurrence of w.*

Proof.



For the occurrences starting at p_1, p_2, Lemma 3.6 implies that w is locally periodic with period $p = p_2 - p_1 = p_2$. Also, if t_1, t_2 are substrings of length p of w, then either t_1, t_2 are cyclic permutations of each other, or cyclic permutations up to the occurrence of one character. Note that the overlap does not imply that w is $(p_3 - p_2)$-locally periodic, since there is a cut at b, hence Theorem 3.2 is not applicable.

- Let $b := |w| - 1$. If the third occurrence does not contain a hole in the overlap, i.e. $w[0..b - p_3]$ does not contain a hole, then the string $w[b - p_3 - p + 1..b - p_3]$ is equal to a substring of length p of the first or of the second occurrence, without hole. This equality implies that there are two equal substrings t_1', t_2' of w, both of length p, where t_1' is left of the hole and t_2' is right of the hole. By Lemma 3.6, this is only possible if w is periodic with period p.
- If the third occurrence contains a hole in the overlap, then there are two subcases: (i) $r \geq p$, where r is the hole position in w. Then let t_1 be the substring $a[r - p_3 - p..r - p_3 - 1]$, which is the substring of the third occurrence immediately left to the hole. Due to the overlap this is equal to a substring t_1 of length p in the first occurrence of w, which is a substring in w right of the hole. Again, this implies that w is periodic with period p.
  (ii) $r < p$, i.e. the hole is in the first p-period of w. Then w is periodic with period p.

Now it remains to show that also $p_3 - p_2$ is a period of w. If p is a divisor of $p_3 - p_2$, then there is nothing to show, so let us assume that p is not a divisor of $p_3 - p_2$. Since w is already periodic with period p, it is sufficient to show that a substring of w of length p is also periodic with period $p_3 - p_2$, and then apply Theorem 3.2. The prefix of the third occurrence, i.e. $w[0..p-1]$ overlaps a part of the first occurrence: $w[p_3..p_3 + p - 1]$. Either $w[0..p-1]$ contains a hole and $w[p_3..p_3 + p - 1]$ does not contain a hole, or vice versa. Moreover, due to p-periodicity of w, there are nontrivial s_1, s_2, s_1', s_2' such that $s_1 s_2 = w[0..p-1]$ and $s_2' s_1' = w[p_3..p_3 + p - 1]$ where $|s_1| = |s_1'|$, $|s_2| = |s_2'|$, and $|s_2| \equiv p_3 - p_2 \mod p$. Also, depending on where the hole is, we have either

- $s_1 s_2 = s_2' s_1'$, $s_1 = s_1'$, $s_2 = s_2'$, or
- $s_1 s_2 =_h s_2' s_1'$, $s_1 = s_1'$, $s_2 =_h s_2'$, or
- $s_1 s_2 =_h s_2' s_1'$, $s_1 =_h s_1'$, $s_2 = s_2'$

Then Fact 2.3 shows the claim on periodicity in the first case, and Lemma 3.4 shows the claim on periodicity in the other cases. ◀

▶ **Remark 3.8.** *The number of common overlap positions in Lemma 3.7 is sharp. For example, let $w = bab\diamond bcb$, and let $p_2 = 2, p_3 = 6$ with $|w| = 7$ be an overlap. Then only one character position is common to all the three occurrences. The string $bab\diamond bcb$ is not 2-periodic.*

The following lemma is the base case of the induction proof in Theorem 3.10.

