Querying and Embedding Compressed Texts

Yury Lifshits[1], Markus Lohrey[2]

[1] Steklov Institut of Mathematics, St.Petersburg, Russia
[2] Universität Stuttgart, FMI, Germany
yura@logic.pdmi.ras.ru, lohrey@informatik.uni-stuttgart.de

Abstract. In this work the computational complexity of two simple string problems on compressed input strings is considered: the querying problem (What is the symbol at a given position in a given input string?) and the embedding problem (Can the first input string embedded into the second input string?). Straight-line programs are used for text compression. It is shown that the querying problem becomes P-complete for compressed strings, while the embedding problem becomes hard for the complexity class Θ_2^p.

1 Introduction

During the last decade, the massive increase in the volume of data has motivated the need for algorithms on *compressed data*, like for instance compressed strings, trees, or pictures. The general goal is to develop efficient algorithms that directly work on compressed data without prior decompression, or to prove under general assumptions from complexity theory that such efficient algorithms do not exist. In this paper we concentrate on algorithms on compressed strings. We investigate two computational problems, which can be trivially solved in linear time for uncompressed input strings: the querying problem and the embedding problem. In the embedding problem we have given two input strings p (the pattern) and t (the text), and we ask whether p can be embedded into t, i.e., p can be obtained by deleting some letters of the text t at arbitrary positions, see Section 4 for a formal definition. In the querying problem the input consists of a string s, a position $i \in \mathbb{N}$, and a letter a, and we ask, whether the i-th symbol of s is a.

For string compression, we choose *straight-line programs* (SLPs), or equivalently context-free grammars that generate exactly one word. Straight-line programs turned out to be a very flexible and mathematically clean compressed representation of strings. Several other dictionary-based compressed representations, like for instance Lempel-Ziv (LZ) factorizations [24], can be converted in polynomial time into straight-line programs and vice versa [18]. This implies that complexity results, which refer to classes above deterministic polynomial time, can be transfered from SLP-encoded input strings to LZ-encoded input strings. It turns out that the computational complexity of the querying problem and the embedding problem becomes very different, when input strings are encoded via SLPs: While for SLP-compressed strings the querying problem (also called *compressed querying problem*) becomes complete for deterministic polynomial time (Thm. 4), the embedding problem (also called *fully compressed embedding problem*; the term fully is used because both, the pattern and the text are assumed to be compressed) becomes hard for the class Θ_2^p (Thm. 1). The latter class consists of all problems that can be accepted by a deterministic polynomial time machine with access to an oracle from NP and such that furthermore all questions to the oracle are asked in parallel [23]. Θ_2^p is located between the first and the second level of the polynomial time hierarchy, it contains NP and coNP and is contained in $\Sigma_2^p \cap \Pi_2^p$. We are currently not apply to prove a matching upper bound. The best upper bound for the fully compressed embedding problem that we can prove is PSPACE (Prop. 1). A corollary of the Θ_2^p-hardness of the fully compressed embedding problem is Θ_2^p-hardness of the *longest common subsequence problem* and the *shortest common supersequence problem* on SLP-compressed strings, even when both problems are restricted to two input strings. These problems have many applications e.g. in computational biology [10].

The paper is organized as follows. After introducing the necessary concepts in Sec. 2, we prove in Sec. 3, based on a reduction from the super increasing subset sum problem [11], P-completeness of the compressed querying problem for a binary input alphabet. For a variable input alphabet, we sharpen this result by showing that even for RLZ-encoded strings the compressed querying problem is P-complete, which solves an open problem from [7]. RLZ-encodings (restricted Lempel-Ziv encodings) can be seen as a restricted form of straight-line programs. In Sec. 4 we show that the fully compressed embedding problem is Θ_2^p-hard. The proof is divided into two main parts. First we prove NP-hardness by a reduction from the subset sum problem (Sec. 4.1). Second, we show how to simulate boolean operations via fully compressed embedding (Sec. 4.2). By taking together these two parts we can deduce hardness for Θ_2^p (Sec. 4.3).

1.1 Related work

Research on pattern matching problems for dictionary-based compressed strings started with the seminal paper [1]. In [17], Plandowski has shown that it can be tested in polynomial time whether two SLPs represent the same text. Plandowski's technique was extended in [8, 14] in order to show that the *fully compressed pattern matching problem* can be solved in polynomial time as well. The fully compressed pattern matching problem is the compressed version of the classical pattern matching problem: for two given SLPs P and T we ask, whether the text represented by T can be written as upv, where p is the text represented by the SLP P. Note the similarity between the *fully compressed pattern matching problem* and *the fully compressed embedding problem* studied in this paper: In the latter problem we also search for a compressed pattern in a compressed text, but we allow that the pattern occurs scattered, i.e., with gaps, in the text. This more liberal notion of pattern-occurrence makes the application of periodicity properties of words (in particular the lemma of Fine and Wilf), which are crucial in [8, 14, 17], impossible, and is in some sense the reason for the higher complexity of the fully compressed embedding problem. A similar complexity jump was observed when moving from ordinary (1-dimensional) to 2-dimensional text, i.e., rectangular pictures: In this framework, fully compressed pattern matching becomes Σ_2^P-complete [3].

The computational problems mentioned so far can be all formulated as particular compressed membership problems, where we ask whether a given compressed text belongs to some formal language, which may either be fixed or given in the input, e.g., in form of an automaton or a grammar. Precise complexity results for these problems were obtained in [2, 13] for regular languages and [12] for context-free languages.

Whereas it is NP-complete to compute (and even hard to approximate up to a constant factor) a minimal SLP that generates a given input string [4], several approaches for generating a small SLP that produces a given input string were proposed and analyzed in the literature, see e.g. [4, 21].

We refer to [7, 15, 18–20, 22] for a more detailed discussion of algorithmic problems on compressed strings.

