Objectives

Applicant's recent research [32, 33, 31, 37, 38, 13, 36, 35, 34] mostly investigated interplay between compression and seemingly unrelated branches of computer science, like algorithm design, formal languages, logic, etc., but also researched topics related to minimisation and small representations of DFAs [21, 22, 39, 40, 34] (note that some belong to both classes [34]). It turned out that an approach based on applying simple compression rules to problems and analysing the form of the instances applies to surprisingly many different problems. The crucial feature of this method, called recompression in the following, is that it shifts focus in the problem: instead of trying to understand the inner combinatorics of the underlying structure and exploit it to efficiently solve the problem, we analyse and modify the way the instance is represented, enforcing that at the same time the representation remains small (usually: linear in the input size) and the instance is gradually simplified.

As a result, at the moment we understand several topics, like word equations, grammar compression, fully compressed pattern matching etc., much better than before. In particular, this approach yielded a steady stream of papers, sometimes reproving existing results in a simpler form, sometimes presenting new results and sometimes yielding breakthroughs in the area.

The applicant would like to continue this line of research, as he is certain that several new results can be obtained using a similar approach: as said, so far the stream of results was quite steady and it is reasonable to believe it will be so for some foreseeable future. Moreover, the research on representations of finite automata is more and more seen as a compression and so is much more related than previously thought; the applicant wants to explore this field again and hopes to find more common topics or techniques in those areas.

To be more specific, the applicant plans to investigate the following problems, which in most cases are generalisations, extensions or akin to problems that were successfully solved using compression-based approach.

1. Context unification:
   (a) What is the computational complexity of the case with one context variable?
   (b) What is the computational complexity of the case with two context variables?
   (c) Is the context unification with regular constraints decidable?
   (d) Is it possible to give a finite description of all solutions of a context unification instance?
2. One-step term rewriting:
   (a) Is the existential theory of one-step term rewriting decidable?
   (b) Is the positive theory one-step term rewriting decidable?
3. Word equations:
   (a) How efficient the algorithms deciding word equations for two variables can be?
   (b) What is the complexity of quadratic word equations?
4. Free inverse monoids:
   (a) Are the equations with one variable over free inverse monoids decidable?
   (b) Is the extendibility problem for free inverse monoids DEXPTIME-hard?
5. Algorithms on compressed trees:
   (a) Is there an efficient data structure for checking the equality of dynamic trees?
   (b) What is the computational complexity of equality testing for non-linear grammars (is it in NP)?
   (c) Is the compressed submatching for non-linear patterns in P?
6. Synchronisation of prefix codes:
   (a) Improve the upper bounds on the length of the shortest synchronising words for Huffman codes.
   (b) Establish the computational complexity of finding a shortest synchronising word for a Huffman code.

7. Decompositions of languages:
   (a) Establish the decidability status of checking whether a regular language is prime under intersection.
   (b) Establish the existence of magic numbers for common language operations for small alphabets.

Significance

It is easier to explain the motivation behind some of the proposed topics by giving a broader picture, in particular by describing the research performed on word equations, with particular focus on recent advances obtained by the applicant using the recompression-based approach.

Recompression: Word equations and beyond

Classic decidability results A word equation consists of two words (u, v), often written as $u = v$, where both u and v contain letters from some alphabet Σ (which is a part of the input) as well as variables from a set $\mathcal{X}$. We are to decide, whether there is a substitution for variables $S : \mathcal{X} \mapsto \Sigma^*$ that turns this formal equation into a true equality of words (also called strings). This problem was first investigated in the context of Hilbert's tenth problem, as it is easy to reduce word equations to it. Thus a possible way to show undecidability of Hilbert's tenth problem was to show the undecidability of word equations. Alas, while Hilbert's tenth problem was eventually shown to be undecidable, word equations turned out to be decidable, as shown by Makanin [59]. His proof was involved in the means of computational complexity, ideas and length. For over 20 years many simplifications of the proof and improvements of the running time were made [29, 83, 49, 25]; however, the essential idea, complexity of the proof and high computational complexity prevailed.

A breakthrough was done by Plandowski and Rytter, who for the first time employed compression in word equations [71]: they showed that the smallest solutions (of size N) of word equation (of size n) has compressed representation whose size is at most polynomial in N and $\log N$, thus the algorithm can guess the compressed representation for each variable and verify the equality of both sides of the equation, which can be done using a suitable algorithm [67] (devised earlier by Plandowski).

It is worth mentioning, that an approach in which we compress a solution (or in the non-deterministic case—we guess the compressed representation of the solution) and then perform the computation directly on the representations using known algorithm that work in polynomial time, turned out to be extremely fruitful in many branches of computer science, like verification, string algorithms, computational topology, computation in groups, etc. The recent survey by Lohrey gives several such successful applications [57].

The next step was done by Plandowski, who showed a doubly exponential bound on N, the proof was involved and included novel factorisation of words [68]. Finally, by better exploiting the interplay between the factorisations and compression, a PSPACE algorithm was obtained [69]. A generalisation of this algorithm can be also used to generate a finite graph representation of all solutions of a word equation [70], no such result was previously known.

The problem of word equations was considered in both generalised and restricted scenarios. The most popular generalisation comes from allowing regular constraints: we require that the substitution for the variables come from some regular languages (whose description is part of the input). On one hand this restriction looks natural from applications point of view and moreover it is important also for the theory of computer science: word equations over groups reduce to the case of word equations with regular constraints and inversion [12] (the reduction is completely syntactic, so any algorithm for word equations can be used on top). Both mentioned algorithms for word equations can accommodate regular constraints [83, 12] and Plandowski's algorithm also works when involution is allowed [12]. However, due to internal mechanics, the description of all solution in this case cannot be obtained (it can in the case of regular constraints [70]).

On the other hand, the scenarios in which we restrict the number of different variables or number of occurrences of variables were investigated:

- for one variable an algorithm with an almost linear-time ($\mathcal{O}(n + \#_X \log n)$, where $\#_X$ is the number of occurrences of X in the equation) was known [16];
- for two variables an $\mathcal{O}(n^5)$ algorithm was known [15];
- the quadratic equation, i.e. such that each variable occurs at most twice, are only known to be in PSPACE, but this can be shown by a much simpler argument and was known before Plandowski's algorithm. It is conjectures, that this problem is in NP, so the same as the similar problem for groups.