▶ **Lemma 3.9.** *Let w be a partial word with one hole, and assume that we have an overlap of $k \geq 3$ occurrences of w starting at $0 = p_1 < p_2 < \ldots < p_k$, respectively. Let $p = p_2 - p_1$ and assume that the last p positions, i.e., $|w| - p, \ldots, |w| - 1$ are common to all the k occurrences. Then the partial word w is periodic, and a period is $gcd(p_2 - p_1, p_3 - p_2, \ldots, p_k - p_{k-1})$.*

Proof. This follows by a repeated application of Lemma 3.7 and Corollary 3.5 as follows, by induction on the number n of overlaps:
Assume that for $j \geq 3$ it is shown that the period is $gcd(p_2 - p_1, p_3 - p_2, \ldots, p_j - p_{j-1})$. Then consider the next occurrence of the partial word w, and focus on the overlap of the occurrence $j + 1$ of w at positions $|w| - p, \ldots, |w| - 1$ with the first or second occurrence of w. We take the first one, if there is no hole in the interval $w[|w| - p..|w| - 1]$, otherwise we take the second one. Due to periodicity with period p, this leads to one of the equations $s_1 s_2 = s_2 s_1$, or $s_1 s_2 =_h s_{2,h} s_1$ or $s_1 s_2 =_h s_2 s_{1,h}$ with $s_i =_h s_{i,h}$ for $i = 1, 2$, and $|s_1| \equiv p_j \mod p$. Simple computation with *gcd* now shows the claim. ◀
Now we are in a position to perform an induction proof for overlaps of any number of occurrences of a partial word with several holes.

▶ **Theorem 3.10.** *Let w be a partial word with n holes, and assume there is an overlap of $m \geq n + 2$ occurrences of w, starting at $p_1 < p_2 \ldots < p_m$. Let p_{max} be $\max\{p_{i+1} - p_i \mid i = 1, \ldots, m-1\}$, and $|w| - p_m \geq 2n \cdot p_{max}$; this means there are $2n \cdot p_{max}$ common positions of all occurrences of w.*
Then the partial word w is periodic, and a period is $p_{all} := gcd(p_2 - p_1, p_3 - p_2, \ldots, p_m - p_{m-1})$. There is a unique string w' that can be overlapped with w, starting at m ending at $|w| - 1$, that is periodic with the same period. Moreover, the overlap is consistent with using the same substitution for every occurrence of w.

Proof. By induction on the number of holes. If w has one hole (i.e. $n = 1$), and there are $m \geq n + 2 = 3$ occurrences, then the claim follows from Lemma 3.9.

Now assume that there are $n > 1$ holes in w. Let r be the position of the leftmost hole of w, and let w' be a partial word, v_1 be a string, such that $w = v_1 \circ w'$. There are two cases:

1. $r < 2p_{max}$. The overlap of $m \geq n+2$ occurrences of w' is constructed from the given overlap by cutting away all prefixes of length $r+1$. For convenience, we have different starting positions $p'_i := p_i + r + 1$. The number of holes is decreased by one, the maximum p_{max} is unchanged, and the condition $|w'| - p_m + 1 \geq 2n' \cdot p_{max}$ with $n' = (n-1)$ holds. Now we can use induction and obtain that w' is periodic with the period as claimed. It remains to show that this claim can be transferred to the full partial word w: We look at the prefix w_1 of the m^{th} (the last) occurrence of w. This string w_1 is positioned such that there are at least $n+1$ other occurrences of w with a common position. For the position $q = p_m + r - 1$ in the prefix v_1 of the m^{th} occurrence of w, there is at least one index $j(q)$, such that $w_{j(q)}[q - p_{j(q)+1}]$ is not a hole. Thus this character is determined by the overlap. We scan the positions from greater indices to smaller ones and observe that for one period p_{all} the suffix $w_{1,1}$ of w_1 is determined. It is also obvious that the partial word $w_{1,1} \circ w'$ is p_{all}-periodic. Now we repeat this process of determining w_1 period-wise and finally obtain that w is p_{all}-periodic.
2. $r \geq 2p_{max}$. This implies $|v_1| \geq 2p_{max}$. The length condition shows that in the overlap of the m occurrences, there are overlaps of v_1 with itself for all the indices $i, i, +1$ where the shift in the overlap is $p_{i+1} - p_i$, and the overlapping part is longer than $|v_1|/2 \geq p_{max}$. Fact 2.4 shows that v_1 is periodic with period $p_{i+1} - p_i$ for all i, and since $|v_1|/2 \geq p_{i+1} - p_i$ for all i, we obtain that v_1 has period p_{all} by Theorem 2.1. Since the copies of v_1 cover the overlap for all positions $0..p_m + |v_1| - 1$, we have already periodicity of $w[0..p_m + |v_1| - 1]$ with period p_{all}. Similar as for the first item, we obtain also for all further positions of w, that these satisfy the periodicity condition by iterating the following step: determine the next p_{all} characters of w using the observation that there is an overlap of at least $n+2$ occurrences and that the focussed positions are in the part where all occurrences overlap. Since at least one occurrence has a character at the questioned position, we see that the next p_{all} characters are determined and satisfy the period condition. Iterating this implies that the periodicity condition holds for w.