2 Preliminaries

We assume that the reader has some basic background in complexity theory [16]. Let Σ be a finite alphabet. The *empty word* over Σ is denoted by ε. For a word $s = a_1 \cdots a_n \in \Sigma^*$ ($a_i \in \Sigma$) let $|s| = n$, $|s|_a = |\{i \mid a_i = a\}|$ (for $a \in \Sigma$), $s[i] = a_i$ (for $1 \leq i \leq n$), and $s[i, j] = a_i a_{i+1} \cdots a_j$ (for $1 \leq i \leq j \leq n$). If $i > j$ we set $s[i, j] = \varepsilon$.

Following [18], a *straight-line program (SLP) over the terminal alphabet Σ* is a context-free grammar G with ordered non-terminal symbols $X_1, \ldots, X_m$ (X_m is the starting symbol) such that there is exactly one production for each symbol: either $X_i \to a$, where $a \in \Sigma$ is a terminal, or $X_i \to X_j X_k$ for some $j, k < i$. The language generated by the SLP G contains exactly one word that is denoted by $\mathrm{eval}(G)$. More generally, every nonterminal X_i produces exactly one word that is denoted by $\mathrm{eval}_G(X_i)$. We omit the index G if the underlying SLP is clear from the context. The size of G is $|G| = m$.

We may allow in SLPs a more liberate form of productions, where the right-hand side for a nonterminal X_i is an arbitrary word over the alphabet $\Sigma \cup \{X_1, \ldots, X_{i-1}\}$. We may even allow exponential expressions of the form X_j^k for $j < i$ and a binary coded integer $k \in \mathbb{N}$. Such a production can be replaced by $O(\log(k))$ many ordinary productions.

3 Querying the i-th symbol

In this section, we study the following computational problem **Compressed Querying**:

INPUT: SLP G (over the terminal alphabet Σ), position $i \in \mathbb{N}$, and $a \in \Sigma$

QUESTION: $\mathrm{eval}(G)[i] = a$?

In this section, we prove that **Compressed Querying** is P-complete. This means that unless P $=$ NC, where NC is the class of all problems that can be solved in polylogarithmic time using polynomially many processors, there does not exist an efficient parallel algorithm for **Compressed Querying**, see [9] for background on P-completeness. All reductions in this section are NC-reductions, i.e., they can be computed in polylogarithmic time with only polynomially many processors.

Theorem 1. Compressed Querying *is* P*-complete. Hardness for* P *even holds for a binary terminal alphabet.*

Proof. Membership in P is easy to see: first compute for every non-terminal X of the input SLP the length ℓ_X of the generated string $\mathrm{eval}(X)$. Now, if we have a production $X \to YZ$ and we want to determine $\mathrm{eval}(X)[i]$ then we first check whether $i \leq \ell_Y$.

In this case we have to find $\mathrm{eval}(Y)[i]$. On the other hand, if $i > \ell_Y$, then we have to determine $\mathrm{eval}(Z)[i - \ell_X]$. This simple idea leads to a polynomial time algorithm.

We prove P-hardness by an NC-reduction from the P-complete problem **Super Increasing Subset Sum** [11]:

INPUT: Integers $w_1, \ldots, w_n, t$ in binary form such that $w_i > \sum_{j=1}^{i-1} w_j$ for all $1 \leq i \leq n$ (in particular $w_1 > 0$).

QUESTION: Do there exist $x_1, \ldots, x_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} x_i \cdot w_i = t$?

Thus, let $w_1, \ldots, w_n, t$ be integers such that $w_i > \sum_{j=1}^{i-1} w_j$. Let $G_1, \ldots, G_n \in \{0, 1\}^*$ be defined as follows, where $s_j = w_1 + \cdots + w_j$ for $1 \leq j \leq n$:

$$G_1 = 10^{w_1 - 1}1, \qquad G_j = G_{j-1}0^{w_j - s_{j-1} - 1}G_{j-1} \text{ for } 2 \leq j \leq n$$

It is straight-forward to construct from the instance $(w_1, \ldots, w_n, t)$ in NC an SLP that generates the string G_n. Note that $w_j > s_{j-1}$ and hence $w_j - s_{j-1} - 1 \geq 0$. Moreover, we claim that $|G_j| = s_j + 1$. This is certainly true for $j = 1$ since $s_1 = w_1$. For $j \geq 2$ we obtain inductively $|G_j| = 2|G_{j-1}| + w_j - s_{j-1} - 1 = 2s_{j-1} + 2 + w_j - s_{j-1} - 1 = s_j + 1$.

We claim that $G_n[t + 1] = 1$ if and only if there exist $x_1, \ldots, x_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} x_i \cdot w_i = t$, which proves the theorem. For this, we prove by induction on j that for every $p \geq 0$: $G_j[p + 1] = 1$ if and only if $\exists x_1, \ldots, x_j \in \{0, 1\} : \sum_{i=1}^{j} x_i \cdot w_i = p$. If $j = 1$, then $G_1[p + 1] = (10^{w_1 - 1}1)[p + 1] = 1$ if and only if $p = 0$ or $p = w_1$, which proves the induction base. Now assume that $j \geq 2$. Then $G_j[p + 1] = 1$ if and only if $(G_{j-1}0^{w_j - s_{j-1} - 1}G_{j-1})[p + 1] = 1$ if and only if $(G_{j-1}[p + 1] = 1$ or $G_{j-1}[p + 1 - |G_{j-1}| - w_j + s_{j-1} + 1] = 1)$ if and only if $(G_{j-1}[p + 1] = 1$ or $G_{j-1}[p + 1 - s_{j-1} - 1 - w_j + s_{j-1} + 1] = G_{j-1}[p + 1 - w_j] = 1)$ (since $|G_{j-1}| = s_{j-1} + 1$). By induction, this is true if and only if

$$\exists x_1, \ldots, x_{j-1} \in \{0, 1\} \left\{ \sum_{i=1}^{j-1} x_i \cdot w_i = p \quad \text{or} \quad \sum_{i=1}^{j-1} x_i \cdot w_i = p - w_j \right\}$$

