1 Research Project Objectives

This project intends to study selected aspects of *grammar compression*, related to algorithmic design, implementation as well as theoretical properties. In grammar compression we represent an input (treated as a string) as a context free grammar generating exactly the given string; for historical reasons this grammar is called an SLP (Straight-Line Programme) for this string; variant of SLPs for trees are also known.

This sort of compression is on one hand applied in practice, where it is for example a basis for popular compression standards like LZ78 and LZW, which in turn are used in, for instance, GIF or PDF file formats. It is also employed in theoretical considerations, as it turns out that for many problems we can represent their solutions as SLPs and perform operations on them directly, thus greatly reducing the processing time. Such an approach is applicable in areas so far away as computations in algebra and programme verification.

Yet much is still to learn about grammar compression in terms of their approximation, running times of algorithms for them, possible extensions to trees and their application in other areas. This project proposes to investigate some unanswered questions about the grammar compression, which the applicant finds particularly important and interesting.

Grammar compressors and entropy In the first group of tasks we will investigate the connection between grammar compression and entropy.

We hope that we can give better (or completely new) upper bounds on the effectiveness of known heuristical grammar compressors in terms of empirical entropy of the string. This would theoretically back the practical usefulness of such compressors.

We plan to improve known grammar compressors so that they will perform better in terms of bit size of the output. The hope is that the investigation of the connection between grammar compression and entropy can lead to improvements of those heuristics. We also plan to develop algorithms whose bit-encoding is provably good when compared to the entropy of the source, i.e. an approximation algorithm which aims at bit-size.

Objectives: *Construct a grammar compressor heuristic that utilises the information on empirical entropy. Give (or improve) upper bounds on performance of popular heuristics for grammar compression in terms of empirical entropy. Construct a grammar compressor that achieves or approximates k-th empirical entropy (for small enough k).*

Approximation ratio of grammar compressors In the second group we will investigate approximation ratios for of grammar compressors.

efficiently

It is known that the smallest SLP for a string cannot be computed, instead approximation algorithms and heuristics are considered. One of the ways to estimate their performance is to consider their approximation ratio, which is in turn naturally complemented by proving lower bounds for this problem.

We plan to improve the lower bounds of approximation ratios for the smallest grammar problem. On the other hand, we want to improve lower and upper bounds on the approximation ratios of popular heuristics for this problem.

Objectives: *Improve the lower bounds for the smallest SLP problem. Improve lower and upper bounds on approximation ratios of heuristical grammar compressors.*

Running time lower bounds In the third group we will investigate (perhaps conditional) lower bounds for running time of algorithms working on grammar compressed strings.

The running times of many algorithms for grammar compressed strings were recently improved, however nearly no running-time lower bounds were known for those problems. This was changed recently, when conditional lower bounds, meaning ones that assume some computational complexity lower bound—in this case—the Strong Exponential Time Hypothesis, were shown for a couple of problems related to SLPs.

We plan to give lower bounds, probably conditional ones, for other problems related to SLPs. We want to mainly focus on the most applied result in this area—equivalence of SLPs (and it more general variant: compressed pattern matching), but also on recognition of a compressed word by a compressed automaton and also on other problems.

Objectives: *Give (conditional) lower bounds for the equivalence testing and fully compressed pattern matching of SLPs. Give a (conditional) lower bound for the fully compressed membership problem of DFAs. Give (conditional) for other problems related do SLPs.*

Forests grammars In the fourth group of tasks we will investigate new model of grammar compression specialised for trees: the Forest SLPs (FSLPs).

This model was introduced recently and it generalises SLPs to trees in a different way, in particular, its basic building blocks are forests and not trees and it uses two types of concatenation operation: horizontal and

vertical one. The main motivation is the inability of the older, well-known TSLPs to handle nodes of unbounded degree, which was the largest drawback of TSLPs.

This model is not well researched and the applicant wants to investigate some of its basic properties. Firstly, what is its exact connection to other tree compression models, in particular top trees and TSLPs (here, with first child next sibling encoding)? The initial work on this model investigated the equivalence under the associativity and commutativity. What other operations can be allowed so that testing equivalence is still possible? Lastly, since the model is akin to both SLPs and TSLPs, can the recompression approach, which was so successful for them, be extended to FSLPs? What are the applications and consequences of such an extension–most likely this would yield alternative equivalence testing algorithm, approximation algorithm and possibly new insight to the context unification problem?

Objectives: *Establish the connections between the FSLP and other tree grammar models. Investigate the equivalence of FSLP under various theories/term rewriting operations. Investigate, how recompression approach generalises to FSLPs and what are the consequences (equivalence, approximation, applications—context unification).*

Applications of grammar compression One of the reasons why grammar compression is so successful is that it has important applications in other areas of computer science. In fact, some such applications can be treated as results for grammar compression itself, especially when the utilised techniques are based mostly on grammar compression and to less extent on the investigated problem.

The applicant wants to investigate the following problems, with a hope that grammar based techniques can be used to solve them. Faster (polynomial-time) algorithm for the satisfiability of equation with one variable in a free group; decision problems for the set of solutions of the context-unification. Computational complexity of quantified integer programming.

While the collection of tasks may seem somehow ad hoc and random, it is the varied applications that made grammar compression what it is now. The applicant has a good track of such applications and he would like to continue this.

Objectives: *Algorithms for word equations with one variable in free groups. Computational complexity of decision problems for the set of solutions of context unification. Computational complexity of quantified integer programming.*

2 Significance of the Project

Grammar compression In grammar compression we represent the input data, usually treated as a string w, as a context-free grammar generating a language consisting of a single word, i.e. w; thus grammar compression is a subclass of more general dictionary compression. For historical reasons, such a grammar is usually called a *Straight-Line Program*, or *SLP* for short. The obvious advantage of this approach (and in general: of dictionary compression) is that highly repetitive sequences can be represented very succinctly and some non-local repetitions can be also exploited. Grammar compression is strongly connected with block compression: the most common block compression algorithms are LZW and LZ77 — the former is in fact a grammar compressor and the latter can be translated into an SLP [13, 101].

The idea that SLP exploits and preserves the structure of the input works for linear data, but not for trees, for which the linearisation already destroys much of the structural information. Luckily, the grammar compression can be generalised also to trees, and the grammar model is known as tree SLP (or TSLP). It inherits most of the useful properties of SLPs [11, 29, 41, 41, 66, 85, 87] (and it is easy to see that is not worse than DAGs and up to exponentially better). However, the advantage of TSLP over linearisation and compression by SLPs is unobvious: it allows better preprocessing but the compression rate may be much worse [29]. While the TSLPs are the most popular grammar compression model, there are some arbitrary design choices made in their definition and other models are also known and investigated, each having some advantages and disadvantages over TSLPs: non-linear TSLPs [84], elementary ordered tree grammars [2], top-trees [8] and a recent model of forest grammars [37]. More general notion of graph grammar compressors was also developed [89].

Heuristics and approximation There are also disadvantages of grammar compression: the size of the smallest SLP cannot be approximated beyond a certain constant multiplicative factor [12, 101] and the corresponding decision problem is NP-hard [12, 110]. Still, there are many practical heuristics [5, 69, 73, 94, 111, 118] as well as many approximation algorithms for this problem [13, 61, 63, 101, 102]. The approximation algorithms are usually good enough for theoretical considerations while the heuristics perform well-enough in practice, so we can manage without the optimal grammar compressor.