Since $m \geq n+2$, the string starting at m to $|w| - 1$ that overlaps all occurrences of w is uniquely defined and is periodic with the same period due to the previous arguments. The periodicity of w, and since w contains hole-free substrings of lengths greater than the period implies that the substitutions can be chosen consistently for all occurrences. ◀

▶ **Corollary 3.11.** *Let w be a partial word with $n \geq 2$ holes, let w be periodic with periods $p, q \leq |w|/(3n)$. Then w is also* $\gcd(p,q)$*-periodic.*

Proof. The periodicity assumptions imply that there is an overlap of w with $p_i = ip$ as well as $p_i = iq$ for $i = 0, 1, 2, \ldots$. The upper bound permits to apply Theorem 3.10: Assume $p < q$ and that q is not a multiple of p. Then in the interval $0..|w|/(3n)$ at least $n+2$ different copies of w are starting. The prerequisite of the theorem holds: $|w| - np \geq 2n \cdot p$. The derived period is $\gcd(p, q-p)$, which is the same as $\gcd(p,q)$. ◀

The corollary is too weak for $n = 1$, since Theorem 3.2 provides a better bound, which indicates that there is space for improvements in the case $n \geq 2$.

4 A Matching-Algorithm Using Dynamic Programming

Let G_s, G_t be SLPs for s and t, respectively, where s is a partial word with n holes, and t is a string, and S is a nonterminal with $val(S) = s$. First we build two tables, a *prefix-table* and

a *result-table* as follows: The coordinate is T, for the nonterminals of G_t. Entries in the lists may be (i) positions a, (ii) arithmetic progressions, represented by triples (a, b, m), where a is the start position, b is the step length, and m is the number of steps, (iii) arithmetic sequences without upper bound, represented by pairs (a, b). The result table contains a list with entries a or (a, b, m), whereas the prefix table contains a list with entries a or (a, b). These entries have the following semantics:

- a represents that a prefix of $val(S)$ is a suffix of $val(T)$ where the suffix starts at position a in $val(T)$.
- (a, b, m) represents prefixes of $val(S)$ as suffixes of $val(T)$, where the starts of the S-suffixes are at positions $a + ib$ for $i = 0, \ldots, m - 1$ in $val(T)$.
- (a, b) is the same as (a, b, m) for the maximal possible m. Moreover, $val(T)$ has a periodic suffix: $val(T)[a..|val(T)| - 1]$ is periodic with period b. The period string is $val(T)[a, a + b - 1]$.