For word equations the recompression method yielded a liner-time algorithm (improving the previous $\mathcal{O}(n + \#_X \log n)$ one, where $\#_X$ is the number of occurrences of X in the equation) [37]. Note that the analysis of the algorithm is far from easy.

Recompression approach Recently, the applicant proposed a simplified solution to word equations [32]. It is also based on compression: it applies two simple compression rules (replacing pairs ab by a fresh letter and replacement of maximal blocks a^ℓ by a fresh letter) directly on the equation and modifying the equation (replacing a variable X by a aX or Xa) so that such replacement schemes can be applied to the instance. The crucial property, is that the algorithm keeps the instance small: the modification of the variables enlarge the size of the instance (proportionally to the number of occurrences of variables, which does not increase over the run of the algorithm), but the compression removes a constant fraction of letters from the instance, and so the instance remains polynomial.

In this way a much simpler algorithm for word equations, with the same bound of PSPACE, was obtained. Similarly to Plandowski's algorithm, it can be also used to generate a finite representation of all solutions.

As the proposed algorithm is simpler, there was a reasonable hope that it could be generalised. Indeed, it was generalised to the case that includes the regular constraints and involution, for which it also yielded a finite representation of all solutions [13], which is a first result of this kind; the same applies also to the case of equations over free groups and regular constraints [13].

Quite surprisingly, the algorithm applies also to the restricted variants of word equations: a recompression based algorithm in the case of one variable turn out to be deterministic and can be sped up so that it runs in linear time [37]. Note that the previous approaches used tailored methods that had little to do with algorithms for the general case.

Other applications of recompression The operations performed on the word equation (replacing pairs and blocks) naturally correspond to rules of a grammar and in fact in the end we can represent a solution of a word equation as a grammar generating exactly one string. This corresponds to a broader concept of *grammar compression*, in which we represent a word as a grammar generating exactly one word, for historical reasons such grammars are called *SLP*s (*Straight Line Programs*).

Due to ever increasing amount of data, recently there is a growing trend in the algorithmic community to develop algorithms that work directly on the compressed representations, without the need of prior decompression. The natural inductive structure of SLPs makes them particularly suitable for such processing, moreover, it is known that block-compression methods (LZ77, LZ78, LZW) can be naturally transformed into SLP and in fact many algorithms that work on such compressed representations as a first step transform them into SLPs.

In a very general scenario, SLPs can be seen as word equations of a very simple form (a production $X \to YZ$ corresponds to an equation $X = YZ$) and so in essence recompression should work for SLPs. The unexpected property of the technique is that it proved to be very efficient in case of SLPs, outperforming many tailored methods for various problems: equality testing and pattern matching for SLPs [30], recognition by compressed automata [34], construction of the grammar for a string [31, 36]. Thus it proved to be a general and powerful tool for dealing with implicit representation of data.

Context unification

Terms are as natural as words in computer science and indeed words can be seen as very simple terms. It is natural to extend the word equations and the algorithms for them also to the term case. (Note that for historical reasons, for terms we usually talk about *unification* and not *equations*.) It is known, that term unification can be decided in polynomial time, assuming that variables represent ground terms [72] and it can be easily seen that this problem does not generalise word equations (as word variables do not represent ground terms).

Second-order unification is a natural generalisation of term unification and word equations: we allow variables to represent functions that take arguments (which are closed terms). However, it is known that this problem is undecidable, even in many restricted subcases [24, 17, 51, 54]. *Context unification* [8, 9, 76] is a natural problem 'in between': we allow variables representing functions, but we insist that they use their argument *exactly once*. It is easy to show that such defined problem generalises word equations, on the other hand, the undecidability proofs for second-order unification do not translate directly to this model.

The sole reasons of being a 'natural generalisation' of word equations does not explain the interest in the problem: context unification naturally appears in other areas of computer science: analysis of natural language [65, 65], distributive unification [77], bi-rewriting systems [52], equality up to constraints [64], linear second-order unification [55, 51], one-step term rewriting [64], bounded second order unification [79], ...

Before processing further, let us state the model a little more formal. A *ground context* is a term over a signature $\Sigma \cup \{\Omega\}$, in which the special symbol Ω (representing a missing argument) is used exactly once. A ground context c can be applied on a term t (another ground context c'), which is denoted as $c(t)$ ($c(c')$, respectively), and the result is a term obtained by substituting t for the special symbol Ω (ground context obtained by substituting c' for the Ω in c). Now, a context term is built from elements of the signature Σ, context variables, which have arity 1 and represent ground contexts (so in particular they take an argument), and variables, which have arity 0 and represent ground terms (and so do not take arguments). In context unification we are given an equality of two such context terms and ask about the existence of a solutions of this equation, where a solution substitutes each context variable with a ground context and each variable with a ground term, so that the equality holds.

It is easy to see that when we consider context unification over signature that contains only unary symbols (and a constant), this is equivalent to word equations, thus indeed context unification generalises them.

Unfortunately, for over two decades the question of decidability of context unification remained open. Despite intensive research, not much is known about the decidability of this problem: only results for some restricted subcases are known [9, 78, 53, 51, 82, 81, 56, 20].

Recently, the applicant extended his algorithm for the satisfiability of word equations to context unification [35]. The extension turned out to be simple: one more compression rule (devised uniquely for trees) was needed: leaf compression, which for a node labelled with f and a child (being a leaf) labelled with c deletes c and replaces the label with a fresh letter f'.

The applicant believes that this result is remarkable, however, it should be rather seen as a first than a last step: there are several problems for context unification that still remained open and chances are that they can be solved using a similar approach.

One variable As in the case of word equations, a natural restriction of context unification are the instances with limited number of variables. As unification without context variables is simple and in general the (term) variables can be treated more easily than the context ones, usually only bounds on the number of context variables are imposed.

The context unification with only one context variable is known to be in NP [20] but it remains open, whether it is NP-hard or maybe even in P.

The additional understanding gained using recompression gives hope for a P algorithm, and if this were to fail, perhaps would allow narrowing down the sources of the problems hardness and thus help in devising the NP-hardness proof.

Two variables The case of context unification with only two context variables were shown to be decidable earlier [82], but the complexity bound was superseded by the algorithm using the recompression-based approach, which yielded a PSPACE upper bound. The applicant hopes to achieve a more efficient algorithm in this case. As already the case of one context variable is only known to be in NP, this class seems like a good candidate.

