Patterns in words versus patterns in trees: a brief survey and new results

Gregory Kucherov and Michaël Rusinowitch

INRIA-Lorraine/LORIA
615, rue du Jardin Botanique, BP 101
54602 Villers-lès-Nancy, France
email: {kucherov,rusi}@loria.fr

Abstract. In this paper we study some natural problems related to specifying sets of words and trees by patterns.

1 Introduction

Patterns are probably the most simple and natural way to specify non-trivial families of combinatorial structures. Abstractly, let $\mathcal{G}$ be a class of combinatorial structures with a substructure relation (such as graphs, trees, strings, etc.). Usually, given $\mathcal{G}$ we can define in a natural way a notion of pattern, interpreted as an *under-specified* structure of $\mathcal{G}$, that is a structure with some "unspecified parts". A pattern defines a set of *instances* which are structures obtained by instantiating the pattern's unspecified parts by other structures. For example, in case of graph structures, patterns could be defined as graphs with some "meta-nodes" which can be instantiated by other graphs.

Using these informal definitions, we now introduce central notions of this paper. For a set S of patterns, we denote by $Inst(S)$ the set of structures which are instances of patterns of S. By $Cont(S)$ we denote the set of structures which have a substructure in $Inst(S)$. In the above example of graphs, if S is a set of patterns (graphs with "meta-nodes"), $Inst(S)$ is the set of instances of patterns from S and $Cont(S)$ could be defined as the set of graphs having a subgraph that is an instance of a pattern of S. We will also study the complements of sets $Inst(S)$ and $Cont(S)$, defined by $\overline{Inst(S)} = \mathcal{G} \setminus Inst(S)$ and $\overline{Cont(S)} = \mathcal{G} \setminus Cont(S)$.

In this paper we consider two structures, which are probably the most widely used data structures in computer science: words and trees. We will define the notion of pattern for each of these structures and we will compare the complexity of different natural problems related to patterns in the cases of words and trees. In this perspective, we survey various known results and give several new ones.

2 Words, Trees and Patterns

Let us start with basic definitions. Given a finite alphabet A of *letters*, *words* over A are defined in the usual way as finite sequences of letters. A^* stands for the set of words over A. From algebraic point of view, words over A are elements of the free monoid generated by A. *Word patterns* over A are defined as words over alphabet $A \cup X$, where X is an infinite alphabet of *variables*. For example, $v = abaababaabaab$ is a word over the alphabet $\{a, b\}$ and assuming that $x, y \in X$, $abaxbayb, xabax, xaxxaxax$ are patterns over $\{a, b\}$. A variable occurring more than once in the pattern is called *non-linear*, otherwise it is *linear*. A *subword* of a word is a fragment of its letter sequence. For example, $baab$ is a subword of v and $abba$ is not. A *substitution* is a morphism $\sigma : (A \cup X)^* \to A^*$ such that $\sigma(a) = a$ for all $a \in A$. A substitution is *non-erasing* if $\sigma(x) \neq \varepsilon$, where ε is the empty word, and *erasing* otherwise. A word $w \in A^*$ is an *instance* of a pattern $p \in (A \cup X)^*$ if $w = \sigma(p)$ for some substitution σ. In this case we say also that p *matches* w. A substitution can be simply seen as a mapping replacing variable occurrences in the pattern by words such that the occurrences of the same variable are replaced by the same word. For example, the word v is an instance of each of the three patterns above.