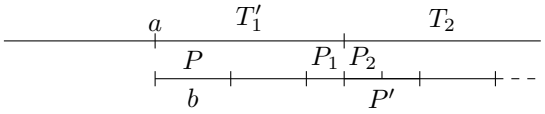
▶ **Algorithm 4.1.** *The construction of the tables is as follows and works bottom-up in G_t, where we fix S. For a nonterminal T with rule $T \to T_1T_2$, we assume that the prefix tables for T_1 and T_2 are already constructed. Then the prefix table of T will be constructed using the entries in the prefix tables of T_1 and T_2. If the entries for T are constructed, then a compaction step is applied to all entries for T. The result-table is like an output and will not be used for further constructions. For every nonterminal T, the following is performed:*

Simple cases:

1. *If $|val(T)| = 1$: If $|val(S)| > 1$, and $val(S)$ has $val(T)$ as prefix, or if $val(S)$ starts with a hole, then there is an entry 0 in the prefix-table of T, otherwise there is no entry in the prefix table. If $|val(S)| = 1$, and $val(S)$ has $val(T)$ as prefix, or if $val(S)$ starts with a hole, then the result-table entry is $(0, 1)$, otherwise there is no entry.*
2. *Let T be a nonterminal with rule $T \to T_1T_2$.*
   a. *All entries a, (a, b) in the prefix table of T_2 are inherited to the prefix table of T as $a + q$ and $(a + q, b)$, respectively, where $q = |val(T_1)|$.*
   b. *An entry a of the prefix table of T_2 is tested by first adding a nonterminal A representing the prefix of $val(S)$ as suffix of $val(T_1)$, then $A' \to AT_2$ is added to the SLP. If $val(A')$ is a proper prefix of $val(S)$, then a is added to the prefix table of T. If $val(S)$ is a prefix of $val(A')$, then a is in the result table of T. Otherwise, no entry is inherited.*

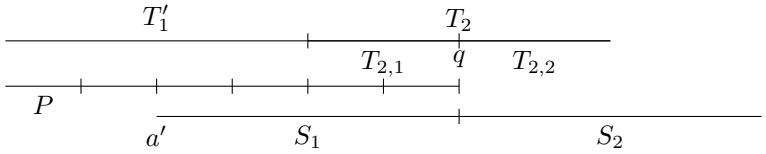
The complex case: *Let T be a nonterminal with rule $T \to T_1T_2$, and let there be an entry (a, b) in the list of the prefix table of T_1.*

Let T_1' be a nonterminal that represents the suffix of $val(T_1)$ starting at a. We determine the maximal number of periods corresponding to entry (a, b) that fit as prefix into $val(T_2)$. Therefore construct the potential period string as a nonterminal P as representing the string $val(T_1)[a, a + b - 1]$. Then construct nonterminals P_1 as a prefix and P_2 as a suffix of P, such that $val(P) = val(P_1P_2)$ and $val(T_1)[a..|val(T_1)| - 1] = val(P)^k val(P_1)$ for some k. Then construct P' as $P' \to P_2P_1$, and its powers on demand and use interval bisection to find the maximal prefix of $val(T_2)$ that is periodic with period string P'.



A similar computation has to be done for S: using interval bisection in order to find the maximal number k' of periods such that a prefix of $val(S)$ matches $val(P)^{k'}$. The possible case and corresponding outcomes and actions are:

1. *$val(T_2)$ is periodic with period string $val(P')$, and $val(S)$ has a proper prefix that matches the periodic string $val(T_1'T_2)$. The new entry for the prefix-table for T is (a, b). There will be no entry for the result-table.*
2. *$val(T_2)$ is periodic with period string $val(P')$, and $val(S)$ matches a prefix of $val(P)^k$, and item (1) does not hold. This means $val(S)$ is periodic as a partial word with period string $val(P)$.*
   Compute the number k by comparing the length of $val(S)$ with the length of $val(T_1'T_2)$ such that the arithmetic sequence $a, a+b, \ldots, a+(k-1)b$ represents the positions in $val(T_1'T_2)$ where S matches substrings of $val(T_1'T_2)$. The entry (a, b, k) is added to the result-table of T. The pair $(a+bk, b)$ is added as entry to the prefix-table of T.
3. *$val(T_2)$ is periodic with period string $val(P')$, and $val(S)$ has a proper prefix that matches $val(P)^m$ for some m, but not the the full periodic substring $val(T_1'T_2)$. Then compute the smallest k by simple arithmetic, such that $a+bk$ is position in $val(T_1'T_2)$, such that S matches the substring starting there. The entry $(a+bk, b)$ is added to the prefix-table of T. There is no result entry added.*
4. *$val(T_2)$ has a proper prefix that is periodic with string $val(P')$, but $val(T_2)$ is not periodic with period string $val(P')$, and $val(S)$ matches a prefix of $val(P)^k$ for some k.*
   This means $val(S)$ is periodic as a partial word with period string $val(P)$.
   Compute the number k using arithmetic, and interval bisection for $val(T_1'T_2)$, such that the arithmetic progression $a, a+b, \ldots, a+(k-1)b$ represents the positions in $val(T_1'T_2)$ where S matches substrings of $val(T_1'T_2)$. The entry (a, b, k) is added to the result-table of T. There are no entries in the prefix-table of T.
5. *$val(T_2)$ has a proper prefix that is periodic with string $val(P')$, $val(T_2)$ is not periodic with period string $val(P')$, and $val(S)$ does not match a prefix of $val(P)^k$ for any k.*



   Then compute nonterminals S_1, S_2 extending the SLP, such that $val(S_1)$ is the maximal prefix of $val(S)$ that matches a prefix of $val(P)^m$ for some m, and $val(S) = val(S_1S_2)$. Also compute $T_{2,1}, T_{2,2}$ extending the SLP with $val(T_2) = val(T_{2,1})val(T_{2,2})$, and such that $val(T_{2,1})$ is the maximal prefix of $val(T_2)$ that is periodic with period string $val(P')$. Then there is at most one potential position for a matching occurrence (derived from the entries in T_1): $val(T_{2,2})$ and $val(S_2)$ must start at the same position. Let the position q be computed as $q := a + |val(T_1T_{2,1})|$. If $val(S_2)$ matches a prefix of $val(T_{2,2})$ or there is a prefix of $val(S_2)$ that matches $val(T_{2,2})$, and $val(S_1)$ matches a suffix of $val(T_1'T_{2,1})$, then we can proceed; otherwise, there is no entry. Let a' be the starting occurrence of $val(S)$ in $val(T_1'T_2)$, i.e. $a' := q - |val(S_1)|$.
   If S matches $val(T_1T_2)$ at position a' and is completely contained, then a' is an entry in the result-table for T. If a proper prefix of S matches $val(T_1T_2)$ at position a' and is not completely contained, then a' will be an entry in the prefix-table of T.

Compaction *Finally, there is a compaction of the prefix table of T to keep the sum of the sizes of all prefix-tables polynomial. The compaction takes the prefix table as computed above as input and generates a completely new prefix table for T. Thereby it may generate also entries for arithmetic progressions in the prefix-table.*
Let h be the number of holes of $val(S)$. Let c be the current position in $val(T)$, where the

start is at position 0*, and where the compaction stops if* $|val(T)| - c < h + 3$ *and the input positions are simply moved to the output in this case.*
Let $d := |val(T)| - c + 1$ *and let* $e := \lfloor d/(2(h+2)) \rfloor$*. If for* $val(T)$ *in the (position) interval* $c, c+e-1$ *there are at least* $h+3$ *positions for* S *including implicit positions from arithmetic progression, then the leftmost* $h+2$ *entries in this interval are expanded, and* b' *is computed as the gcd of the position differences according to Theorem 3.10. If* p' *is the* $(h+3)^{th}$ *position, then the entry* (p', b') *is added to the output for* T*.*
Then do the same for all intervals $[c+ie, c+(i+1)e-1]$ *for* $i \in \{1, \ldots, h+1\}$*. This covers the left half of* $val(T)[d..|val(T)|-1]$*. Continue with the same procedure for the right half of* $val(T)$*, but with refreshed* c, d, e*.*

The two tables are a complete description of all occurrences of prefixes of $val(S)$ as suffixes of $val(T)$ and of matching occurrences of S in T. The matching occurrences can be found in the result table of minimal nonterminals: If $T \to T_1T_2$, then only the occurrences of S in T that cross the border between $val(T_1)$ and $val(T_2)$ are mentioned in the result-table of T. Other occurrences are mentioned at T_1, T_2 or their descendents.