Both the heuristical and approximation algorithms were analysed from theoretical point of view. Unfortunately, in both cases we are far from understanding them fully: For approximation algorithms there is a huge gap between the known lower bounds (small constants [12, 101]) and provable approximation bounds ($\mathcal{O}(\log N)$, where N is the length of the uncompressed text [13, 61, 63, 101, 102]). Bridging this gap from either side is probably one of the main open problems in the area, with potential impact on all applications of grammar compression.

The approximation ratio of the heuristics can also be analysed. Matching lower and upper bounds are known for some of them [13, 54], for others this is not the case; in particular this applies to Re-Pair [73], which is known for its good practical behaviour as well as various applications [16, 27, 46, 47, 68, 112].

On the other hand, other standard method for evaluating compressors is to compare how well they behave with regards to k-empirical entropy [71, 90]. This approach was applied to grammar compressors [93], but in general was not well-developed.

It is worth noting that so far the approximation and heuristical algorithms used completely different approaches and were far apart. However, recently theoretical insight from approximation algorithm was used to (slightly) improve the performance of the renowned Re-Pair algorithm [30].

Algorithms for compressed data However, the main significance of grammar compression lays elsewhere: compression of stored and transmitted data becomes more and more popular. However, this data still needs to be accessed and processed and decompressing it on each occasion may negate the gain of smaller storage. Thus a popular trend in algorithmic design is to develop algorithms that work directly on the compressed representation of data, without the need of full decompression [4, 26, 38–40, 43–45, 51, 56, 95]. SLP have simple inductive definitions, which makes particularly suitable for processing. Numerous efficient algorithms for string-processing primitives, like checking equality, finding occurrences or longest common prefix, etc., were developed; their main common feature is that their running time is polynomial in the size of the compressed representation, which may be logarithmic in size of the decompressed data. In fact, such algorithms are so easy and effective, that similar algorithms for popular block-compressed standards (so LZ77, LZW or LZ78) routinely first transform the block representation into the SLP representation and only afterwards perform some processing [38–40] (which, however, may be different than the generic algorithm for SLPs due to some additional structure introduced during the translation).

Processing compressed input made an impact on many areas, as the classical problems were investigated under the premiss that the input is given in a compressed form. Most notably, compressed membership problems were investigated in formal languages and automata theory [7, 79–81] as well as in combinatorial group theory [49, 50, 83, 88] (in which they even lead to answers to some classical problems [83]) and processing compressed strings is already a rich subfield of stringology. Similarly, term rewriting, unification with TSLP compressed input was investigated [19, 31, 34, 84, 107, 108].

SLP as a tool So far we described the compressed data more as a challenge: we have to process it and this is more difficult than for uncompressed input. A different approach, or rather—a paradigm—is to use compression as a tool: represent the problem and its solution as SLPs of well-bounded size and then apply the algorithmic techniques for SLPs as a blackbox. This idea was first implemented by Plandowski and Rytter [99] in their work on word equations. This became a game changing paradigm in areas, in which the processed data is regular enough so that its SLP-representation is provably small. Most notably those include: word equations (in semigroups and groups) [15, 21, 23–25, 64, 96, 97, 99], combinatorial group theory [49, 50, 83, 88] computational topology [103, 105], verification [20, 48, 52, 74], term rewriting and unification [32, 33, 35, 36, 58, 75, 76]. Yet this approach has applications in seemingly distant areas such as graph recognition [104] or finding hidden Markov model [78]; see a recent survey of Lohrey [82] for various applications.

The above approach is not problem-free: firstly, as the approach is divided into two steps, something can be lost in between and the analysis is not as tight as possible: (see for instance original Plandowski's argument for word equations: the initial algorithm [99] is very simple, but it requires a separate, involved bound on the size of the solution [96] and obtaining the PSPACE bound actually requires melding those two steps together [97] and using some dependencies between them). Secondly, since we use the algorithmic techniques as black boxes, we are restricted to the operations they provide and extending them might require rewriting the whole construction, see for instance [74]. Furthermore, the translation to SLP can itself lead to some unnecessary size increase, which again can be eliminated only by melding the whole thing together. Lastly, in some cases the needed combinatorial properties are too complex to be actually analysed; this was in particular notable for tree SLPs, for which the generalisations of some solved problems for strings were left open (most notably: context unification [17, 18, 58, 106]—generalisation of word equations, and approximation of the smallest

TSLP [11, 66]).

Recompression Those problems were answered in the recompression technique developed by the applicant. In essence, instead of separately expressing the problem as an SLP and then applying the algorithmic black box on top, we perform the SLP compression directly on the instance, modifying it appropriately so that such changes are indeed applicable. In particular, one then focuses more on the structure and properties of the representation rather than on the combinatorial problem underneath. The difference may seem small, but it is essential: this method is much more rooted in the investigations of grammars, as it puts very little attention to the combinatorics of the objects and much on the combinatorics of the SLPs.

This approach turned out to be extremely successful: on one hand it gave new, simpler proofs of existing results [24, 25, 61, 63, 64], more efficient algorithms for others [56, 60, 62, 65] as well as new results for old problems open for years [58, 66]. In particular, it allowed easy extension of results for strings to trees [58, 66] and simplified significantly the results for word equations [64, 65], especially in more involved cases of groups and traces [24, 25].

Consequences of the project goals, implications. Grammar compression is heavily utilised, both in practice and in theory, despite that there are many gaps in our knowledge about it. Fulfilling of proposed objectives will change this situation.

Better heuristics would mean that smaller grammars are used in applications, which immediately would speed them up. Moreover, even though we do use those heuristics, we still cannot tell, which one should be better and why. The objective of this project is to understand them better.

Improving the approximation algorithms would improve many theoretical results in varied areas of computer science that employ it as a first step; on the other hand, the more likely fact that a lower bound is found for those algorithms would mean that improvements in those applications have to be found somewhere else. A similar statement applies to algorithms that process SLP-compressed data: when the lower bound for equivalence of SLPs is shown, speeding up algorithms using it must resort to other means.

Better tree compressions models (FSLP) that are still equipped with all the tools of the older model (TSLP, toptree) mean that more applications can be found for tree compression and that it can be extended to areas in which it previously was not so successful, like for instance XML compression.

Still, this is a theory project, the most important is the understanding of grammar compression model.

On a more measurable side, the obtained results will be presented at conferences, depending on their quality, those would be ICALP, STACS or ESA—the highest ranking conferences in Europe, more practical results on grammar compression would be most likely presented on DCC, a conference devoted to compression, while other results would be presented at MFCS, DLT, CPM conferences. The corresponding full versions of would be submitted to appropriate journals.

3 Work Plan

Grammar compression and entropy It is known that the size of the smallest grammars cannot be approximated within a small constant factor [12, 13], however, in that work the size of the SLP is simply the sum of lengths of its rules. From practical point of view, it is equally important, how those rules are bit-encoded and there are many ways to do that [5, 46, 69, 73, 85]. This influences the actual construction of the heuristical compressors: for instance it is argued that binary rules can be represented more succinctly [73]. In particular, in Re-Pair entropy coder is used to encode part of the constructed grammar [73] and not using it leads to worse practical performance [28].

This naturally leads to the idea that heuristics for grammar compression should take the entropy into the account and such attempts were made, see for instance [111, 114], but the practical performance was improved only slightly.