A *tree* is a well-formed expression over a *signature* Σ of function symbols, where each symbol is indexed by an integer number, called its *arity*. For example, $u = f(f(f(a, a), h(a)), h(a))$ is a tree over the signature $\Sigma = \{f, h, a\}$, where symbols f, h, a have arity 2,1,0 respectively. The set of trees over Σ is denoted by $T(\Sigma)$. From algebraic point of view, $T(\Sigma)$ is a free Σ-algebra generated by Σ. Thus, we are dealing with node-labeled trees representing *first-order terms* over a given signature. We will use the words *tree* and *term* interchangeably. Clearly, we assume that the signature contains at least one 0-arity (constant) symbol, otherwise the set of terms is empty. A *tree pattern* is a tree over $\Sigma \cup X$, where X is an infinite set of 0-arity symbols of *variables*. Thus, $f(x, h(y))$, $f(f(f(y, a), x), x)$ are tree patterns over $\{f, h, a\}$, where x, y are variables. A *subtree* of a tree t is a subexpression of t. In other words, a subtree of t is a tree occurring at some node of t. The subtrees of u are $f(f(f(a, a), h(a)), h(a))$, $f(f(a, a), h(a))$, $f(a, a)$, $h(a)$ and a. Note that $h(a)$ and a have several occurrences in u. A *substitution* is a homomorphism $\sigma : T(\Sigma \cup X) \to T(\Sigma)$ such that $\sigma(a) = a$ for each constant a from Σ. Again, if $t = \sigma(p)$ for some term t, pattern p and substitution σ, then t is said to be an *instance* of p, and p is said to *match* t. Similar to words, a substitution replaces variables in patterns by trees such that the same variable is replaced by the same term. For example, term u is an instance of both patterns $f(x, h(y))$ and $f(f(f(y, a), x), x)$, but is not an instance of $f(f(x, x), h(a))$.

Note that words can be represented as trees in at least two ways. One way is to map each letter to a distinct unary symbol, and to add to the signature one constant symbol. Then a word can be naturally represented by a non-branching tree. However, to represent a pattern consistently, we need to introduce variables at internal nodes (second-order variables) which does not fit to our framework. Another way is to map each letter to a corresponding constant symbol and use one additional binary symbol for concatenation. In this case, however, one word is represented by several trees, due to the associativity property of concatenation. In general, words can be seen as trees over one associative function symbol. We will see in this paper that this associativity property makes many problems on words much more difficult than their counterparts for trees.

3 Problems

We now state the problems we will address in this paper. We assume that we are given a set S of word (resp. tree) patterns. As defined in Introduction, $Inst(S)$ denotes the set of word (resp. tree) instances of patterns of S, and $Cont(S)$ denotes the set of words (trees) having respectively a subword (subtree) that is an instance of a pattern from S. If S consists of a single pattern p, we will write $Inst(p)$ and $Cont(p)$ as a short-hand for $Inst(\{p\})$ and $Cont(\{p\})$.

We are interested in the following problems for both words and trees. Below u is a word (resp. tree), p is a word (resp. tree) pattern, and S is a set of patterns.

P1.1 $u \in Inst(p)$?
P1.2 $u \in Cont(p)$?
P2.1 is $\overline{Inst(S)}$ a finite set?
P2.2 is $Inst(S)$ a regular set?
P3 $Inst(p) \subseteq Inst(S)$?
P4 $Inst(p) \subseteq Cont(S)$?
P5.1 is $\overline{Cont(S)}$ a finite set?
P5.2 is $Cont(S)$ a regular set?

These questions are standard language-theoretic problems. P1.1 and P1.2 are membership problems for $Inst$- and $Cont$-languages. Since $Inst(S)$ and $Cont(S)$ are generally infinite, it makes sense to ask if these sets are co-finite. This justifies problems P2.1 and P5.1. Problems P2.2 and P5.2 ask whether $Inst(S)$ (respectively $Cont(S)$) is a regular set of words (trees). If the notion of regular word set (language) is well-known, the notion of regular tree language is probably less standard. For readers who are not familiar with regular tree languages, we refer to books [GS84,NP92]. Finally, problems P3 and P4 are also usual language inclusion questions, as $Inst(S) = \cup_{p \in S} Inst(p)$, and $Inst(S) \subseteq L$ iff for all $p \in S$, $Inst(p) \subseteq L$.