Regular constraints: linear context unification Adding regular constraints to context unification is natural and also has a theory-oriented reason: the resulting problem is known to be equivalent to linear-second order unification [55].

However, the recompression based approach for word equations requires that we restrict the signature only to the letters that are present in the considered equation. Such an assumption cannot be made in case of word equations with regular constraints, as such trimming can make the instance unsatisfiable. This can be walked around [13], but requires some technical effort.

Unfortunately, in the context unification problem the exact signature is crucial and already the algorithm for deciding the satisfiability of context unification involves some tailored trimming of signature. This makes further trimming problematic and it remains an open question, whether it can be done.

Description of all solutions of a context equation Description of all solutions of a context unification is desirable, but as in the case of regular constraints, the trimming of signature used by the recompression based algorithm is a problem, as in order to describe the set of solutions, it seems that no trimming should be done at all. In case of word equations the problem was walked around using a homomorphism-based classification of solutions and it was shown that it is enough to consider minimal solutions in this classification [70, 32]. It seems unlikely that such an approach works also in case of context unification. On the other hand, the recent variant of recompression-based algorithm uses a technique of decompressing the symbols that are not present in the equations [13]. Perhaps a similar technique can be used also in the case of context unification.

One-step term rewriting

Term rewriting stems from mathematical logic and are a versatile tool of computer science. Given some rewriting system we would like not only to use it, but also to reason about it. Theory of one-step term rewriting was investigated, as on one hand it is relatively simple and on the other many interesting properties of rewriting systems can be expressed using it. Initial hope was that the whole first-order theory is decidable, as important properties which are decidable for arbitrary *finite* rewrite systems can be expressed in this logic, while undecidable properties of rewrite systems seemed not to be expressible in it. However, this was shown to be not the case: this theory is undecidable [84], even in very restricted cases [87, 60].

More formally, a rewrite R consists of a finite set of rewriting rules $\{r_i \to r_i'\}_{i=1}^{k}$, where r_i and r_i' and terms built using function symbols and variables (which represent full terms). The rewriting looks as follows: the term t can be rewritten to t', if $t = C(r)$ and $s = C(r')$, where C is a ground context and (r, r') is an instance of some of the pairs (r_i, r_i'), in the sense that there is a substitution for variables used in (r_i, r_i') that turns (r_i, r_i') to (r, r'). The formula of theory of one step term-rewriting (note that there is such a theory for each rewriting system) uses atomic predicates of the form $x \to y$, where x and y are variables, note that no function symbols nor equality is allowed.

As the whole first order theory was shown to be undecidable, it was investigated, whether at least some fragments of the logic are decidable. As of today, it remains open, whether the positive fragment (so no negation) or existential fragment (so only a prefix of existential quantifiers is allowed) are decidable. On the other hand, it was shown that the positive existential fragment is decidable: Already the definition of atomic formula in the theory given above readily gives a reduction from this fragment to the context unification. Moreover, the target instances are of quite restrictive form and in fact they always instances of the so called ‘stratified context unification’, which was shown to be decidable [78] in NP [53].

Still, the decidability status of existential theory and positive theory remain unknown. There is some hope that recompression based methods can be applied also in this setting: firstly, the known decidability proof and the definition clearly link one-step term rewriting to context unification. Secondly, that grammar compression was instrumental in obtaining the exact computational complexity of stratified context unification [53]. Lastly, the applicant hopes that the regular constraints (assuming that they indeed we can solve instances involving them) added to context unification can be helpful here.

Algorithms on compressed trees

While the recompression technique was developed for problems somehow related to strings, the algorithm for context unification already demonstrated that it can be used for problems with some inherent tree structure. (Note that the trees we consider are rooted, ordered and labelled, moreover, to each label f we assign arity $\mathrm{ar}(f)$ and we want nodes labelled by f to have $\mathrm{ar}(f)$ children; thus, they can be equally seen as terms.)

Algorithms on compressed trees Much of the data appearing in computer science has a natural tree (or term) structure. This structure can be disregarded during compression, but this may worsen the compression ratio and moreover makes further processing of the data (especially in the compressed form) much harder. Instead, there are models of compression which explicitly take the tree structure into the account, in particular, the grammar compression is generalised from strings to trees [4]. Those grammars preserve the main property of the grammar compression for strings: the produced compressed representation still has a nice inductive structure and it seems that it can be naturally processed.

This field seems to be much less understood then its string counterpart: For instance, only recently a first algorithm with a guaranteed approximation ratio was proposed [38], and it was constructed by the applicant (and M. Lohrey) using the recompression technique.

On the other hand, compression-based techniques were already successfully applied in term-related research [20, 53], which gives reasonable hopes that more can be done more using such techniques.

Model Although grammar for strings are already folklore, their trees counterparts are not as popular and for this reason the appropriate definition is supplied here. Just as in case of strings the nonterminals represent fragments of strings, i.e. strings as well, in case of trees the nonterminals should represent fragments of trees, i.e. also trees. However, when we insist that those trees are well-formed terms then we immediately conclude that our grammar are in fact equivalent to dags, which seem to be much less expressible than SLPs. Thus our nonterminals should represent fragments that can be plugged into other fragments, which is formalised using arities and parameters: each nonterminal A has an *arity* $\mathrm{ar}(A)$, which intuitively means that it represents a tree with $\mathrm{ar}(A)$ missing arguments. A rule for such a nonterminal is of the form $A(y_1, \ldots, y_{\mathrm{ar}(A)}) \to t$, where $y_1, \ldots, y_{\mathrm{ar}(A)}$ are *parameters* (that are letters from outside the signature) that can be used in t. Then a rule applied on a tree $A(t_1, \ldots, t_k)$, where $t_1, \ldots, t_k$ are terms, is rewritten as $t[t_1, \ldots, t_k]$, which denotes t in which the parameter y_i is replaced with t_i, for each i. (Note that in this convention the usual (string) grammars have all nonterminals of arity 1, as they represent words to which somethign can be always concatenated to the right). Since we consider tree grammars as a mean of compression, we are only interested in grammars whose nonterminals define a unique tree.