Note that entropy may measure two things: in the classical information theory meaning, we are given a source of random symbols and entropy is measure on the distribution of those symbols. In particular, the text (and so its compressed size) is a random variable. On the other hand, the empirical entropy is a measure for a given string. There was an extensive work on grammar compression and entropy of the source [109, 113, 117, 118], the applicant believes that the investigations based on empirical entropy are more suitable for the analysis of grammar compressors, as they fit more into the usual worst-case paradigm in which we calculate the performance on a fixed input and take the supremum of those results. To recall, given a string w of length n over an alphabet of size σ the empirical entropy of w is $\sum_i n_i \log(n/n_i)$ where n_i is the number of occurrences of the i-th letter. The empirical entropy lower-bounds the size of the text if the only way to encode it is to assign

different bit strings to different symbols. There are ways (Huffman coding, arithmetic coding,...) to encode the text so that the size if the empirical entropy plus some smaller terms. This can be generalised to take into the account the *contexts* in an appropriate way, in which case we talk about the k-th empirical entropy, there different ways to define, it though [90]. The predictive coding, for instance PPM, can be seen as an application of this approach to compression.

Objective: *Construct a grammar compressor heuristic that utilises the information on empirical entropy.* There is no reason why empirical entropy should not be employed also in grammar compression. The applicant believes that one can use the information about the entropy to improve the performance of grammar compressors. This believe is backed by previous related work of the applicant: insights drawn from the recompression technique by the applicant allowed a practical improvement to the well-established Re-Pair grammar compressor [30]; on the other hand, the recent result on word equations [65] used non-trivial strategies for choosing pairs to be compressed and it allowed smaller bit-size encoding.

Objective: *Give (or improve) upper bounds on performance of popular heuristics for grammar compression in terms of empirical entropy.*

Comparisons with entropy are also used as a benchmark of compressors: we compare the k-th empirical entropy of a string and the bit-size of the output of the compressor on this string. Since the entropy is a lower bound on how well the string can be compressed (using a specific way of compression) one should expect that good compressors should not deviate from entropy too much: if the difference is too large then the compressors based on coding entropy would be clearly superior, which is not observed in practice. For practical compressors (other than the ones that actually are entropy coders) the obtained bounds are often coarse, but they do provide a theoretical reassurance for practically observed phenomena.

This approach was successfully used to analyse compressors based on Burrows-Wheeler transform [90], for which it was shown that they produce an output at most constantly larger than the k-th empirical entropy (for small enough k). The effectiveness of most popular block compressors of LZ77 and LZ78 was also evaluated in this setting [71] and similar results were obtained: i.e. their output can be larger than the H_0 by factors 3 and 8, respectively.

The LZ78 already is a grammar compressor and the LZ77 can be transformed to an SLP and vice-versa [13, 101]. This does not bear immediate results on grammar compressors, as those transformations do not count the size in bits and the LZ77 one also needs to encode of the offsets to previous occurrences, a problem which is not present in grammar compressors. Still, the structural lemmata developed in this work were also employed in the analysis of Re-Pair, for which it was shown that it produces an output twice the size of H_k (plus some terms of smaller order) [93]. However, the 'small additive' term can be relatively large and the previous work utilising this framework [71, 90] tried hard to avoid such additional terms.

The applicant already obtained some partial results which show that replacement of pairs in the text does not increase the k-th empirical entropy for $k \geq 1$ (which does not hold for $k = 0$, though). Hopefully this can be used to derive a more general bound for a larger class of grammar compressors and improve the bound for Re-Pair and similar grammar compressors.

Objective: *Construct a grammar compressor that achieves or approximates k-th empirical entropy (for small enough k).*

The ultimate goal of the above objective would be to show that some known grammar compressor achieves pure H_k empirical entropy for small ks, or in a worst case, approximates it within some small constant. However, this seems unrealistic, as those compressors were not devised with this in mind. We aim at constructing such a compressor. The applicant believes that he has a reasonable chance of achieving this goal: he has experience in approximation algorithms for grammar compression [61, 63] that use new techniques (recompression) and this technique recently proved useful in efficient bit-encoding (for word equations) [65].

Approximation ratio of grammar compressors Since construction of the smallest grammar is not feasible, we have to resort to approximation algorithms and heuristics. We need a measure to asses the quality of the obtained solutions and for approximation algorithms the standard one is the approximation ratio: given a word w we compare the grammar size returned by the algorithm, call it $A(w)$, and the optimal one, denote it by $\mathrm{OPT}(w)$; i.e. $|A(w)|/|\mathrm{OPT}(w)|$. Then the approximation ratio is the supremum over all words w. On the other hand, a *lower bound* k for the problem means that no algorithm with an approximation ratio smaller than k exists, unless some computational complexity hypothesis fails, usually $\mathsf{P} \neq \mathsf{NP}$.

While the heuristical algorithms are constructed so that they have good practical behaviour and not that it is possible to give proofs on their performance, we can calculate their approximation ratios as well. It is to be expected that they are worse than the corresponding ratios of approximation algorithms, but this it is good to

have some solid, theoretical argument about their performance and moreover this may help to narrow down the instances on which they perform poorly and either improve the heuristics or disregard those instances as not occurring in practice.

Objective: *Improve the lower bounds for the smallest SLP problem.* There are several algorithms for the smallest SLP problem that have an approximation ratio $\mathcal{O}(1 + \log(n/g))$ [13, 61, 63, 101, 102], where n is the length of the word and g the size of the smallest grammar for it (the size of the multiplicative constant hidden in the $\mathcal{O}$ notation is relatively small: $6, 84$ [63]). Those algorithms utilise different approaches and techniques and it seems hard to improve them.

There are strong reasons to believe that those results are nearly optimal: one can try to restrict the class of allowed strings and one of the simplest such subclasses arises when we allow only one letter in the alphabet. Then a word a^n is naturally identified with a number n and the SLPs over such words turn to the well-established notion of *addition chains*. The classic problem for the addition chain is a construction of a chain for a sequence of numbers: given a sequence of ℓ numbers $n_1, \ldots, n_\ell$, construct shortest sequence (called the *addition chain for* $n_1, \ldots, n_\ell$) $m_1, \ldots, m_k$ such that each m_i is either 1 or $m_i = m_{i'} + m_{i''}$ for some $i', i'' < i$ and each n_i is equal to some element in $m_1, \ldots, m_k$. It is easy to show that an addition chain for $n_1, \ldots, n_\ell$ is essentially an SLP for the word $ba^{n_1}ba^{n_2}b \cdots ba^{n_\ell}b$ and vice-versa [13]. A classic result [116] for addition chains constructs a sequence of length $\log(\max_i n_i) + \mathcal{O}(\sum \frac{\log n_i}{\log \log n_i})$. This in particular yields an $\mathcal{O}(\log n / \log \log n)$ approximation algorithm, which is also the currently best known. On the other hand, the size-bound on the addition chains are best possible, within a constant (this was claimed without a proof in [116], but it follows also from the argument in [54]). This implies that an approximation algorithm for the smallest SLP-problem with approximation ratio better than $\mathcal{O}(\log n / \log \log n)$ seems unlikely.

Furthermore, most of the approximation algorithms for the smallest SLP problem begin with the LZ77 encoding of the word, which is on one hand constructible in linear time and on the other known to be at most of the size of the smallest SLP [13, 101]. Then this encoding is somehow converted to an SLP, with a size increase $\mathcal{O}(\log(n/g))$. A Kolmogorov-complexity based argument [54] shows that some SLPs are at least $\mathcal{O}(\log n / \log \log n)$ larger, meaning that no approximation algorithm following such a path can achieve a better approximation ratio than $\mathcal{O}(\log n / \log \log n)$.

On the other hand, the lower bounds for the smallest SLP problem are far smaller and seem much less developed: only a constant lower bound of $8569/8568$ [101] is known. Furthermore, it uses an alphabet of unbounded size, which is not really practical. Recently a proof of NP-hardness for the decision problem was given also in the case of a fixed-size alphabet [12] (more specifically: alphabet size is 24). The authors of that work did not try to put convert their proof into the approximation lower bound, but this seems achievable with standard means.