▶ **Proposition 4.2.** *Let* S *be a nonterminal in* G_s *and* T *be a nonterminal in* G_t*, and assume the two tables are constructed with Algorithm 4.1. Then the prefix-table is a complete description of all occurrences of prefixes of* $val(S)$ *as suffixes of* $val(T)$*, and the result-tables of the nonterminals contributing to* T *(including* T*) are a complete description of all matching occurrences of* S *in* T*.*

Proof. The argument is by induction: Let $T \to T_1T_2$ be a rule in G_t: We assume that the prefix-entries for T_1 and T_2 are complete, and then have to argue that the step of Algorithm 4.1 constructs a complete set of entries for T. This requires to verify that all the cases are treated. This is done in the algorithm due to the exact knowledge about periodicities. ◀

▶ **Proposition 4.3.** *The compaction step is correct. Also the semantics of the entries satisfied: for a periodic entry* (a, b)*, the suffix of* $val(T)$ *is periodic with period* b*. Moreover, the period is the optimal one, since the input contains all matching occurrences for prefixes of* $val(S)$*.*

Proof. This follows from Theorem 3.10. ◀

▶ **Proposition 4.4.** *The construction of the prefix- and result-table requires at most polynomial time in the size of* $G := G_s \cup G_t$*. For a fixed* S*, the size of the tables is of order* $O(|G|^4)$*.*

Proof. The operations can all be done in polynomial time, and the table has a polynomial number of places (see Lemma 2.7). What remains to be shown is that the number of entries in the list for nonterminal T is polynomial. Let n be the size of the SLP G and h be the number of holes of $val(S)$, and assume that the rule for detection of arithmetic sequences does not apply. Then we can determine an upper bound for the number of entries as follows: Here a single entry and a pair for arithmetic sequences count as one entry. Let $d = |val(T)|$, and let $e = d/(2(h+2))$. In the interval $[0, e]$ at most $h+3$ are starting, since otherwise a compaction step is possible. The same for all the intervals until $[(h+1)e, (h+2)e]$. Thus there are at most $(h+2)(h+3)$ entries in the left half of T. The same argument applies to the right half in the role of T. Due to the generation of strings by an SLP not more than n interval divisions by 2 are possible, hence at most $n * (h+2)(h+3)$ entries are in the table for the nonterminal T. ◀

4.1 Results for Matching of Strings with Holes

▶ **Theorem 4.5.** *The LCCSM problem can be solved in polynomial time: If s is a string with n holes, and t is a string, compressed by SLPs G_s and G_t, respectively, then a representation of all the matching occurrences of s in t can be computed in polynomial time.*

Proof. Using Algorithm 4.1 for the construction of the result-table is complete and requires polynomial time in $|G_t| + |G_s|$ by Proposition 4.4. ◀

Note that the number of matching occurrences may be exponential in $|G_t| + |G_s|$. The computed table represents all occurrences in a compact way in polynomial space.