There are two variants of tree grammars: *linear* and *non-linear*. In the former one we assume that in the rule $A(y_1, \ldots, y_{\mathrm{ar}(A)}) \to t$ in t each parameter y_i is used at most once, in non-linear ones we do not make such an assumption. This distinction is clear when we look at grammars as a rewriting system: the linear grammars locally rewrite A with some term, without changing other parts of the term. The non-linear grammars can copy subterms, which can be also seen as locally rewriting dags: in such a case copying a subterm boils down to directing an edge at the same node.

Linear tree grammars Linear tree grammars are much simpler and it seems that they correspond more closely to SLPs, for instance, their equality can be checked in polynomial time [4], a generated tree is at most exponentially larger than the grammar, one can normalise the grammars so that they used only one parameter [58] (and the size increase is only polynomial), etc.

One of the inspirations for the recompression was the work of Mehlhorn et al., who proposed an efficient structure for checking the equality of strings under the operations of concatenation and taking subwords [62] (note that such a model easily encompasses SLPs). It seems interesting to see, whether this data structure can be also extended to trees, so that it still captures the linear tree grammars? The applicant believes that his experience and understanding will allow an appropriate definition of such an extension as well as an efficient implementation of it.

Non-Linear tree grammars In contrast to linear tree grammars, the non-linear variant seem to be much more powerful: the generated tree can be doubly exponential, parameter reduction can cause exponential blow-up [58], etc. In particular, many basic decision problem have unknown computational complexity, most notably, checking the equivalence of two non-linear tree grammars is between P and PSPACE [4]. It seems unlikely that this problem is in P, but still believable that it is in NP. The applicant would like to investigate this problem in greater detail: it seems that the non-linearity and copying of arguments is a challenge but hopefully the recompression approach can be used also in the non-linear case.

Submatching for compressed trees Research in term rewriting was not left untouched by the compression revolution. One of the fundamental questions in the area is the submatching problem: we are given two terms s and t, where s may use variables while t is a ground term and ask, whether there is an instance of s in t; such an instance is a subterm of t which is equal to s with some substitution for its variables. This question is naturally motivated: we can view s as a left-hand side of a rewriting rule and ask, whether this rule can be applied to t (and in what places).

A natural subvariant of this problem is when s is linear, that is, each variable in it is used at most once. For such a variant the submatching problems for s and t given by (linear) tree grammars is known to be in P [80]. On the other hand, the computational complexity of a similar problem when s is non-linear remains unknown (one can easily see that it is in NP, though). The applicant wants to investigate this problem. At a first sight the non-linearity of s may be a problem, on the other hand the context unification is in some sense non-linear and t is given by a linear grammar, which gives some hope for a positive solution.

Word equations revisited

As already observed, the recompression method reshaped the landscape of word equations and surprisingly, it showed to be effective both in the general case and very restricted ones (one-variable). Thus there is a hope that one can employ such an approach to solve other problems for word equations. The most important one is to establish the exact computational complexity of the problem (the usual conjecture is that they should be in NP), however, this seems as too ambitious a goal.

Still, there are some restricted subclasses, for which progress halted for some time.

Two variables The best known algorithm generating a finite representation of all solutions of a word equation with 2 variables runs in $\mathcal{O}(n^5)$ time and is based solely on string combinatorics [15]: it identifies a couple of classes of potential solutions and then tests solutions from those classes one by one.

It seems plausible that employing compression can be used to speed up the computation: the considered classes of solutions are all well-compressible (they are usually of a form $(u^k w)^\ell$ for some strings u, w and natural numbers k, ℓ) and the recompression based algorithm for comparing such representations [30] is much more efficient than previously known.

Quadratic equations In the quadratic equations we are considering only instances in which each variable occurs at most twice (though they can have arbitrary many occurrences). It is substantially easier to give a PSPACE algorithm for such instances [11] and it is widely believed that this case should be in NP (so the same as the general case, but here the belief is stronger and the solution seems more reachable than in the general case). However, no progress in this subcase was obtained for several years. On the other hand, it is known that the corresponding problem in group case is in NP [48], however, the argument is very geometric and it is unlikely that it can be transferred to the string case. Still, there is hope that compression based approach can help in solving this special case.

Free inverse monoid

The interplay between algebra and computer science has many faces. On one side, developing algorithms for algebraic questions brings more new ideas into the area; on the other, exploiting algebraic techniques yielded new algorithms in many unexpected places; lastly, many phenomena in computer science can be modelled using some algebraic constructs and their structure can be leveraged in order to obtain better algorithms.

Following the third approach, semigroups are often used in computer science. If one gets really lucky, even groups can be used as a model. Inverse monoids are an intermediate class between general semigroups and groups: in *inverse monoids* every element x has an *inverse* x^{-1}, which satisfies the following relations: $xx^{-1}x = x$ and $x^{-1}xx^{-1} = x^{-1}$. There were many expectations from this model, as on one hand it seems to have more structure than semigroups and on the other more expressible than groups.

Just as the groups can be represented by permutations, inverse monoids can be represented by partial injections (and on the other hand, the monoid of partial injections is an inverse monoid). Similarly to free semigroups and free groups, also inverse monoids have an universal structure, called *free inverse monoid*, whose approachable construction is due to Scheiblich [75], a similar definition was given also by Munn [63]. In this construction elements of the monoid are finite subtrees of the Cayley graph of a free group, they are always rooted in e and have a distinguished vertex a. Multiplication of two such trees T_a and T_b is defined as a union of T_a and T_b rerooted to a and the new distinguished vertex is ab. More formally, elements are pairs (a, T_a), where a is an element of the free group and T_a is a finite subtree of the Caylay graph, such that $a, e \in T_a$. The multiplication is defined by $(a, T_a)(b, T_b) = (ab, T_a \cup aT_b)$ where aT_b is the T_b translated by a.

Alas, free inverse monoids turned out seem to be essentially more difficult than free groups and free semigroups: while solving equations in the two former one is decidable, it is undecidable in the case of free inverse monoids [73]. This naturally leads to investigations of restricted subclasses of equations as well as sine simplified variants of equations' solving.

One variable The most natural restriction of equations, in view of research performed for word equations, is limiting the number of variables. Unfortunately, so far even the problem of the decidability of equations in free inverse monoids with one variable remains open. The applicant would like to investigate this topic: it looks plausible that the new methods for solving one variable word equations [37] could be also employed also in this more general scenario. Moreover, there are hints [14] that solving language equations is also needed when solving the equations over free inverse monoids and applicant's earlier research topics focused on language equations, thus he has experience in this subject as well.