This may lead to a belief that the approximation ratio may depend on the alphabet size, but it is known that if there is an $\mathcal{O}(1)$ approximation algorithm for the binary alphabet then there is $\mathcal{O}(1)$ approximation algorithm for any alphabet (even unbounded one) [6, 54]. Thus even if the approximation ratio depends on the alphabet, most likely none it is not constant anyway.

As the lower bounds for the smallest SLP problem seem underdeveloped, the applicant will try to improve them. A very ambitious goal is to try to also improve the approximation algorithms to $\mathcal{O}(\log n / \log \log n)$, but this is probably much more demanding.

Objective: *Improve lower and upper bounds on approximation ratios of heuristical grammar compressors.* There are several heuristics for grammar compression that perform well in practice, the most well known are: LZ78 [118], Bisection [70], Greedy [5], Sequential [115] (which is an improved version of Sequitur [94]), LongestMatch [69], Greedy [5], Re-Pair [73]. A systematic study of their approximation ratios was initiated in [101] and upper bounds on the approximation ratio were given, all of the form $\mathcal{O}((n/logn)^c)$ for appropriate constant $c \in \{\frac{1}{2}, \frac{2}{3}, \frac{3}{4}\}$. The technique used to upper-bound the approximation ratio is very general and it is hard to assume that it is tight, yet matching lower bounds for LZ78 and Bisection were given [54].

Except for above mentioned LZ78 and Bisection, the upper and lower bounds on approximation ratio of popular heuristics are far apart. Some hope is raised by a recent Kolmogorov complexity-based lower-bounded for the approximation ratio of Re-Pair [55] is $\mathcal{O}(\log n / \log \log n)$, given by the applicant, as this approach was not used in this context before. On the other hand, the working of Re-Pair makes it suitable for the analysis using the recompression technique by the applicant. For other heuristics: Bisection, Greedy and LongestMatch, they work in a different fashion and probably new techniques have to be developed to upper-bounds their approximation ratio.

Running time lower bounds Running time is one of the most important quality of an algorithm and many

try to improve algorithms for known problems by decreasing their running time. However, it is equally important to know when to stop trying and lower bounds are a formal answer to this problem, i.e we are looking for proofs that a given problem cannot be solved faster. Proving lower bounds is often challenging, in particular one needs to decide on the model, in which the lower bounds is shown, for instance decide what type of operations can the algorithm perform and what is their cost, can it use randomness and so on.

Despite numerous attempts many basic algorithmic problems with simple algorithms that are believed to be (almost) optimal do not have matching lower bounds. It was recently understood, that there is a good reason for that: a better algorithm for such a problem would falsify an open hypothesis in computational complexity theory. Being more specific, the *Strong Exponential Time Hypothesis* (*SETH* for short) [57] claims that for each $\epsilon > 0$ there there is $k > 0$ such that k-SAT with n variables cannot be solved in $\mathcal{O}(2^{(1-\epsilon)n})$ time. (There is also a weaker version of the hypothesis, known as *ETH*). In the recent years there is an explosion of work showing that assuming SETH (or better: ETH), many known algorithms for basic combinatorial problems are essentially optimal, those include for instance the orthogonal vectors, (combinatorial) k-Clique, 3-sum, edit distance, longest common subsequence, Fréchet distance, longest common weakly increasing subsequence and others. This field of research is usually known as *conditional lower bounds* or *fine-grained computational complexity*. What is remarkable is that with this approach problems that are within P are now comparable and we establish dependencies between them.

This line of research included also topics related to SLPs: matching or almost matching lower bounds (assuming SETH) for several problems for SLPs (membership problem for DFAs, substring Hamming distance, pattern matching with wildcards, longest common subsequence) were recently given [1].

Objective: *Give (conditional) lower bounds for the equivalence testing and fully compressed pattern matching of SLPs.* Most of applications of algorithms for SLPs employ the equivalence testing (given two SLPs decide, whether they define the same string) or more generally—compressed pattern matching (given two SLPs for strings s and t determine, whether $t = usv$ for some strings u, v). Much attention was given to those problems and the running time for them were gradually improved [45, 67, 77, 92, 95] up to a cubic algorithm (note that there was also an independent line of research on this topic outside of string community [3, 52, 91]). Currently, the best algorithms for those have running times $\mathcal{O}((|m| + |n|) \log M)$, where n, m are the sizes of the two SLPs and M is the size of the decompressed pattern [42, 60], or the shorter of the two strings, when we test for equivalence; the bounds of the former have to be slightly increased depending on the exact model of computation [60], the latter algorithm [42] is randomised but also works in dynamic setting, it can be derandomised at some slight running time expense [3].

Not much is known about the lower bounds for those problems. The aforementioned research on conditional lower bounds failed to generate results in this case. Note that matching bound of $\mathcal{O}((n + m) \log M)$ *is known* for the dynamic variant, in which also the randomisation is allowed [42]. However, the hardness really comes from a dynamic setting and it does not transfer easily to the static scenario. Note that proposing appropriate bound is also a challenge on its own: there were conjectured bounds for this problem ($\mathcal{O}(nm \log M)$ [77]) that were beaten later on.

Objective: *Give a (conditional) lower bound for the fully compressed membership problem of DFAs.*

The applicant is also going to investigate the fully compressed membership problem for DFAs [59, 100]. In this problem we are given an SLP compressed word w and a deterministic finite automaton A, whose transitions are also labelled with SLP-compressed words (some constraints on the transition labels need to be satisfied so that this automaton is truly deterministic). This problem was renowned for many years, as its (and a more general variant of the automaton being given as a nondeterministic automaton) computational complexity was not known and only recently was shown to be in P (NP, respectively) by the applicant [59]. The problem is close to the heart of the applicant and interesting, as the P algorithm is nontrivial, so lower bounds for it would be interesting. Note that the upper bounds for this problem were not pushed to the limit, so this work would probably entail improving the upper bounds on this problem.

Objective: *Give (conditional) for other problems related do SLPs.* Besides those two particular problems, the applicant would like to study the conditional lower bounds for other SLP-related problems: the field is rich with problems and algorithms for them and the new results on conditional lower bounds suggest that more can be shown in this setting.

Forest grammars Tree SLPs are a good grammar compression model for trees, as most of the important properties of SLPs generalise to them: there is a practical compressors using it [85], their equivalence can be tested and in general algorithms can be performed on them [11, 29], they have an approximation algorithm [41, 66], there is a tree variant of LZ77 [41] that is related to TSLPs, they were applied in order to solve old open

problems [58], etc.

Yet, TSLPs have drawbacks. The main one concerns the arity: in the TSLP model each node label determines the arity, which is explicitly or implicitly given in the signature. This is not the case in XML—one of the main application and source of examples for the structure-preserving compression—in which the arity of nodes is not a priori bounded. In a similar spirit, when TSLPs are used in connection with term rewriting or arithmetic circuits [53], its labels may represent associative operations, say addition or multiplication, and then it is usually better to treat them a big addition or multiplication as a single node with a large degree, which is not a priori bounded, rather then as a collection of several binary nodes. There are ways to walk around this, for instance: first child next sibling encoding (fcns), in which the output is a binary tree and in the encoding the first child becomes left child and the right sibling becomes the right child. But fcns (and other possible walk-arounds) tend to destroy the structure of the tree, and preserving this structure is the main reason to use the TSLPs in the first place.