A representation of all periods of a compressed partial word s can be computed: Let S be a nonterminal with $val(S) = s$. Compute nonterminals $S_i, i = 1, \ldots, m$ with $m \leq |G|$ that represent the maximal hole-free substrings of s. Then compute all the periods of $val(S_i)$ for $i = 1$ using a fully compressed matching algorithm [17], also Algorithm 4.1 could be specialized to this. This results in a polynomial number of potential periods. Now test every potential period p as follows: First compute a nonterminal S' representing S where the holes are instantiated by characters using the period p. Then compute a nonterminal P with $val(P) = val(S)[0..p-1]$ and check whether $val(P^k P') =_h val(S)$ where P' represents a prefix of $val(P)$ and $|val(P^k P')| = |val(S)|$. For every period of $val(S)$, at least one divisor will be found, their number is at most polynomial and every check can be done in PTIME. Note that Algorithm 4.1 cannot be used to find periodicities of S in S (see item 5.3).

▶ **Proposition 4.6.** *All periods of an SLP-compressed string s can be computed in PTIME.*

5 Extensions and Generalizations

5.1 Adding Variables Representing Strings

▶ **Lemma 5.1.** *Let s be a string linear in the variables, every variable is a string-variable, let t be a string, and assume that both are compressed by an SLP. Then the question whether there exists a substitution σ, such that $\sigma(s)$ occurs in t can be decided in polynomial time.*

Proof. Let S, T be nonterminals with $val(S) = s, val(T) = t$. An eager left-to-right search is sufficient: If $val(S) = s_1 x_1 \ldots x_n s_{n+1}$, where s_i are ground, then first construct nonterminals S_i with $val(S_i) = s_i$ for $i = 1, \ldots, n$. Then identify the first occurrence of s_1 in t. The next step is to construct a nonterminal representing the suffix of t right to the occurrence and repeat the search. This is a sequential algorithm, every step can be performed in polynomial time, and the required SLP does not grow in size. Hence whether a match exists can be decided in polynomial time, and a match can be computed as a side-effect. ◀

This can be generalized to a mix of string-variables and character-variables:

▶ **Proposition 5.2.** *Let s be a string linear in the variables, which may be string-variables or character-variables, let t be a string, and assume s, t are SLP-compressed. Then the question whether there exists a substitution σ, such that $\sigma(s)$ occurs in t can be decided in PTIME.*

Proof. Similar to the previous proof: let $s = s_1 x_1 \ldots x_n s_{n+1}$, where x_i are all the string-variables, and s_i may contain character-variables. Then an eager left-to-right search will run in polynomial time, using Theorem 4.5, similar as in the proof of Lemma 5.1. ◀

5.2 Nonlinear Patterns

Suppose, we permit multiple occurrences of the same character-variable in the pattern. Note that the number of different variables is at most $|G_s|$, however, there may be an exponential

number of occurrences of variables in s. Unfortunately, Theorem 3.10 is no longer applicable. If the number of occurrences is polynomial, then we can use our current method as follows:

▶ **Proposition 5.3.** *Let the number of occurrences of variables in s be k. Then the pattern match problem can be decided in time $O(poly_k(|G_s|, |G_t|))$ where poly is a polynomial.*

Proof. Linearize the pattern s giving s', such that $\rho(s') = s$ for some variable-variable substitution. Perform a pattern match using s', resulting in a polynomially-sized result-table. Then scan every entry: For single entries a, test whether s' occurs in t at position a, which is polynomial. For arithmetic sequences of length $< k + 2$ use the method for single entries, otherwise a unique substitution can be computed from the sequence. Then test whether this substitution is compatible with ρ. All the tests can be done in polynomial time. ◀

5.3 Further Possible Extension and Problems

Pattern Target is a Partial Word. If the target t of the pattern match is a partial word, then our method does no longer work as expected, since the overlap-theorem can no longer by applied. For example let $s = aa\circ bb$, and $t = aaa\circ\circ bb$, then s matches t at positions $0, 1, 2$. However, the three occurrences of s do not overlap, and so the overlap theorem cannot be used for compacting the entries in the prefix-table in Algorithm 4.1.

Compressed Partial Words Including Holes If holes are not seen as variables, but represented as an extra symbol, then compression can represent partial words with a larger number of holes, up to an exponential number of holes. However, then our arguments fail, in particular the application of the overlap-theorem 3.10 no longer helps in compacting the prefix-table entries, since the number of holes is no longer $O(|G_s|)$.