Extendibility problem On the other hand, the other natural problem is the *extendibility*: given an equation $u = v$ in the free inverse monoid we can also consider its projection onto the corresponding free group: using the Scheiblich [75] construction each constant is of a form (a, T_a) and we restrict it to a. From the same construction it is clear that if the equation in free inverse monoid is satisfiable, so is the equation obtained as a projection onto the free group, but not necessarily the other way around and one can easily give examples that this is not the case. The extendibility problem is a strict formulation of the latter condition: given an equation $u = v$ in the free inverse monoid and a solution of the corresponding projection onto the free group, can this solution be extended to a solution in the free inverse monoid? This problem was first solved by Dies, Meakin and Sénizergues [10], but the computational complexity of their solution is very high, mostly because it employs Rabin's Tree Theorem.

Recently, the computational complexity of this problem was lowered to DEXPTIME [14], which was partially possible by using a DEXPTIME algorithm for solving language equations (of a specific form) [1]. The latter problem is in fact DEXPTIME-hard [1], however, the equations that are obtained in [14] are not expressible enough to directly accommodate the known DEXPTIME-hardness proof. Thus, it remains open, whether this problem is indeed DEXPTIME-hard or perhaps its computational complexity is lower. The applicant would like to investigate this problem: on one hand, so far the used method ignored the corresponding word equation and focused mostly on sets, maybe some insight into the projected group equation could help? On the other hand, the applicant has also some previous experience with showing lower bounds for (seemingly simple) language equations [42, 43, 41] and so believes that he is knowledgeable in the topic.

Synchronisation of prefix codes

While applicant's research focused mostly on application and properties of grammar compression, this projects should also help to extend the scope of this research to other compression methods. On one hand grammar compression potentially offers very high compression ratio, on the other its main drawback is the high comutation time which may cause the whole method to be too slow in several applications. One of the fastest alternatives are variable-length codes, which can be quickly calculated and easily processed: we partition the input string into blocks of the same length and then uniformly replace the blocks with varying-lengths representations $\{w_1, \ldots, w_N\}$. To avoid ambiguity, the code-words $\{w_1, \ldots, w_N\}$ should be *prefix-free* in the sense that none of its word is a prefix of the other; in this way the encoded text can be greedily decoded. Such codes are known as *Huffman codes* and there is a standard way of calculating them for a given string. Moreover, one can easily see [3] that a prefix code corresponds naturally to a DFA called the *decoder*, whose states are proper prefixes of words from this code.

One of the problems with compressed data is reliability in case of presence of errors in the compressed text. Eventually, a single error may possibly destroy the whole encoded string. One of the proposed solutions to this problem (for Huffman codes) are codes that can be synchronised, regardless of the possible errors: a code is *synchronising* if there exists a word (called a *synchronising word* for this code), which guarantees that the subsequent text will be decoded correctly, regardless of errors occurring before it. In technical terms, a word w is synchronising if for any u_1, u_2, v the partitions of u_1wv and u_2wv into code-words exist and they coincide on v. This property is not only useful in case of transmission errors, but helps in general task of data recovery. Most binary Huffman codes are synchronising [18] and in fact by adding some redundancy we can turn any Huffman code into a synchronising one [5].

Biskup and Plandowski [3] tried to upper-bound the maximal lengths of synchronising words of binary Huffman codes. They gave an $\mathcal{O}(Nh \log N)$ bound for the length of the shortest synchronising words, where h is the maximum length of a code-word in the code. Since h can be linear in N, the bound in terms of N alone is $\mathcal{O}(N^2 \log N)$. Their bound is constructive in the sense that they provided a polynomial algorithm that, given a Huffman code, computes its synchronising word of such length. They also showed classes of codes whose shortest synchronizing words are of length $\Theta(N)$, and conjectured that this is the worst case, i.e. that a synchronising word of length $\mathcal{O}(N)$ always exists. As a side note, all their considerations applied only to binary codes.

Using automata-theoretic approach, Béal et al. [2] improved (and generalised) the upper bound to $\mathcal{O}(N^2/k)$, where k is the size of the alphabet (in particular, this work covers also non-binary alphabets). Unfortunately, this bound is non-constructive: only existence of such a word was shown, no algorithm for its construction was provided.

The difference between lower and upper bounds is huge and the applicant would like to improve the known upper bounds, in particular, apply them to an alphabet of an arbitrary size; in the best case the bounds should be constructive.

A natural related questions is the *exact length* of the shortest synchronising words for a Huffman code. The computational complexity of this problem is unknown, and for a related (more general) question about the length of the shortest synchronising words for a DFA, completeness for $\mathsf{FP}^{\mathsf{NP}[\log]}$ was shown [66].

Decomposition of formal languages

Prime languages When considering a formal language and not a single string, another way of lowering the size of the description of this language (so, in some sense, a compression) is some sort of a *factorisation*, according to some language operation $*$. First attempts of this kind were done for concatenation, i.e. a language L was represented as a concatenation of other languages $L_1 \cdot L_2 \cdots L_k$. This problem has been extensively studied by Salomaa et al. [27, 26, 61, 74] and in particular they introduced a notion of a *prime* language, i.e. one for which only trivial factorisations exist (a factorisation is trivial if one of $L_1, L_2, \ldots, L_k$ is equal to L and all other to $\{\epsilon\}$).

Recently similar notions were reinvestigated with another application in mind: parallelisation of the computation [50]. The underlying idea is that the language representation can be large but it can be distributed in parallel into smaller representations. More formally, for a given language L and operation $*$ (say, binary, for ease of presentation) we are interested in representing L as $L_1 * L_2 * \cdots * L_k$, where size of the representation of each L_i is strictly smaller than the size of L (in our case we are interested only in regular languages and the size is the number of states of a minimal DFA recognising the language).

As an example, consider the automata-theoretic approach to model-checking introduced by Vardi and Wolper [86]. One can transform a formula into a corresponding generalized Büchi automaton and if the formula si representable as conjunction of several subformulas, the language of this automaton is a finite intersection of simpler languages, whose model checking can be done in parallel.