But this is equivalent to $\exists x_1, \ldots, x_j \in \{0, 1\} : \sum_{i=1}^{j} x_i \cdot w_i = p$. □

Note that in Thm. 1, P-hardness already holds for a binary alphabet. If we allow the terminal alphabet to be part of the input, then we can prove P-hardness even for a restricted form of SLPs, so called *restricted Lempel-Ziv encodings*, briefly RLZ-encoding. For a given string $w \in \Sigma^+$, the *RLZ-factorization* of w is the unique factorization $w = f_1 f_2 \cdots f_k$ such that for every $i \geq 1$, f_i is either the longest non-empty prefix of $f_i f_{i+1} \cdots f_k$ such that there exists $1 \leq j, k < i$ with $f_i = f_j \cdots f_k$, or f_i is the first symbol of $f_i f_{i+1} \cdots f_k$. In this situation, the *RLZ-encoding* of w, briefly $\mathrm{RLZ}(w)$ is the sequence $c_1 c_2 \cdots c_k$, where $c_i = f_i$ if $f_i \in \Sigma$ or $c_i = [j, k]$ if $f_j f_{j+1} \cdots f_k$ is the longest non-empty prefix of $f_i f_{i+1} \cdots f_k$. Note that from $\mathrm{RLZ}(w)$ one can easily construct an SLP generating w.

Example 1. Let $w = abaababaabaababaababa$. Then the RLZ-factorization of w is $a \,|\, b \,|\, a \,|\, aba \,|\, baaba \,|\, ababaaba \,|\, ba$ and $\mathrm{RLZ}(w) = aba[1, 3][2, 4][4, 5][2, 3]$.

The following theorem solves an open problem from [7], where a corresponding result for LZ-encoded input strings (see [7] for the definition) was shown:

Theorem 2. *The following problem is* P*-complete:*

INPUT: An alphabet Σ, a string $w \in \Sigma^*$ given by its RLZ-encoding, a position $i \in \mathbb{N}$, and $a \in \Sigma$

QUESTION: $w[i] = a$?

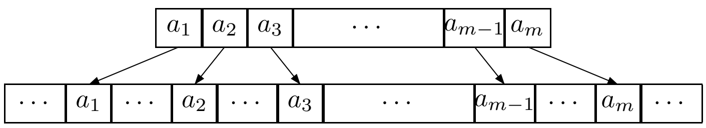
Proof. Membership in P follows from Thm. 1. For P-hardness we use almost the same construction as in the proof of Thm. 1. For a given instance $(w_1, \ldots, w_n, t)$ of **Super Increasing Subset Sum** we define strings $G_1, \ldots, G_n \in \{1, \$_1, \ldots, \$_n\}^*$ as follows, where $s_j = w_1 + \cdots + w_j$ for $1 \leq j \leq n$:

$$G_1 = 1\$_1^{w_1-1}1, \qquad G_j = G_{j-1}\$_j^{w_j - s_{j-1}-1}G_{j-1} \text{ for } 2 \leq j \leq n$$

The proof of Thm. 1 shows that $G_n[t+1] = 1$ if and only if there exist $x_1, \ldots, x_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} x_i \cdot w_i = t$. It remains to prove that RLZ(G_n) can be constructed in NC from $(w_1, \ldots, w_n, t)$. In the following let $\ell(i)$ for $i \in \mathbb{N}$ be the number of factors in the RLZ-factorization of a^i. One can show that $\ell(i) \in O(\log(i))$ and RLZ(a^i) can be calculated in NC from the binary encoding of i. Now we determine the number λ_i of factors of the RLZ-factorization of the string G_i. We have $\lambda_1 = 2 + \ell(w_1 - 1)$ and $\lambda_i = \lambda_{i-1} + \ell(w_i - s_{i-1} - 1) + 1$ for $i > 1$. Thus, $\lambda_i = (i+1) + \sum_{k=1}^{i} \ell(w_i - s_{i-1} - 1)$. Also the numbers λ_i $(1 \leq i \leq n)$ can be calculated in NC using the prefix sum algorithm. Now we can set in parallel for all $1 \leq i \leq n$ the factor from position $\lambda_{i-1} + 1$ to λ_i of RLZ(G_n) (where $\lambda_0 = 0$) to RLZ$(\$_i^{w_i - s_{i-1}-1})^{+\lambda_{i-1}}[1, \lambda_{i-1}]$, where RLZ$(w)^{+j}$ is the same as RLZ(w) but where j is added to all numbers. □

4 Complexity of Embedding

We say that a string $p = a_1 \cdots a_m$ can be *embedded* into a string $t = b_1 \cdots b_n$ $(a_i, b_j \in \Sigma)$, briefly $p \hookrightarrow t$, if there exist positions $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ such that $b_{i_k} = a_k$ for $1 \leq k \leq m$. One also says that p is a *subsequence* of t, see the following diagram:



In this section, we study the complexity of the following problem **Fully Compressed Embedding**, for short **Embedding**:

INPUT: SLPs P and T

QUESTION: $\text{eval}(P) \hookrightarrow \text{eval}(T)$?

The following upper bound for **Embedding** is easy to prove:

Proposition 1. Embedding *belongs to* PSPACE.

Proof. The straight-forward greedy algorithm that solves the embedding problem for uncompressed strings in linear time results in a PSPACE-algorithm for SLP-compressed strings. The crucial observation is that a position in a string, which is represented by an SLP, can be stored in polynomial space with respect to the size of the SLP. □

A simple greedy algorithm for checking $\text{eval}(P) \hookrightarrow \text{eval}(T)$ can be easily implemented within the time bound $|\text{eval}(P)| \cdot |T|^{O(1)} \leq 2^{O(|P|)} \cdot |T|^{O(1)}$. This shows in particular that **Embedding** is fixed parameter tractable in the sense of [5], when the size of the pattern-SLP is chosen as the parameter (which is reasonable, because in most patter matching applications the pattern is much smaller than the text).