Term Matching Looking for submatching (or the encompassment relation) of terms as a generalisation of strings is of course an interesting generalization. The technical report [24] contains an approach to compressed term matching but has to leave open the issue whether a single compressed term submatch can be performed in polynomial time and even the question for terms with linear variable occurrence was left open.

6 Conclusion and Further Work

Future work is to analyze the CCSM problem for strings. Another important issue is to analyze term-submatching, to generalize CCSM and LCCSM to terms, and then to improve the algorithms in [24].

Acknowledgements I thank David Sabel for discussions on compressions and for reading the paper.

References

1 D. Benanav, D. Kapur, and P. Narendran. Complexity of matching problems. *J. Symb. Comput.*, 3(1/2):203–216, 1987.

2 P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *J. Comput. Syst. Sci.*, 65(2):332–350, 2002.

3 J. Berstel and L. Boasson. Partial words and a theorem of Fine and Wilf. *Theor. Comput. Sci.*, 218(1):135–141, 1999.

4 Ph. Bille, R. Fagerberg, and I. Li Gørtz. Improved approximate string matching and regular expression matching on Ziv-Lempel compressed texts. *ACM Transactions on Algorithms*, 6(1), 2009.

5 F. Blanchet-Sadri. Periodicity on partial words. *Comput. Math. Appl.*, 47:71–82, 2004.

6 F. Blanchet-Sadri. *Algorithmic combinatorics on partial words*. Chapman & Hall/CRC, 2008.

7 F. Blanchet-Sadri, D. Dakota Blair, and R. V. Lewis. Equations on partial words. In *MFCS*, *LNCS* 4162 , pages 167–178. Springer, 2006.

8 F. Blanchet-Sadri and R. A. Hegstrom. Partial words and a theorem of Fine and Wilf revisited. *Theor. Comput. Sci.*, 270(1-2):401–419, 2002.

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

10 N. J. Fine and H. S. Wilf. Uniqueness theorem for periodic functions. *Proc. Am. Math. Soc.*, 16:109–114, 1965.

11 A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification and matching on compressed terms. *ACM Trans. Comput. Log.*, 12(4):26:1–26:37, 2011.

12 A. Jez. Faster fully compressed pattern matching by recompression. *CoRR*, abs/1111.3244, 2011.

13 M. Karpinski, W. Rytter, and A. Shinohara. Pattern-matching for strings with short description. In *CPM '95*, *LNCS* 937, pages 205–214. Springer-Verlag, 1995.

14 M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nord. J. Comput.*, 4(2):172–186, 1997.

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

16 J. Levy, M. Schmidt-Schauß, and M. Villaret. The complexity of monadic second-order unification. *SIAM J. of Computing*, 38(3):1113–1140, 2008.

17 Y. Lifshits. Processing compressed texts: A tractability border. In *CPM 2007*, pages 228–240, 2007.

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

19 M. Lohrey. Compressed membership problems for regular expressions and hierarchical automata. *Int. J. Found. Comput. Sci.*, 21(5):817–841, 2010.

20 N. Markey and Ph. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Inf. Process. Lett.*, 90(1):3–6, 2004.

21 G. Navarro. Regular expression searching on compressed text. *J. Discrete Algorithms*, 1(5-6):423–443, 2003.

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

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

24 M. Schmidt-Schauß. Pattern matching of compressed terms and contexts and polynomial rewriting. Frank report 43, Institut für Informatik. Goethe-Univ. Frankfurt am Main, 2011.

25 T. Yamamoto, H. Bannai, S. Inenaga, and M. Takeda. Faster subsequence and don't-care pattern matching on compressed texts. In *22nd CPM*, *LNCS* 6661, pages 309–322. Springer, 2011.

26 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

27 J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.