To stress the connection to the older idea of factorisations under concatenation, we say that a language is *prime* (with respect to this particular operation $*$), if it can be only trivially factorised (with respect to $*$).

Verifying whether a language is prime seems to be challenging: already for intersection (and regular languages) its decidability status remains unknown. The applicant would like to investigate this and similar questions.

Magic numbers This problem is also investigated from a different angle in the field of descriptive complexity (recall that the *state complexity* of a language is the number of states in a minimal DFA recognising it): Can a regular language with state complexity n be the result of a k-ary operation $*$ on languages with state complexities $n_1, \ldots, n_k$? Given complexities $n_1, \ldots, n_k$, all possible resulted complexities n lie within some range $[\alpha, \beta]$ of natural numbers. The range of possible outcomes is a subject of recent extensive research in the field of descriptive complexity (for the current state of art, see [19]). If this range is not continuous, a number $n \in [\alpha, \beta]$ that is not a possible outcome is called a *magic number* (for $*$).

Magic numbers arose for the first time from determinisation of NFAs: does there exists a minimal NFA with m states, whose equivalent smallest DFA has n states, for all n and $m \leq n \leq 2^m$ [28]? The problem was systematically studied, and recently it was shown that there are no magic numbers for regular languages on a ternary alphabet [45]. On the other hand, there are magic numbers in case of unary alphabets [23]. The question for binary alphabets remains open, although some partial results showing non-existence of magic numbers in certain intervals were obtained.

Recently, the problem of magic numbers was also considered for other common $*$ operations: Kleene star [6], concatenation [44], square [7], reversal [47]. So far, existence of magic numbers was shown only for a few particular settings, e.g. determinisation of a unary NFA [23], determinisation of a symmetric difference of unary NFAs [85], and cyclic shift of languages over binary and ternary alphabets [46]. In general, for common operations (e.g. concatenation, reversal, and Kleene star) and fixed size of the alphabet the problem turned out to be difficult and the applicant wants to investigate it in more detail.

Work plan

The research problems are rather parallel and there is little dependency between them; however, the suspected ones are going to be listed below.

Firstly, the applicant wants to investigate the problems related to context unification, hopefully extending the known algorithm to the case with regular constraints (perhaps using the new technique of decompressing the non-existing letters [13]) as well the description of all solutions of the context equation. Initial investigations by the applicant suggest that those are needed in the analysis of existential theory of one-step term rewriting and hopefully the case of positive follows.

Applicants intuition is that also similar techniques are needed for the non-linear submatching problem for compressed trees, though perhaps even the existing methods used in context unification are enough (as they already deal with non-linearity).

Other research topics can be performed in parallel.

For the context unification with two variables [82], the known approach to this problem does not utilise at all the compression techniques and it seems almost sure that employing them will reduce the complexity of the problem. The subproblem with one variable used this type of techniques [20], but the results for one-variable word equations [37] suggest that those can be done more efficiently.

Similarly, the known solution for word equations with two variables [15] does not employ compression techniques and the speed up in the case of one variable gives hope for a similar speed-up in this case, especially that the solutions are known to be well-compressible. The case with quadratic equations does not have such preliminary results, but the applicant still hopes that the upper bound can be improved.

For research on free inverse monoids, the problem seems to combine the language equations [14] and word equations. Since a lot of understanding was gained in both, solving equations in one variable seems within reach. Concerning the extendibility problem, the new upper bound narrows down the search for appropriate lower bound.

For data structure for checking the equality of dynamic trees, the original data structure used similar operations as the replacement of blocks and pairs from recompression for strings. From recent work on approximation of tree compression [38] we already know that adding the operation of leaf compression to the tree case should suffice. In order to combine the trees one should probably consider splitting the tree (into two) at a specific node.

Concerning the non-linear grammars, the applicant so far has no clear intuition, but still hopes for a positive outcome.

For the synchronisation of the prefix codes, it seems that the automata-theory approach [2] is more suitable. The applicant hopes that firstly it can be made constructive and secondly, improved, as so far it seems that the upper bound is not tight. Moreover, a detailed study of lower and upper bounds for computing the length of the shortest synchronising words [66] should at least yield some intuition for the case of synchronisation of the Huffman codes.

Concerning the problems related to representations of DFAs, not much is known for the decomposition by intersection and the applicant hopes to change it. The same applies also to the search of magic numbers for language operations.

Methodology

A traditional mathematical-proof methodology applied to computer science is going to be used, to be more precise:

Worst-case performance The algorithms are judged by the worst-case performance. This applies to running time, space usage, approximation guarantees, etc. Thus, the obtained bounds are universal.

Proofs of claims The claimed properties (of algorithms, devices, etc.) are shown using proofs.

Experiments Though performing tests on experimental data is planned, such experiments will only provide heuristical results which may help in devising proofs but are not meant to replace them.

How proofs are done By 'proof' we mean a standard mathematical proof.

Literature references

[1] Franz Baader and Paliath Narendran. Unification of concept terms in description logics. *Journal of Symbolic Computation*, 31:277–305, 2001.

[2] M.-P. Béal, M. V. Berlinkov, and D. Perrin. A quadratic upper bound on the size of a synchronizing word in one-cluster automata. *International Journal of Foundations of Computer Science*, 22(2):277–288, 2011.

[3] M. T. Biskup and W. Plandowski. Shortest synchronizing strings for Huffman codes. *Theoretical Computer Science*, 410(38–40):3925–3941, 2009.

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

[5] R. M. Capocelli, A. A. De Santis, L. Gargano, and U. Vaccaro. On the construction of statistically synchronizable codes. *IEEE Transactions on Information Theory*, 38(2):407–414, 1992.

[6] K. Čevorová. Kleene Star on Unary Regular Languages. In *DCFS*, volume 8031 of *LNCS*, pages 277–288. Springer, 2013.

[7] K. Čevorová, G. Jirásková, and I. Krajňáková. On the Square of Regular Languages. In *CIAA*, volume 8587 of *LNCS*, pages 136–147. Springer, 2014.

[8] Hubert Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *Journal of Symbolic Computation*, 25(4):397–419, 1998.

[9] Hubert Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *Journal of Symbolic Computation*, 25(4):421–453, 1998.

[10] Timothy Deis, John C. Meakin, and Géraud Sénizergues. Equations in free inverse monoids. *International Journal of Algebra and Computation*, 17:761–795, 2007.