A different solution of *Forest SLPs* (FSLPs) was recently proposed [37], which formalise the grammar for trees in a different way: Nonterminals represent forests with at most one designated node—a place to plug in other forests. Two types of 'concatenation' are allowed: *horizontal*, which simply creates a new forest out of two ones, and *vertical*, in which all roots of the trees in the bottom forest become children of the designated node. Similar formalism was investigated before [9], but there it was used to define regular expressions over trees, in particular, no investigation of the grammar model was done before.

This model clearly generalises TSLPs as a TSLP can be converted to an FSLP and it seems very promising. So far only few results on this model were obtained. The applicant wants to investigate some of its properties.

Objective: *Establish the connections between the FSLP and other tree grammar models.* Each TSLP can be seen as a FSLP of a special kind and TSLP for fcns encoding can be converted to an FSLP [37]. There are natural questions to ask: does a backward transformation exist as well? Other well known and investigated formalism for tree compression are the *toptrees* [8]. Is there any connection or transformation between those two models? As this model is new, not much is known and the applicant hopes to establish the answers to those questions.

Objective: *Investigate the equivalence of FSLP under various theories/term rewriting operations.* The FSLP model was mainly introduced to answer a specific question: given two TSLPs, are the defined trees equivalent, if some node-labels denote associative and some commutative symbols [37]? Here a label a is commutative if $a(t, t')$ and $a(t', t)$ are equivalent terms and it is associative if $a(t, a(t', t''))$ and $a(a(t, t'), t'')$ are; the former generalises earlier work on equivalence of unordered trees [86].

This can be seen as a simple case of equivalence of terms under theory, in which we are given some (usually simple) theory (here—associativity and commutativity) and ask whether two terms are equivalent under it. In general, various problems can be stated using such an approach—for instance, equivalence under distributivity, associativity and commutativity generalises the polynomial identity testing, for which no polynomial-time algorithms are known (though there are randomised ones). So we cannot hope to have a general solution to this problem, even without the compression. Still, it seems plausible that this should be doable for other operations than just associativity and commutativity, say idempotent function symbols or distributivity. The applicant believes that this can be decided when the equivalence is defined using simple-enough rewriting rules, with the exact meaning being determined during the investigation.

Objective: *Investigate, how recompression approach generalises to FSLPs and what are the consequences (equivalence, approximation, applications—context unification).* The main techniques used for FSLPs so far (and also in the research that inspired it [86]) are based on canonical representations of trees and performing the canonisation directly on TSLPs/FSLPs. The applicant wants to investigate, whether the recompression approach can be also defined for FSLPs. At first sight this seems plausible, as the model is similar to TSLPs and SLPs, for which the approach works was very successful. If indeed the recompression can be generalised to FSLPs, the consequences would probably entail alternative equivalence testing algorithms, approximation algorithms and possibly an application to context unification problem, for which the TSLPs were applied before [58]. In general: for the satisfiability of context unification was shown to be decidable [58], it is still much less understood than word equations, which it generalises: for instance, the representation of all solutions remains out of reach, the bounds on space complexity are much more robust than for word equations [65] and subclasses defined by limiting the number of variables have unknown computational complexity. There is a chance that all this is due to a bad tool being used—a more robust notion of FSLPs may shed some light on those problems.

Applications of grammar compression Probably the most surprising and successful at the same time

application of grammar compression was its usage in the decidability proofs for word equations [25, 62, 64, 65, 96–99] and context unification [35, 36, 58]. Many of those new results used the recompression technique and its main feature is the change of paradigm: instead of dealing with the combinatorics of the object (words or terms) we deal with combinatorics of its compressed representation. It seems that the old and new paradigm are somehow apart, yet there are initial work that combine them both, that is, at the same time we exploit the properties of the compressed form and the instance [21, 62].

The applicant feels that he has enough expertise in the method so that new results should follow. In some sense the problems listed below may seem as an odd collection and hardly constitute a group of connected topics, yet, they are seem approachable using developed tools.

Objective: *Algorithms for word equations with one variable in free groups.* One of the ways to solve equations in groups is to reduce to equations in a corresponding semigroup (with involution and regular constraints); this approach was applied to free groups [23] and it can be extended to larger classes of groups, for instance, to right-angled Artin groups [22, 24]. The computational complexity of equations over free groups remains open, the wide-believed containment of word equations (over free semigroup) in NP would naturally imply that also equations over free groups are in NP. Yet, it is also believed that those equations should be somehow easier, due to additional group structure; this is supported by the fact that quadratic equations in free groups are in NP, while a similar problem for free groups remains open.

There are simple subclasses, which are known to be in P, the simplest one is the equation with one variable [10] (the case of two variables case is also in P). This solution, in contrast to previous ones that used group-theoretic tools, is based on a reduction to the word case and string-combinatorics. Yet, the obtained algorithm has quite high exponent (20) and it seems that the recompression approach together with some tools of string combinatorics (in particular those related to treatment of SLPs) should allow reduction of this exponent.

Objective: *Computational complexity of decision problems for the set of solutions of context unification.* The set of solutions for a word equation has a finite representation [64, 98] (as a graph, whose edges represent transformations of the solutions) which allows answers to standard questions about the solution set, for instance, is the number of solutions finite? Existence of a similar representation for context unification is unknown and the applicant doubts it.

Thus there is not default way to proceed with questions concerning the solution set of a context unification. The applicant would like to investigate those problems.

Objective: *Computational complexity of quantified integer programming.* It is a folklore knowledge that Integer Programming, i.e. a satisfiability of system of linear equations in which all coefficients and variables are integers, is NP-complete. Despite this knowledge, only recently [14] it has been shown that a quantified version of this problem, in which the variables are bound by universal or existential quantifiers, has also a neat computational complexity description: a subclass with k alternation of quantifiers is complete for the k-th level of ~~arithmetical~~ polynomial hierarchy (the containment is easily seen, the hardness requires an involved proof). Intuitively, the bound on the whole problem, without the restriction on the number of the quantifier alternation, should be PSPACE, yet it does not follow from the proof.

There are many, already folklore, proofs that IP is in NP and they are all based on bounding the size of the solution, the same strategy was also used in the aforementioned work on quantified IP [14]. There is a different proof [64, Chapter 4], which an approach similar to recompression: it iteratively halves the variables and constants and argues that the resulting equation can be kept in polynomial space. This yields a weaker result: IP is shown to be in PSPACE, but no bound on the size of the solution is used. On the other hand, this approach has its own merits in terms of word equations: it eliminates the need of using the bound on the exponent of periodicity [72], which in turn heavily depended on involved estimations of minimal solution sizes for IP. Hopefully this approach will eliminate the need of such estimations also in the case of quantified IP. Note that this approach cannot be generalised straight away, as bits of different variables depend on each other in a more complex way. Still the applicant believes that such dependencies should be somehow bounded and that this method can be extended also to the quantified case.

4 Methodology

Since part of the project involves implementation and testing, the methodology is split into two parts.

Evaluation of implementations For implementations of grammar compressors, we are going to test them on two corpora: the *Pizza&Chili* corpus (`http://pizzachili.dcc.uchile.cl/`) and the *Canterbury Corpus* (`http://corpus.canterbury.ac.nz/`). The compressor will be evaluated by comparing the

bit sizes of the output; when comparing with grammar compressors, also the ratio of sizes of sizes of the grammars will be compared.

The implementations are proofs of concept rather than industry standards, thus they are not going to be heavily optimised in terms of speed. Nevertheless, running times will be compared with standard compressors and asymptotic running time upper bounds are going to be established.

Theoretical work In the more theoretical part, we will seek mathematically provable bounds. The the of methodology depends on the objective group.