Our main result states that **Embedding** is hard for the complexity class Θ_2^p. In Sec. 4.1, we will first only prove NP-hardness. Then, in Sec. 4.2 we show how to simulate boolean operations within **Embedding**. From this, we will deduce hardness for Θ_2^p in Sec. 4.3.

4.1 NP-hardness of Embedding

Let us recall the well-known NP-complete problem **Subset Sum** (see [6]):
INPUT: Integers $w_1, \dots, w_n, t$ in binary form
QUESTION: Do there exist $x_1, \dots, x_n \in \{0, 1\}$ with $\sum_{i=1}^{n} x_i \cdot w_i = t$?

Theorem 3. Embedding *is* NP-*hard.*

Proof. We prove the theorem by a polynomial time reduction from **Subset Sum** to **Embedding**. Let $t, \bar{w} = (w_1, \dots, w_n)$ be input data for **Subset Sum**; w.l.o.g. assume that $n > 1$. We are going to construct SLPs G and H such that there exists a subset of $\{w_1, \dots, w_n\}$ with sum equal to t if and only if $\text{eval}(G) \hookrightarrow \text{eval}(H)$.

We begin with some notation. Let $s = w_1 + \cdots + w_n$ and $N = 2^n s$. We can assume that $t < s$. Let $x \in \{0, \dots, 2^n - 1\}$ be an integer. With x_i $(1 \leq i \leq n)$ we denote the i-th bit in the binary representation of x, where x_1 is the least significant bit. Thus, $x = \sum_{i=1}^{n} x_i 2^{i-1}$. We define $x \circ \overline{w} = \sum_{i=1}^{n} x_i w_i$, thus, $x \circ \overline{w}$ is the sum of the subset of $\{w_1, \dots, w_n\}$ encoded by the integer x. Hence, $t, \overline{w}$ is a positive instance of **Subset Sum** if and only if $\exists x \in \{0, \dots, 2^n - 1\} : x \circ \overline{w} = t$. We now define strings g and h as follows:

$$h_1 = \prod_{x=0}^{2^n-1} (10^s) = (10^s)^{2^n} \qquad h_2 = 0^{2N} \qquad h_3 = \prod_{x=0}^{2^n-1} (0^{x \circ \overline{w}} 10^{s - x \circ \overline{w}})$$

$$h_4 = 0^{t+1} \qquad h_0 = h_1 h_2 h_3 h_4 \qquad h = h_0^{5N}$$

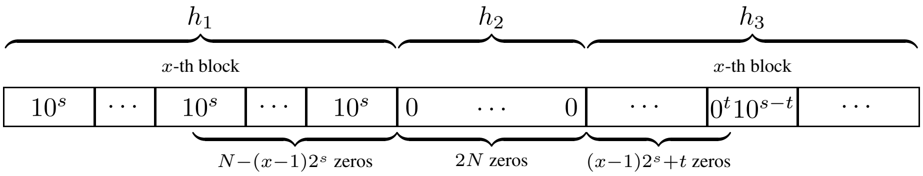
$$g_0 = 10^{3N+t} 10^{N+1} \qquad g = g_0^{5N-1}$$

We use the symbol $\prod$ to denote the concatenation of the corresponding words performed in order the $x = 0, \dots, 2^n - 1$.

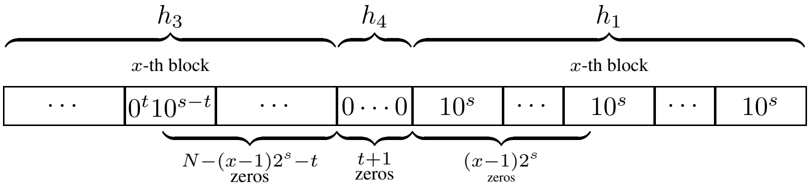
We first claim that the strings g and h can be generated by SLPs of polynomial size with respect to the size of the input $t, \overline{w}$. Note that with only one exception, namely the definition h_3, only a constant number of concatenations and integer exponents with polynomially many bits are used in the definition of g and h. These constructions can be directly realized by SLPs. Finally, a construction of a polynomial size SLP for h_3 was presented in [12].

Now we prove that $g \hookrightarrow h$ if and only if $\exists x \in \{0, \dots, 2^n - 1\} : x \circ \overline{w} = t$. First assume that there is $x \in \{0, \dots, 2^n - 1\}$ such that $x \circ \overline{w} = t$. Consider the prefix

$h_1h_2h_3h_4h_1$ of h. We can embed $g_0 = 10^{3N+t}10^{N+1}$ into $h_1h_2h_3h_4h_1$: map the initial 1 of g_0 to the x-th block 10^s of h_1. Since $x \circ \overline{w} = t$, the number of 0's in $h_1h_2h_3$ between the 1 in the x-th block 10^s of h_1 and the x-th block $0^{x\circ\overline{w}}10^{s-x\circ\overline{w}} = 0^t10^{s-t}$ of h_3 is precisely $N-(x-1)2^s+2N+(x-1)2^s+x\cdot\overline{w} = 3N+t$, see the following diagram:

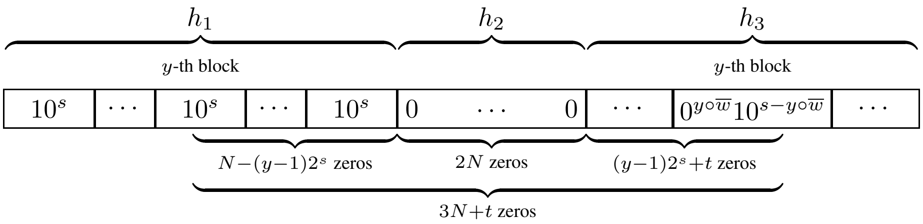


To these $3N+t$ many 0's we map the first $3N+t$ many 0's of g_0. Then the second 1 of g_0 is mapped to the 1 in the x-th block 0^t10^{s-t} of h_3. The next $N+1$ many 0's following this 1 are used for embedding the remaining $N+1$ 0's of g_0. The crucial point is that after this embedding, we again arrive at the 1 in the x-th block 10^s of h_1, see the following diagram:

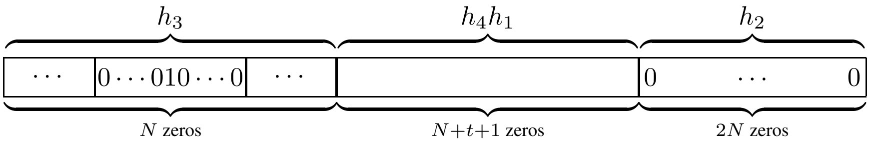


This observation shows that g_0^k can be embedded into $h_0^{k+1} = (h_1h_2h_3h_4)^{k+1}$ for every $k \geq 1$. In particular $g = g_0^{5N-1} \hookrightarrow h_0^{5N} = h$.

Next, we prove the reverse direction. Assume that $g \hookrightarrow h$. We have to show that there is $x \in \{0,\ldots,2^n-1\}$ such that $x \circ \overline{w} = t$. In order to deduce a contradiction, assume that $x \circ \overline{w} \neq t$ for all $x \in \{0,\ldots,2^n-1\}$. Not every 0 in h can be the image of a 0 from g under our embedding $g \hookrightarrow h$. Let us estimate the total number of such unused 0's. Our embedding $g \hookrightarrow h$ consists of $5N-1$ disjoint embeddings of g_0 into h. There are two 1's in g_0 and there are exactly $3N+t$ many 0's between them. We claim that there is no pair of two 1's with exactly $3N+t$ many 0's between them in h. In order to prove this, we consider two 1's in h and make a case distinction on the position of the first 1. First assume that the left 1 belongs to h_1. More precisely, assume that the left 1 is the 1 in the y-th block of 10^s of h_1. By reading precisely $3N+t$ many 0's in h, we arrive at position $t+1$ in the y-th block of h_3 (note that $t < s$). But since $y \circ \overline{w} \neq t$, the $(t+1)$-th symbol in the y-th block of h_3 is not 1. This prove the case that the left 1 belongs to h_1. The following diagram visualizes the situation (where we assume that $t > y \circ \overline{w}$):



In the second case, the left 1 in our pair is situated in h_3. Then, by reading $3N+t$ many 0's in h, we end up in h_2, which does not contain 1's at all:



We have now shown that for each embedding of g_0 in h between the images of the two 1's in g_0, there must be at least $3N+t+1$ many 0's in h. Thus, for every embedding of $g_0 = 10^{3N+t}10^{N+1}$ in h we need at least $3N+t+1+N+1 = 4N+t+2$ many 0's in h. Since $g = g_0^{5N-1}$, we need at least

$$(4N+t+2)\cdot(5N-1) = 5N\cdot(4N+t+1)+(N-t-2) > 5N\cdot(4N+t+1)$$

many 0's in h. For the last inequality note that $N = s\cdot 2^n \geq 4s > s+2 > t+2$. We obtain a contradiction, because from the construction of h, we see that h contains precisely $5N\cdot(4N+t+1)$ many 0's. □

4.2 Simulating boolean operations

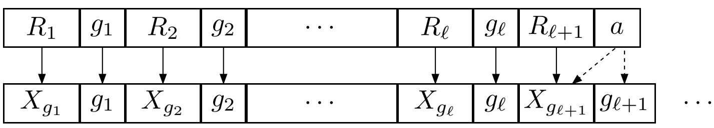
Proposition 2. *For SLPs G and H over a terminal alphabet Σ, $|\Sigma| \geq 1$, we can construct in polynomial time SLPs G' and H' over the terminal alphabet Σ such that*

$$\mathrm{eval}(G) \hookrightarrow \mathrm{eval}(H) \quad \Leftrightarrow \quad \mathrm{eval}(G') \not\hookrightarrow \mathrm{eval}(H'). \tag{1}$$

Proof. Let $\mathrm{eval}(G) = g_1\cdots g_k$ and $\mathrm{eval}(H) = h_1\cdots h_m$. For $a\in\Sigma$ let $X_a = (a_1\cdots a_n)^{m+1}$, where $\{a_1,\ldots,a_n\} = \Sigma\setminus\{a\}$ (the order on $\Sigma\setminus\{a\}$ is arbitrary here, if $n=0$, then $X_a=\varepsilon$). Let $a\in\Sigma$ be arbitrary and let G' and H' be SLPs with

$$\mathrm{eval}(G') = \mathrm{eval}(H)a = h_1\cdots h_m a \quad\text{and}\quad \mathrm{eval}(H') = X_{g_1}g_1\cdots X_{g_k}g_k.$$

These SLPs can be constructed in polynomial time from G and H. For G' this is clear. For H' we have to replace every terminal symbol a in G by a new nonterminal A and add the rule $A\to X_a a$. It remains to show (1). First assume that $\mathrm{eval}(G)\not\hookrightarrow \mathrm{eval}(H)$. Then we can write $\mathrm{eval}(H) = R_1g_1\cdots R_lg_lR_{l+1}$, where $l<k$ and for $1\leq i\leq l+1$, the word R_i does not contain the letter g_i. Since $|R_i|\leq m$, for every $1\leq i\leq l+1$ it is true that $R_i\hookrightarrow X_{g_i}$. Thus, we can embed the prefix $\mathrm{eval}(H) = R_1g_1\cdots R_lg_lR_{l+1}$ of $\mathrm{eval}(G')$ into the prefix $X_{g_1}g_1\cdots X_{g_l}g_lX_{g_{l+1}}$ of $\mathrm{eval}(H')$. The final letter a of $\mathrm{eval}(G')$ can be either also mapped to $X_{g_{l+1}}$ (if $a\neq g_{l+1}$; here it is important that $|X_{g_{l+1}}|>m$ so that R_{l+1} does not completely occupy $X_{g_{l+1}}$) or it can be mapped to g_{l+1} (if $a=g_{l+1}$).