[11] Volker Diekert. Makanin's algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12, pages 342–390. Cambridge University Press, 2002.

[12] Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Inforamtion and Computation*, 202(2):105–140, 2005.

[13] Volker Diekert, Artur Jeż, and Wojciech Plandowski. Finding all solutions of equations in free groups and monoids with involution. In Edward A. Hirsch, Sergei O. Kuznetsov, Jean-Éric Pin, and Nikolay K. Vereshchagin, editors, *CSR*, volume 8476 of *LNCS*, pages 1–15. Springer, 2014.

[14] Volker Diekert, Florent Martin, Géraud Sénizergues, and Pedro Ventura Alves da Silva. Equations over free inverse monoids revisited. unpublished manunscript, 2014.

[15] Robert Dąbrowski and Wojciech Plandowski. Solving two-variable word equations. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP*, volume 3142 of *LNCS*, pages 408–419. Springer, 2004.

[16] Robert Dąbrowski and Wojciech Plandowski. On word equations in one variable. *Algorithmica*, 60(4):819–828, 2011.

[17] William M. Farmer. Simple second-order languages for which unification is undecidable. *Theoretical Computer Science*, 87(1):25–41, 1991.

[18] C. F. Freiling, D. S. Jungreis, F. Theberge, and K. Zeger. Almost all complete binary prefix codes have a self-synchronizing string. *IEEE Transactions on Information Theory*, 49(9):2219–2225, 2003.

[19] Y. Gao, N. Moreira, R. Reis, and S. Yu. A Review on State Complexity of Individual Operations. Technical Report DCC-2011-08, DCC - FC, University of Porto, 2012. `www.dcc.fc.up.pt/dcc/Pubs/`.

[20] Adria Gascón, Guillem Godoy, Manfred Schmidt-Schauß, and Ashish Tiwari. Context unification with one context variable. *Journal of Symbolic Computation*, 45(2):173–193, 2010.

[21] Pawel Gawrychowski and Artur Jeż. Hyper-minimisation made efficient. In Rastislav Královic and Damian Niwiński, editors, *MFCS*, volume 5734 of *LNCS*, pages 356–368. Springer, 2009.

[22] Paweł Gawrychowski, Artur Jeż, and Andreas Maletti. On minimising automata with errors. In Filip Murlak and Piotr Sankowski, editors, *MFCS*, volume 6907 of *LNCS*, pages 327–338. Springer, 2011.

[23] V. Geffert. Magic numbers in the state hierarchy of finite automata. *Information and Computation*, 205(11):1652–1670, 2007.

[24] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[25] Claudio Gutiérrez. Satisfiability of word equations with constants is in exponential space. In *FOCS*, pages 112–119. IEEE Computer Society, 1998.

[26] Yo-Sub Han, Arto Salomaa, Kai Salomaa, Derick Wood, and Sheng Yu. On the existence of prime decompositions. *Theoretical Computer Science*, 376(1-2):60–69, 2007.

[27] Yo-Sub Han, Kai Salomaa, and Derick Wood. Prime decompositions of regular languages. In Oscar H. Ibarra and Zhe Dang, editors, *DLT*, volume 4036 of *LNCS*, pages 145–155. Springer, 2006.

[28] K. Iwama, Y. Kambayashi, and K. Takaki. Tight bounds on the number of states of DFAs that are equivalent to n-state NFAs. *Theoretical Computer Science*, 237(1–2):485–494, 2000.

[29] Joxan Jaffar. Minimal and complete word unification. *Journal of the ACM*, 37(1):47–85, 1990.

[30] Artur Jeż. Faster fully compressed pattern matching by recompression. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *ICALP (1)*, volume 7391 of *LNCS*, pages 533–544. Springer, 2012.

[31] Artur Jeż. Approximation of grammar-based compression via recompression. In Johannes Fischer and Peter Sanders, editors, *CPM*, volume 7922 of *LNCS*, pages 165–176. Springer, 2013.

[32] Artur Jeż. Recompression: a simple and powerful technique for word equations. In Natacha Portier and Thomas Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 233–244, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[33] Artur Jeż. Recompression: Word equations and beyond. In Marie-Pierre Béal and Olivier Carton, editors, *DLT*, volume 7907 of *LNCS*, pages 12–26. Springer, 2013.

[34] Artur Jeż. The complexity of compressed membership problems for finite automata. *Theory of Computing Systems*, 55:685–718, 2014.

[35] Artur Jeż. Context unification is in PSPACE. In Elias Koutsoupias, Javier Esparza, and Pierre Fraigniaud, editors, *ICALP*, volume 8573 of *LNCS*, pages 244–255. Springer, 2014.

[36] Artur Jeż. A *really* simple approximation of smallest grammar. In Sergei Kuznetsov, Alexander Kulikov, and Pavel Pevzner, editors, *CPM*, volume 8486 of *LNCS*, pages 182–191. Springer, 2014.

[37] Artur Jeż. One-variable word equations in linear time. *Algorithmica*, 2014. accepted.

[38] Artur Jeż and Markus Lohrey. Approximation of smallest linear tree grammar. In Ernst W. Mayr and Natacha Portier, editors, *STACS 2014*, volume 25 of *LIPIcs*, pages 445–457. Schloss Dagstuhl — Leibniz-Zentrum fuer Informatik, 2014.

[39] Artur Jeż and Andreas Maletti. Computing all ℓ-cover automata fast. In Béatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud, and Denis Maurel, editors, *CIAA*, volume 6807 of *LNCS*, pages 203–214. Springer, 2011.

[40] Artur Jeż and Andreas Maletti. Hyper-minimization for deterministic tree automata. *International Journal on Foundations of Computer Science*, 24(6):815–830, 2013.

[41] Artur Jeż and Alexander Okhotin. Equations over sets of natural numbers with addition only. In Susanne Albers and Jean-Yves Marion, editors, *STACS*, volume 3 of *LIPIcs*, pages 577–588. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[42] Artur Jeż and Alexander Okhotin. Conjunctive grammars over a unary alphabet: Undecidability and unbounded growth. *Theory of Computing Systems*, 46(1):27–58, 2010.

[43] Artur Jeż and Alexander Okhotin. Computational completeness of equations over sets of natural numbers. *Information and Computation*, 237:56–94, 2014.