Grammar compressors and entropy Here we will employ the methodology of empirical k-entropy. The H_0 empirical entropy is calculated as follows: given a string w of length n over an alphabet Σ of the empirical entropy is: $H_0(w) = \sum_{a\in\Sigma} n_a \log(n/n_a)$, where n_a is the number of occurrences of a in w. This definition extends to higher empirical entropy: for a fixed sequence c of length k let w_c be the set of letters that are directly proceeded by c in w. Then $H_k(w) = \sum_{c\in\Sigma^k} H_0(w_c)$. This definition is also sometimes extended to take into the account that sequence of the same letter also carries some information (namely: length), which is lost in the empirical entropy [90]. Those notions are usually considered only for *small enough* k, which usually means that we consider only $k = o(\log n)$, as for $k = \Theta(\log n)$ and $c \in \Sigma^k$ most of w_c is of length 1 and so the H_k brings no information for the large piece of the input word.

Approximation ratio of grammar compressors We utilise the methodology of approximation ratio as a tool to evaluate the performance of the algorithm. Given a word w we can compare the size of the SLP returned by the algorithm, call it $A(w)$, and the optimal one, denote it by $\mathrm{OPT}(w)$; i.e. $|A(w)|/|\mathrm{OPT}(w)|$. Then the approximation ratio is the supremum over all words w.

A lower bound for approximation ratio is a tool to show that approximation ratio needs to be at least some function (of input size), as otherwise some computational complexity assumption fails (most often: $\mathsf{P} \neq \mathsf{NP}$).

For particular algorithm showing lower and upper bounds on their approximation ratios may require specific tools. A novel and useful approach for upper bounds is the recompression [61, 63, 66] which is based on performing simple compression operations on the instance and (as a though experiment) on the optimal SLP. Then the size of the grammar is bounded in terms of the modified optimal SLP rather than in terms of the instance.

Some recent lower bounds on grammar size are based on a classic notion of Kolmogorov complexity [54, 55]. In this methodology, related to information science, we use the fact that for each n there are binary words of length n such that they cannot be described (in particular: by an SLP plus a programme producing a word from an SLP) with less than n bits.

Running time lower bounds Proving lower bounds requires a model, which describes what are the allowed operations. We are going to work assuming SETH, which in essence says that SAT with n variables cannot be solved in time $\mathcal{O}(2^{(1-\epsilon)n})$. To use it one has to encode the instance of SAT in the instance of ones problem (in our case: equivalence of SLPs) so that fast answer would imply the falseness of SETH. As several hardness results were already shown using SETH, it is often enough to reduce another problem, for which we know a lower bound assuming SETH, for instance, Orthogonal Vectors, in which we are given a set of binary vectors and we are to decide, whether there are two such that at each position at least one of them has a 0.

There are also other lower bound techniques, which are not provably weaker, but they seem less likely to give results for the considered problem.

Forests grammars For the analysis of FSLPs we will use standard methodology: we will compare the sizes of smallest FSLP and TSLP (also in case of fcns encoding) or toptree for a given tree and calculate the supremum of those coefficients We will also employ the recompression methodology, described earlier.

Applications of grammar compression For the equations with one-variable, the applicant wants to use the combinatorial methodology of analysing the combinatorics of the reduced words. In particular, the applicant wants to reduce the analysis of the group case to the one of the semigroup and then use the two most successful approaches: one based on string combinatorics, in which the strings are analysed from the point of view of periodicity; and the approach using recompression, in which the strings are processed using simple compression operations.

For the problems related to context unification, the applicant wants to use the recompression methodology, in particular represent the solutions in terms of TSLPs or FSLPs. Similar methodology will hopefully work also in case of quantified IP.

5 Appointing a new scientific team

The applicant plans to hire 2 PhD students, each for 4 years, which is a standard time for a completion of a PhD programme; the whole project lasts 5 years, yet some delays in finding PhD students, administration procedures etc. are to be expected and this should cover the whole time of the project. The call for PhD students will be an open one, in particular open to students from abroad.

It is also planned to hire a postdoc for a period of 2 years, with an initial period of one year and possible extension to the second year or the position being passed to a different candidate. No formal preference will be given, but the applicant would like to hire someone with earlier international experience, i.e. PhD obtained outside Poland or postdoc position outside Poland. Due to strong competition on the postdoc market, the applicant expects some delays and vacancies, thus a total 2 year period.

The applicant would like to encourage master students to take part in the research, which hopefully would lead to a publishable master thesis; the applicant has such an experience before with his current PhD student, whose master thesis was published [30]. Moreover, the applicant believes that introducing master students into research increases the chances of theirs joining the academia, which is a worthy goal on its own. To this end smaller 1-year scholarships for total of 3 master students are secured. The applicant believes that he will have 3 good master students during the project.

Finally, the applicant expects that he will be able to draw attention of other researchers at this institute so that they would collaborate on parts of the projects. To this end some additional funds for 48 months are secured.

None of the participants will be a priori assigned to particular tasks, but rather depending on their interests and progress on particular objectives. In particular this should help them collaborate on particular tasks.

The applicant believes that this size of the research team is appropriate, as it is big enough to fulfil the projects objectives and at the same time not too big, so that the applicant can grasp, what is being done in the group.

6 Literature references

[1] A. Abboud, A. Backurs, K. Bringmann, and M. Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In *FOCS*, 2017. to appear.

[2] T. Akutsu. A bisection algorithm for grammar-based compression of ordered trees. *Inf. Process. Lett.*, 110(18–19):815–820, 2010.

[3] S. Alstrup, G. S. Brodal, and T. Rauhe. Pattern matching in dynamic texts. In D. B. Shmoys, editor, *SODA*, pages 819–828, 2000.

[4] A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in z-compressed files. *Journal Computational Systems and Sciences*, 52(2):299–307, 1996.

[5] A. Apostolico and S. Lonardi. Some theory and practice of greedy off-line textual substitution. In *DCC*, pages 119–128, 1998.

[6] J. Arpe and R. Reischuk. On the complexity of optimal grammar-based compression. In *DCC*, pages 173–182, 2006.

[7] M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: From word to circuit evaluation. *SIAM J. Comput.*, 26(1):138–152, 1997.

[8] P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. *Infmormation and Computation*, 243:166–177, 2015.

[9] M. Bojańczyk and I. Walukiewicz. Forest algebras. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas].*, volume 2 of *Texts in Logic and Games*, pages 107–132, 2008.

[10] D. Bormotov, R. Gilman, and A. Myasnikov. Solving one-variable equations in free groups. *Journal of Group Theory*, 12:317–330, 2009.

[11] G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.

[12] K. Casel, H. Fernau, S. Gaspers, B. Gras, and M. L. Schmid. On the complexity of grammar-based compression over fixed alphabets. In I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors, *ICALP*, volume 55 of *LIPIcs*, pages 122:1–122:14, 2016.

[13] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

[14] D. Chistikov and C. Haase. On the complexity of quantified integer programming. In I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, editors, *ICALP*, volume 80 of *LIPIcs*, pages 94:1–94:13, 2017.

[15] L. Ciobanu, V. Diekert, and M. Elder. Solution sets for equations over free groups are EDT0L languages. *IJAC*, 26(5):843–886, 2016.

[16] F. Claude and G. Navarro. A fast and compact web graph representation. In N. Ziviani and R. A. Baeza-Yates, editors, *SPIRE*, volume 4726 of *LNCS*, pages 118–129, 2007.

[17] H. Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *J. Symb. Comput.*, 25(4):397–419, 1998.

[18] H. Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *J. Symb. Comput.*, 25(4):421–453, 1998.