Now assume that $\text{eval}(G) \hookrightarrow \text{eval}(H)$. Then we can write $\text{eval}(H) = R_1 g_1 \cdots R_k g_k R$, where for $1 \leq i \leq k$, the word R_i does not contain the letter g_i. We claim that

$$\forall 1 \leq i \leq k : R_1 g_1 \cdots R_i g_i \not\hookrightarrow X_{g_1} g_1 \cdots X_{g_{i-1}} g_{i-1} X_{g_i}. \tag{2}$$

Our proof goes by induction on i. In the case $i = 1$ this follows, since g_1 does not occur in X_{g_1}. For the induction step assume that (2) is true for some $i \geq 1$ and that moreover

$$R_1 g_1 \cdots R_{i+1} g_{i+1} \hookrightarrow X_{g_1} g_1 \cdots X_{g_i} g_i X_{g_{i+1}}. \tag{3}$$

Recall that the last symbol g_{i+1} of $R_1 g_1 \ldots R_{i+1} g_{i+1}$ does not occur in the suffix $X_{g_{i+1}}$ of $X_{g_1} g_1 \ldots X_{g_i} g_i X_{g_{i+1}}$. Thus, (3) implies that already $R_1 g_1 \cdots R_i g_i R_{i+1} g_{i+1} \hookrightarrow X_{g_1} g_1 \cdots X_{g_{i-1}} g_{i-1} X_{g_i} g_i$ and hence $R_1 g_1 \cdots R_i g_i R_{i+1} \hookrightarrow X_{g_1} g_1 \cdots X_{g_{i-1}} g_{i-1} X_{g_i}$. But this contradicts (2).

For $i = k$, (2) implies $R_1 g_1 \cdots R_k g_k \not\hookrightarrow X_{g_1} g_1 \cdots X_{g_{k-1}} g_{k-1} X_{g_k}$. But then $\text{eval}(G') = R_1 g_1 \cdots R_k g_k R a \not\hookrightarrow X_{g_1} g_1 \cdots X_{g_{k-1}} g_{k-1} X_{g_k} g_k = \text{eval}(H')$. □

Thm. 3 and Prop. 2 immediately imply that **Embedding** is also coNP-hard.

Proposition 3. *For SLPs G_1, H_1, G_2, H_2 over a terminal alphabet Σ, $|\Sigma| \geq 2$, we can construct in polynomial time SLPs G, H over the terminal alphabet Σ such that*

$$(\text{eval}(G_1) \hookrightarrow \text{eval}(H_1) \textit{ and } \text{eval}(G_2) \hookrightarrow \text{eval}(H_2)) \quad \Leftrightarrow \quad \text{eval}(G) \hookrightarrow \text{eval}(H).$$

Proof. W.l.o.g. assume that G_1 and G_2 (resp. H_1 and H_2) have disjoint sets of non-terminals. Let S_i (resp. T_i) be the start non-terminal of G_i (resp. H_i). Let $N = 1 + \max\{|\text{eval}(H_1)|, |\text{eval}(H_2)|\}$. Then G (resp. H) contains all productions of G_1 and G_2 (resp. H_1 and H_2) and the additional production $S \to S_1 1^N 0 1^N S_2$ (resp. $T \to T_1 1^N 0 1^N T_2$), where $0, 1 \in \Sigma$. Here, S (resp. T) is the start non-terminal of G (resp. H). Thus,

$$\text{eval}(G) = \text{eval}(G_1)\, 1^N\, 0\, 1^N\, \text{eval}(G_2)$$
$$\text{eval}(H) = \text{eval}(H_1)\, 1^N\, 0\, 1^N\, \text{eval}(H_2).$$

Clearly, if $\text{eval}(G_1) \hookrightarrow \text{eval}(H_1)$ and $\text{eval}(G_2) \hookrightarrow \text{eval}(H_2)$, then $\text{eval}(G) \hookrightarrow \text{eval}(H)$. For the other direction note that if $\text{eval}(G_1) 1^N 0 1^N \text{eval}(G_2)$ can be embedded into $\text{eval}(H_1) 1^N 0 1^N \text{eval}(H_2)$, then by the choice of N, the 0 at position $|\text{eval}(G_1)| + N + 1$ in $\text{eval}(G_1) 1^N 0 1^N \text{eval}(G_2)$ can neither be mapped to the prefix $\text{eval}(H_1)$ nor to the suffix $\text{eval}(H_2)$ of $\text{eval}(H)$. Thus, this 0 has to be mapped to the 0 at position $|\text{eval}(H_1)| + N + 1$ in $\text{eval}(H_1) 1^N 0 1^N \text{eval}(H_2)$. This implies that both $\text{eval}(G_1) \hookrightarrow \text{eval}(H_1)$ and $\text{eval}(G_2) \hookrightarrow \text{eval}(H_2)$. □

Of course, using Prop. 2 and 3 we can also simulate an OR-operation. But the problem with our construction for NOT is that it cannot be iterated since one application of the construction for Prop. 2 leads to a quadratic blow-up in the size of the SLPs. Therefore, in order to encode circuits, we must also present a construction for OR:

Proposition 4. *For SLPs G_1, H_1, G_2, H_2 over a terminal alphabet Σ, $|\Sigma| \geq 2$, we can construct in polynomial time SLPs G, H over the terminal alphabet Σ such that*

$$(\text{eval}(G_1) \hookrightarrow \text{eval}(H_1) \textit{ or } \text{eval}(G_2) \hookrightarrow \text{eval}(H_2)) \quad \Leftrightarrow \quad \text{eval}(G) \hookrightarrow \text{eval}(H).$$

Proof. W.l.o.g. assume that G_1, G_2, H_1, and H_2 have pairwise disjoint sets of non-terminals. Let S_i (resp. T_i) be the start non-terminal of G_i (resp. H_i). Let $N = 1 + |\mathrm{eval}(G_1)| + |\mathrm{eval}(G_2)|$. Then G contains all productions of G_1 and G_2 and the additional production $S \to S_1 0 1^N 0 S_2$. The SLP H contains all productions of G_1, H_1, G_2, H_2 and the additional production $T \to T_1 0 1^N S_1 0 S_2 1^N 0 T_2$. Thus, we have

$$\begin{aligned}\mathrm{eval}(G) &= \mathrm{eval}(G_1)\, 0\, 1^N\, 0\, \mathrm{eval}(G_2)\\ \mathrm{eval}(H) &= \mathrm{eval}(H_1)\, 0\, 1^N\, \mathrm{eval}(G_1)\, 0\, \mathrm{eval}(G_2)\, 1^N\, 0\, \mathrm{eval}(H_2).\end{aligned}$$

Clearly, if $\mathrm{eval}(G_1) \hookrightarrow \mathrm{eval}(H_1)$ or $\mathrm{eval}(G_2) \hookrightarrow \mathrm{eval}(H_2)$, then $\mathrm{eval}(G) \hookrightarrow \mathrm{eval}(H)$. For the other direction assume that $\mathrm{eval}(G) = \mathrm{eval}(G_1)\, 0\, 1^N\, 0\, \mathrm{eval}(G_2)$ can be embedded into $\mathrm{eval}(H) = \mathrm{eval}(H_1)\, 0\, 1^N\, \mathrm{eval}(G_1)\, 0\, \mathrm{eval}(G_2)\, 1^N\, 0\, \mathrm{eval}(H_2)$. Consider the 1^N-block of $\mathrm{eval}(G)$. If a 1 from this block is mapped to the prefix $\mathrm{eval}(H_1)$ of $\mathrm{eval}(H)$, then $\mathrm{eval}(G_1) \hookrightarrow \mathrm{eval}(H_1)$. If a 1 from the 1^N-block of $\mathrm{eval}(G)$ is mapped to the first 1^N-block of $\mathrm{eval}(H)$, then the 0 at position $|\mathrm{eval}(G_1)| + 1$ in $\mathrm{eval}(G)$ cannot be mapped right of the 0 at position $|\mathrm{eval}(H_1)| + 1$ in $\mathrm{eval}(H)$. But then again the prefix $\mathrm{eval}(G_1)$ of $\mathrm{eval}(G)$ is embedded into the prefix $\mathrm{eval}(H_1)$ of $\mathrm{eval}(H)$. Completely analogously it follows that if a 1 from the 1^N-block of $\mathrm{eval}(G)$ is mapped to the suffix $\mathrm{eval}(H_2)$ of $\mathrm{eval}(H)$ or to the second 1^N-block of $\mathrm{eval}(H)$, then $\mathrm{eval}(G_2) \hookrightarrow \mathrm{eval}(H_2)$. The only remaining case, namely that every 1 in the 1^N-block of $\mathrm{eval}(G)$ is mapped into $\mathrm{eval}(G_1)\, 0\, \mathrm{eval}(G_2)$ cannot occur, since $N > |\mathrm{eval}(G_1)\mathrm{eval}(G_2)|$. □

4.3 Hardness for Θ_2^p

Recall that Θ_2^p is the class of all problems that can be accepted by a deterministic polynomial time machine with access to an oracle from NP and such that furthermore all questions to the oracle are asked in parallel [23].

Proposition 5. *If $A \subseteq \{0,1\}^*$ is NP-complete, then the following problem is Θ_2^p-complete:*

INPUT: A boolean circuit C with input gates labeled by words over $\{0,1\}$?

QUESTION: Does C evaluate to true when every input gate g that is labeled with $w \in \{0,1\}^*$ evaluates to true (resp. false) if $w \in A$ (resp. $w \notin A$).

Proof. For the membership in Θ_2^p note that we can evaluate all input gates of C in parallel by using the language A as an oracle. Then, the whole circuit can be evaluated in polynomial time. Hardness for Θ_2^p follows from a result from [23]: It is Θ_2^p-complete to decide for a given list of strings $w_1, w_2, \ldots, w_n \in \{0,1\}^*$, whether the number $|\{i \mid w_i \in A\}|$ is odd. By taking a boolean circuit for parity, this problem can be easily encoded into a boolean circuit with A-instances at input gates. □

Theorem 4. *Even for SLPs with a binary terminal alphabet,* **Embedding** *is Θ_2^p-hard.*

Proof. Let C be a circuit with input gates labeled with instances of the NP-complete **Subset Sum** problem. By the usual doubling argument, we can assume that negation gates only occur directly above input gates. We first define inductively for every gate c strings $u(c)$ and $v(c)$ and then argue that (i) c evaluates to true if and only if $u(c) \hookrightarrow$