[44] G. Jirásková. Concatenation of Regular Languages and Descriptional Complexity. In *CSR*, volume 5675 of *LNCS*, pages 203–214. Springer, 2009.

[45] G. Jirásková. Magic numbers and ternary alphabet. In Volker Diekert and Dirk Nowotka, editors, *DLT*, volume 5583 of *LNCS*, pages 300–311. Springer, 2009.

[46] G. Jirásková and A. Okhotin. State complexity of cyclic shift. *RAIRO - Theoretical Informatics and Applications*, 42:335–360, 2008.

[47] Š. Juraj. Reversal on Regular Languages and Descriptional Complexity. In *DCFS*, volume 8031 of *LNCS*, pages 265–276. Springer, 2013.

[48] Olga Kharlampovich, I. G. Lysënok, Alexei G. Myasnikov, and Nicholas W. M. Touikan. The solvability problem for quadratic equations over free groups is NP-complete. *Theory of Computing Systems*, 47(1):250–258, 2010.

[49] Antoni Kościelski and Leszek Pacholski. Complexity of Makanin's algorithm. *Journal of the ACM*, 43(4):670–684, 1996.

[50] Orna Kupferman and Jonathan Mosheiff. Prime languages. In Krishnendu Chatterjee and Jiri Sgall, editors, *MFCS*, volume 8087 of *LNCS*, pages 607–618. Springer, 2013.

[51] Jordi Levy. Linear second-order unification. In Harald Ganzinger, editor, *RTA*, volume 1103 of *LNCS*, pages 332–346. Springer, 1996.

[52] Jordi Levy and Jaume Agustí-Cullell. Bi-rewrite systems. *Journal of Symbolic Computation*, 22(3):279–314, 1996.

[53] Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011.

[54] Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Information and Computation*, 159(1–2):125–150, 2000.

[55] Jordi Levy and Mateu Villaret. Linear second-order unification and context unification with tree-regular constraints. In Leo Bachmair, editor, *RTA*, volume 1833 of *LNCS*, pages 156–171. Springer, 2000.

[56] Jordi Levy and Mateu Villaret. Currying second-order unification problems. In Sophie Tison, editor, *RTA*, volume 2378 of *LNCS*, pages 326–339. Springer, 2002.

[57] Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

[58] Markus Lohrey, Sebastian Maneth, and Manfred Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *Journal of Computer and System Sciences*, 78(5):1651–1669, 2012.

[59] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 2(103):147–236, 1977. (in Russian).

[60] Jerzy Marcinkowski. Undecidability of the first order theory of one-step right ground rewriting. In Hubert Comon, editor, *RTA*, volume 1232 of *LNCS*, pages 241–253. Springer, 1997.

[61] Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Factorizations of languages and commutativity conditions. *Acta Cybernetica*, 15(3):339–351, 2002.

[62] Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997.

[63] Walter D. Munn. Free inverse semigroups. *Proceedings London Mathematical Society*, 29:385–404, 1974.

[64] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In William McCune, editor, *CADE*, volume 1249 of *LNCS*, pages 34–48. Springer, 1997.

[65] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. A uniform approach to underspecification and parallelism. In Philip R. Cohen and Wolfgang Wahlster, editors, *ACL*, pages 410–417. Morgan Kaufmann Publishers / ACL, 1997.

[66] J. Olschewski and M. Ummels. The complexity of finding reset words in finite automata. In *MFCS*, volume 6281 of *LNCS*, pages 568–579. Springer, 2010.

[67] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In Jan van Leeuwen, editor, *ESA*, volume 855 of *LNCS*, pages 460–470. Springer, 1994.

[68] Wojciech Plandowski. Satisfiability of word equations with constants is in NEXPTIME. In *STOC*, pages 721–725, 1999.

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

[70] Wojciech Plandowski. An efficient algorithm for solving word equations. In Jon M. Kleinberg, editor, *STOC*, pages 467–476. ACM, 2006.

[71] Wojciech Plandowski and Wojciech Rytter. Application of Lempel-Ziv encodings to the solution of word equations. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP*, volume 1443 of *LNCS*, pages 731–742. Springer, 1998.

[72] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[73] Bella V. Rozenblat. Diophantine theories of free inverse semigroups. *Siberian Mathematical Journal*, 26:860–865, 1985.

[74] Arto Salomaa and Sheng Yu. On the decomposition of finite languages. In Grzegorz Rozenberg and Wolfgang Thomas, editors, *DLT*, pages 22–31. World Scientific, 1999.

[75] H. E. Scheiblich. Free inverse semigroups. *Proceedings of the American Mathematical Society*, 38:1–7, 1973.

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

[77] Manfred Schmidt-Schauß. A decision algorithm for distributive unification. *Theoretical Computer Science*, 208(1–2):111–148, 1998.

[78] Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. *Journal of Logic and Computation*, 12(6):929–953, 2002.

[79] Manfred Schmidt-Schauß. Decidability of bounded second order unification. *Information and Computation*, 188(2):143–178, 2004.

[80] Manfred Schmidt-Schauß. Linear compressed pattern matching for polynomial rewriting (extended abstract). In Rachid Echahed and Detlef Plump, editors, *TERMGRAPH*, volume 110 of *EPTCS*, pages 29–40, 2013.

[81] Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equation. In *RTA*, volume 1379 of *LNCS*, pages 61–75. Springer, 1998.

[82] Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. *Journal of Symbolic Computation*, 33(1):77–122, 2002.

[83] Klaus U. Schulz. Makanin's algorithm for word equations—two improvements and a generalization. In Klaus U. Schulz, editor, *IWWERT*, volume 572 of *LNCS*, pages 85–150. Springer, 1990.

[84] Ralf Treinen. The first-order theory of linear one-step rewriting is undecidable. *Theoretical Computer Science*, 208(1–2):179–190, 1998.

[85] L. van Zijl. Magic Numbers for Symmetric Difference NFAs. In *CIAA*, volume 3317 of *LNCS*, pages 333–334. Springer, 2005.

[86] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[87] Sergei G. Vorobyov. $\forall\exists^*$-equational theory of context unification is Π^0_1-hard. In Lubos Brim, Jozef Gruska, and Jirí Zlatuska, editors, *MFCS*, volume 1450 of *LNCS*, pages 597–606. Springer, 1998.