[19] C. Creus, A. Gascón, and G. Godoy. One-context unification with STG-compressed terms is in NP. In A. Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, volume 15 of *LIPIcs*, pages 149–164, Dagstuhl, Germany, 2012.

[20] W. Czerwiński and S. Lasota. Fast equivalence-checking for normed context-free processes. In K. Lodaya and M. Mahajan, editors, *FSTTCS*, volume 8 of *LIPIcs*, pages 260–271, 2010.

[21] V. Diekert and M. Elder. Solutions of twisted word equations, EDT0L languages, and context-free groups. In I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, editors, *ICALP*, volume 80 of *LIPIcs*, pages 96:1–96:14, 2017.

[22] V. Diekert and A. Muscholl. Solvability of equations in graph groups is decidable. *IJAC*, 16(6):1047–1070, 2006.

[23] V. Diekert, C. Gutiérrez, and C. Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Inf. Comput.*, 202(2):105–140, 2005.

[24] V. Diekert, A. Jeż, and M. Kufleitner. Solutions of word equations over partially commutative structures. In I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors, *ICALP*, volume 55 of *LIPIcs*, pages 127:1–127:14, 2016.

[25] V. Diekert, A. Jeż, and W. Plandowski. Finding all solutions of equations in free groups and monoids with involution. *Inf. Comput.*, 251:263–286, 2016.

[26] M. Farach and M. Thorup. String matching in Lempel-Ziv compressed strings. *Algorithmica*, 20(4): 388–404, 1998.

[27] J. Fischer, V. Mäkinen, and G. Navarro. Faster entropy-bounded compressed suffix trees. *Theor. Comput. Sci.*, 410(51):5354–5364, 2009.

[28] P. Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, Feb. 1994.

[29] M. Ganardi, D. Hucke, M. Lohrey, and E. Noeth. Tree compression using string grammars. In E. Kranakis, G. Navarro, and E. Chávez, editors, *LATIN*, volume 9644 of *LNCS*, pages 590–604, 2016.

[30] M. Gańczorz and A. Jeż. Improvements on re-pair grammar compressor. In A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, editors, *DCC*, pages 181–190, 2017.

[31] A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context matching for compressed terms. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 93–102, 2008.

[32] A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification with singleton tree grammars. In R. Treinen, editor, *RTA*, volume 5595 of *LNCS*, pages 365–379, 2009.

[33] A. Gascón, G. Godoy, M. Schmidt-Schauß, and A. Tiwari. Context unification with one context variable. *J. Symb. Comput.*, 45(2):173–193, 2010.

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

[35] A. Gascón, M. Schmidt-Schauß, and A. Tiwari. Two-restricted one context unification is in polynomial time. In S. Kreutzer, editor, *CSL*, volume 41 of *LIPIcs*, pages 405–422, 2015.

[36] A. Gascón, A. Tiwari, and M. Schmidt-Schauß. One context unification problems solvable in polynomial time. In *LICS*, pages 499–510, 2015.

[37] A. Gascón, M. Lohrey, S. Maneth, and C. P. Reh. Checking isomorphism of compressed forests. 2017. unpublished, submitted to a conference.

[38] P. Gawrychowski. Pattern matching in Lempel-Ziv compressed strings: fast, simple, and deterministic. In C. Demetrescu and M. M. Halldórsson, editors, *ESA*, volume 6942 of *LNCS*, pages 421–432, 2011.
[39] P. Gawrychowski. Tying up the loose ends in fully LZW-compressed pattern matching. In C. Dürr and T. Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 624–635, 2012.
[40] P. Gawrychowski. Optimal pattern matching in lzw compressed strings. *ACM Transactions on Algorithms*, 9(3):25, 2013.
[41] P. Gawrychowski and A. Jeż. LZ77 factorisation of trees. In A. Lal, S. Akshay, S. Saurabh, and S. Sen, editors, *FSTTCS*, volume 65 of *LIPIcs*, pages 35:1–35:15, 2016.
[42] P. Gawrychowski, A. Karczmarz, T. Kociumaka, J. Lacki, and P. Sankowski. Optimal dynamic strings. *CoRR*, abs/1511.02612, 2015.
[43] L. Gąsieniec and W. Rytter. Almost optimal fully LZW-compressed pattern matching. In *DCC*, pages 316–325, 1999.
[44] L. Gąsieniec, M. Karpiński, W. Plandowski, and W. Rytter. Randomized efficient algorithms for compressed strings: The finger-print approach. In D. S. Hirschberg and E. W. Myers, editors, *CPM*, volume 1075 of *LNCS*, pages 39–49, 1996.
[45] L. Gąsieniec, M. Karpiński, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. In R. G. Karlsson and A. Lingas, editors, *SWAT*, volume 1097 of *LNCS*, pages 392–403, 1996.
[46] R. González and G. Navarro. Compressed text indexes with fast locate. In B. Ma and K. Zhang, editors, *CPM*, volume 4580 of *LNCS*, pages 216–227, 2007.
[47] R. González, G. Navarro, and H. Ferrada. Locally compressed suffix arrays. *J. Exp. Algorithmics*, 19: 1.1:1.1–1.1:1.30, Jan. 2015.
[48] S. Gulwani and A. Tiwari. Computing procedure summaries for interprocedural analysis. In R. D. Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 253–267, 2007.
[49] N. Haubold and M. Lohrey. Compressed word problems in hnn-extensions and amalgamated products. *Theory Comput. Syst.*, 49(2):283–305, 2011.
[50] N. Haubold, M. Lohrey, and C. Mathissen. Compressed decision problems for graph products and applications to (outer) automorphism groups. *IJAC*, 22(8), 2012.
[51] M. Hirao, A. Shinohara, M. Takeda, and S. Arikawa. Fully compressed pattern matching algorithm for balanced straight-line programs. In *SPIRE*, pages 132–138, 2000.
[52] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theor. Comput. Sci.*, 158(1&2):143–159, 1996.
[53] D. Hucke, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. In V. Raman and S. P. Suresh, editors, *FSTTCS*, volume 29 of *LIPIcs*, pages 457–468, 2014.
[54] D. Hucke, M. Lohrey, and C. P. Reh. The smallest grammar problem revisited. In S. Inenaga, K. Sadakane, and T. Sakai, editors, *SPIRE*, volume 9954 of *LNCS*, pages 35–49, 2016.
[55] D. Hucke, A. Jeż, and M. Lohrey. Approximation ratio of repair. *CoRR*, abs/1703.06061, 2017.
[56] T. I. Longest common extensions with recompression. In J. Kärkkäinen, J. Radoszewski, and W. Rytter, editors, *CPM*, volume 78 of *LIPIcs*, pages 18:1–18:15, 2017.
[57] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
[58] A. Jeż. Context unification is in PSPACE. In E. Koutsoupias, J. Esparza, and P. Fraigniaud, editors, *ICALP*, volume 8573 of *LNCS*, pages 244–255, 2014.
[59] A. Jeż. The complexity of compressed membership problems for finite automata. *Theory of Computing Systems*, 55:685–718, 2014.
[60] A. Jeż. Faster fully compressed pattern matching by recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, Jan 2015.
[61] A. Jeż. Approximation of grammar-based compression via recompression. *Theoretical Computer Science*, 592:115–134, 2015.
[62] A. Jeż. One-variable word equations in linear time. *Algorithmica*, 74:1–48, 2016.
[63] A. Jeż. A *really* simple approximation of smallest grammar. *Theoretical Computer Science*, 616:141–150, 2016.
[64] A. Jeż. Recompression: a simple and powerful technique for word equations. *J. ACM*, 63(1):4:1–4:51, Mar 2016.
[65] A. Jeż. Word equations in nondeterministic linear space. In I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, editors, *ICALP*, volume 80 of *LIPIcs*, pages 95:1–95:13, 2017.
[66] A. Jeż and M. Lohrey. Approximation of smallest linear tree grammar. *Information and Computation*,