$v(c)$ and (ii) $u(c)$ and $v(c)$ can be generated by "small" SLPs. If c is an unnegated input gate that is labeled with the **Subset Sum** instance I then $u(c) = g$ and $v(c) = h$, where g and h are the two strings that are constructed from I in the proof of Thm. 3. If c is a negated input gate that is labeled with with the **Subset Sum** instance I, then again we first construct from I the words g and h as described in the proof of Thm. 3. Then we apply the construction from the proof of Prop. 2 to g and h and assign the resulting strings to $u(c)$ and $v(c)$, respectively. For AND- and OR-gates we use the constructions from Prop. 3 and 4: If c is an AND-gate with inputs c_1 and c_2, then

$$u(c) = u(c_1)\, 1^N\, 0\, 1^N\, u(c_2) \;\text{ and }\; v(c) = v(c_1)\, 1^N\, 0\, 1^N\, v(c_2), \tag{4}$$

where $N = 1 + \max\{|v(c_1)|, |v(c_2)|\}$. If c is an OR-gate with inputs c_1 and c_2, then

$$u(c) = u(c_1)\, 0\, 1^N\, 0\, u(c_2) \;\text{ and }\; v(c) = v(c_1)\, 0\, 1^N\, u(c_1)\, 0\, u(c_2)\, 1^N\, 0\, v(c_2), \tag{5}$$

where $N = 1 + |u(c_1)| + |u(c_2)|$. From Thm. 3 and Prop. 2–4 it follows immediately that C evaluates to true if and only if $u(o) \hookrightarrow v(o)$, where o is the output gate of C.

It remains to argue that for every gate c, the strings $u(c)$ and $v(c)$ can be generated by SLPs of size polynomially bounded in the size of the circuit C (which is the number of gates plus the size of all **Subset Sum** instances at the leafs). Note that if we define $n(c) = \max\{|u(c)|, |v(c)|\}$ then we have $n(c) \le 8 \cdot \max\{n(c_1), n(c_2)\} + 5$ in case c is an AND- or OR-gate with inputs c_1 and c_2. It follows that $n(c)$ is bounded exponentially in the size of the circuit C. Moreover, we can calculate the binary representations of the lengths $|u(c)|$ and $|v(c)|$ for every gate c in polynomial time. Thus, we can construct SLPs of polynomial size for the factors 1^N in (4) and (5). This implies that for every gate c, $u(c)$ and $v(c)$ can be generated by SLPs of polynomial size. ☐

Let us close this paper with a corollary of Thm. 4. In the problem **Longest Common Subsequence** (**LCS**) (resp. **Shortest Common Supersequence** (**SCS**)), one asks for a finite set R of strings and $n \in \mathbb{N}$ whether there is a string w with $|w| \ge n$ and $\forall v \in R : w \hookrightarrow v$ (resp. $|w| \le n$ and $\forall v \in R : v \hookrightarrow w$). These problems are known to be NP-complete, but for $|R| = 2$ they can be solved in polynomial time (see [6]). For SLP-encoded input strings, **LCS** and **SCS** can be both solved in PSPACE.

Corollary 1. *The problems* **LCS** *and* **SCS** *for SLP-encoded input strings are Θ_2^p-hard, even if $|R| = 2$ for the input set R.*

Proof. For $u, v \in \Sigma^*$ we have $u \hookrightarrow v$ if and only if $(\{u, v\}, |u|)$ (resp. $(\{u, v\}, |v|)$ is a true instance of **LCS** (resp. **SCS**). Hence, the corollary follows from Thm. 4. ☐

5 Open problems

The main open problems that remains from this paper concerns the precise complexity of **Embedding**. Our results leave a gap from Θ_2^p to PSPACE. In Thm. 2 (P-completeness of querying RLZ-encoded input strings) it is open, whether the underlying alphabet can be fixed to, e.g., a binary alphabet.

Acknowledgments This work was done during a visit of the first author at University of Stuttgart, Germany, which was supported by the DFG project GELO. The first author was also supported by a grant from the project INTAS 04-77-7173.

References

1. A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. *J. Comput. Syst. Sci*, 52(2):299–307, 1996.
2. M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: From word to circuit evaluation. *SIAM J. Comput.*, 26(1):138–152, 1997.
3. 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.
4. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.
5. R. G. Downey and M. R. Fellows. *Parametrized Complexity*. Springer, 1999.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness*. Freeman, 1979.
7. L. Gasieniec, A. Gibbons, and W. Rytter. Efficiency of fast parallel pattern searching in highly compressed texts. In *Proc. MFCS'99*, LNCS 1672, pages 48–58. Springer, 1999.
8. L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding (extended abstract). In *Proc. SWAT 1996*, LNCS 1097, pages 392–403. Springer, 1996.
9. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford Univ. Press, 1995.
10. D. Gushfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge Univ. Press, 1999.
11. H. J. Karloff and W. L. Ruzzo. The iterated mod problem. *Inf. Comput.*, 80(3):193–204, 1989.
12. M. Lohrey. Word problems on compressed word. In *Proc. ICALP 2004*, LNCS 3142, pages 906–918. Springer, 2004. long version appears in SIAM J. Comput.
13. N. Markey and P. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Inf. Process. Lett.*, 90(1):3–6, 2004.
14. M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proc. CPM 97*, LNCS 1264, pages 1–11. Springer, 1997.
15. G. Navarro. Regular expression searching on compressed text. *J. Discrete Algorithms*, 1(5–6):423–443, 2003.
16. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
17. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, LNCS 855, pages 460–470. Springer, 1994.
18. W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 262–272. Springer, 1999.
19. W. Rytter. Algorithms on compressed strings and arrays. In *Proc. SOFSEM'99*, LNCS 1725, pages 48–65. Springer, 1999.
20. W. Rytter. Compressed and fully compressed pattern matching in one and two dimensions. *Proceedings of the IEEE*, 88(11):1769–1778, 2000.
21. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1–3):211–222, 2003.
22. W. Rytter. Grammar compression, LZ-encodings, and string algorithms with implicit input. In *Proc. ICALP 2004*, LNCS 3142, pages 15–27. Springer, 2004.
23. K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theor. Comput. Sci.*, 51:53–80, 1987.
24. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.