251:215–251, 2016.
[67] M. Karpiński, W. Rytter, and A. Shinohara. Pattern-matching for strings with short descriptions. In *CPM*, volume 937 of *LNCS*, pages 205–214, 1995.
[68] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage system: a unifying framework for compressed pattern matching. *Theor. Comput. Sci.*, 1(298):253–272, 2003.
[69] J. C. Kieffer and E. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Information Theory*, 46(3):737–754, 2000.
[70] J. C. Kieffer, E. Yang, G. J. Nelson, and P. C. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Information Theory*, 46(4):1227–1245, 2000.
[71] S. R. Kosaraju and G. Manzini. Compression of low entropy strings with lempel-ziv algorithms. *SIAM J. Comput.*, 29(3):893–911, 1999.
[72] A. Kościelski and L. Pacholski. Complexity of Makanin's algorithm. *J. ACM*, 43(4):670–684, 1996.
[73] N. J. Larsson and A. Moffat. Offline dictionary-based compression. In J. A. Storer and M. Cohn, editors, *Data Compression Conference*, pages 296–305, 1999.
[74] S. Lasota and W. Rytter. Faster algorithm for bisimulation equivalence of normed context-free processes. In R. Královič and P. Urzyczyn, editors, *MFCS*, volume 4162 of *LNCS*, pages 646–657, 2006.
[75] J. Levy, M. Schmidt-Schauß, and M. Villaret. The complexity of monadic second-order unification. *SIAM J. Comput.*, 38(3):1113–1140, 2008.
[76] J. Levy, M. Schmidt-Schauß, and M. Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011.
[77] Y. Lifshits. Processing compressed texts: A tractability border. In B. Ma and K. Zhang, editors, *CPM*, volume 4580 of *LNCS*, pages 228–240, 2007.
[78] Y. Lifshits, S. Mozes, O. Weimann, and M. Ziv-Ukelson. Speeding up HMM decoding and training by exploiting sequence repetitions. *Algorithmica*, 54(3):379–399, 2009.
[79] M. Lohrey. Word problems and membership problems on compressed words. *SIAM Journal of Computing*, 35(5):1210–1240, 2006.
[80] M. Lohrey. Compressed membership problems for regular expressions and hierarchical automata. *International Journal of Foundations of Computer Science*, 21(5):817–841, 2010.
[81] M. Lohrey. Leaf languages and string compression. *Inf. Comput.*, 209(6):951–965, 2011.
[82] M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2): 241–299, 2012.
[83] M. Lohrey and S. Schleimer. Efficient computation in groups via compression. In V. Diekert, M. V. Volkov, and A. Voronkov, editors, *CSR*, volume 4649 of *LNCS*, pages 249–258, 2007.
[84] M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012.
[85] M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Inf. Syst.*, 38 (8):1150–1167, 2013.
[86] M. Lohrey, S. Maneth, and F. Peternek. Compressed tree canonization. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *ICALP*, volume 9135 of *LNCS*, pages 337–349, 2015.
[87] M. Lohrey, S. Maneth, and C. P. Reh. Traversing grammar-compressed trees with constant delay. In A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, editors, *2016 Data Compression Conference, DCC 2016, Snowbird, UT, USA, March 30 - April 1, 2016*, pages 546–555, 2016.
[88] J. MacDonald. Compressed words and automorphisms in fully residually free groups. *International Journal of Automation and Computing*, 20(3):343–355, 2010.
[89] S. Maneth and F. Peternek. Compressing graphs by grammars. In *ICDE*, pages 109–120, 2016.
[90] G. Manzini. An analysis of the burrows-wheeler transform. *J. ACM*, 48(3):407–430, 2001.
[91] K. Mehlhorn, R. Sundar, and C. Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997.
[92] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *CPM*, volume 1264 of *LNCS*, pages 1–11, 1997.
[93] G. Navarro and L. M. S. Russo. Re-pair achieves high-order entropy. In *DCC*, page 537, 2008.
[94] C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res. (JAIR)*, 7:67–82, 1997.
[95] W. Plandowski. Testing equivalence of morphisms on context-free languages. In J. van Leeuwen, editor, *ESA*, volume 855 of *LNCS*, pages 460–470, 1994.
[96] W. Plandowski. Satisfiability of word equations with constants is in NEXPTIME. In *STOC*, pages

721–725, 1999.

[97] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004.

[98] W. Plandowski. An efficient algorithm for solving word equations. In J. M. Kleinberg, editor, *STOC*, pages 467–476, 2006.

[99] W. Plandowski and W. Rytter. Application of Lempel-Ziv encodings to the solution of word equations. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *ICALP*, volume 1443 of *LNCS*, pages 731–742, 1998.

[100] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In J. Karhumäki, H. A. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels are Forever*, pages 262–272, 1999.

[101] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.

[102] H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *J. Discrete Algorithms*, 3(2-4):416–430, 2005.

[103] M. Schaefer, E. Sedgwick, and D. Stefankovic. Algorithms for normal curves and surfaces. In O. H. Ibarra and L. Zhang, editors, *COCOON*, volume 2387 of *LNCS*, pages 370–380, 2002.

[104] M. Schaefer, E. Sedgwick, and D. Stefankovic. Recognizing string graphs in NP. *J. Comput. Syst. Sci.*, 67(2):365–380, 2003.

[105] M. Schaefer, E. Sedgwick, and D. Stefankovic. Computing dehn twists and geometric intersection numbers in polynomial time. In *CCCG*, 2008.

[106] M. Schmidt-Schauß. Unification of stratified second-order terms. Internal Report 12/94, Johann-Wolfgang-Goethe-Universität, 1994.

[107] M. Schmidt-Schauß. Matching of compressed patterns with character-variables. In A. Tiwari, editor, *RTA*, volume 15 of *LIPIcs*, pages 272–287, 2012.

[108] M. Schmidt-Schauß, D. Sabel, and A. Anis. Congruence closure of compressed terms in polynomial time. In *FroCos*, volume 6989 of *LNCS*, pages 227–242, 2011.

[109] D. Sheinwald. On the ziv-lempel proof and related topics. *Proceedings of the IEEE*, 82(6):866–871, Jun 1994.

[110] J. A. Storer and T. G. Szymanski. The macro model for data compression. In R. J. Lipton, W. A. Burkhard, W. J. Savitch, E. P. Friedman, and A. V. Aho, editors, *STOC*, pages 30–39, 1978.

[111] M. Vasinek and J. Platos. Prediction and evaluation of zero order entropy changes in grammar-based codes. *Entropy*, 19(5):223, 2017.

[112] R. Wan. Browsing and searching compressed documents.

[113] A. D. Wyner and J. Ziv. The sliding-window lempel-ziv algorithm is asymptotically optimal. *Proceedings of the IEEE*, 82(6):872–877, Jun 1994.

[114] E. Yang and D. He. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform .2. with context models. *IEEE Trans. Information Theory*, 49(11):2874–2894, 2003.

[115] E. Yang and J. C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform - part one: Without context models. *IEEE Trans. Information Theory*, 46 (3):755–777, 2000.

[116] A. C.-C. Yao. On the evaluation of powers. *SIAM J. Comput.*, 5(1):100–103, 1976.

[117] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977.

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