4 The Tree Case

We now start with the tree case and survey what is known here about the questions above. This will motivate our study and will allow to compare these results with their counterparts for the word case.

P1.1 is a trivial problem for the tree case. It asks whether a term is an instance of a tree pattern, which can be easily done in linear time. It is sufficient to check if the pattern coincides with the term at all non-variable positions, and check that the subterms of the term corresponding to distinct occurrences of the same variable in the pattern coincide. Clearly, this can be done in time $O(|u| + |p|)$.

P1.2 is the *subterm matching* problem which has numerous applications in functional and logic programming, automated deduction, term rewriting and other areas related to symbolic computation. The problem consists of testing whether a given pattern occurs in a given tree, that is matches one of its subtrees. Usually, one wants also an algorithm to find all such subtrees, and not only to test if there is one. The restricted version of this problem, when the pattern contains only linear variables, is known under the name *tree matching*. In early 80's, a simple practical solution has been proposed [HO82]. More recently, a series of work has been done to find the most efficient (in the worst-case) algorithm for tree matching. We refer to the latest achievement [CHI99] which proposes an $O(n \log^3 n)$ deterministic algorithm, where n is the size of the tree (assumed to be bigger than the size of the pattern). The algorithm (as well as previously proposed theoretically efficient algorithms) is however rather complicated and difficult to implement, and the problem of designing an efficient and practical tree matching algorithm is still on the agenda. Now, if a pattern contains non-linear variables, we can preprocess the subject tree by indexing its nodes in such a way that if the subtrees rooted in two nodes are the same, then these nodes have the same index. This preprocessing can be done in linear time (under the assumption that the signature has a constant size) by a bottom-up traversal of the tree. Then we can "forget" about repeated variables in the pattern and consider all variable nodes to be labeled by distinct variables. We then run a tree matching algorithm for linear patterns, and check, each time we find an occurrence of the linear pattern, if the subterms corresponding to occurrences of the same variable in the original pattern are equal (by looking at their indexes). This comparison takes time proportional to the maximal number of occurrences in the original pattern ($O(|p|)$ in the worst case), which introduces a $|p|$ factor with respect to the theoretic complexity of linear pattern matching. We refer to [RR92] for a detailed algorithm of subterm matching in presence of non-linear variables.

Let us now turn to problem P2.1, and consider a generalization of it. Instead of asking whether $\overline{Inst(S)}$ is finite, we ask if $\overline{Inst(S)}$ can be itself represented as $Inst(S')$ for some finite set of patterns S'. Such a set S' is called a *complement representation* of S [KP98]. Again, non-linear variables in patterns of S play an important role. Consider the set $S = \{h(x), f(h(x), y)\}$ over the signature $\{f, h, a\}$ as above. Then the set $S' = \{a, f(a, x), f(f(x, y), z)\}$ is a complement representation of S. One can generalize this and prove that if all patterns in the set are linear, a finite complement representation of this set can be constructed. However, one can prove that the set $S = \{f(x, x)\}$ does not have a finite complement representation. The exhaustive analysis of the situation has been given in [LM87]. The main result can be stated as follows.

Theorem 1 ([LM87]). *A set of patterns S has a finite complement representation iff there exists a set of* linear *patterns S_{lin} such that $Inst(S) = Inst(S_{lin})$. Moreover,*

- *if such a set S_{lin} exists, it can be obtained by instantiating the non-linear variables in the patterns of S by terms,*

– *the property of having a finite complement representation is decidable.*

Let us illustrate Theorem 1 by an example. Consider the set $S = \{a, f(x, h(y)), f(x, x),\ f(x, f(y, z))\}$, still over the signature $\{f, h, a\}$. This set contains a non-linear term $f(x, x)$. However, a simple analysis shows that $f(x, x)$ can be replaced by $f(a, a)$ without changing the set of instances. Thus, $Inst(S) = Inst(S_{lin})$, where S_{lin} is a set of linear patterns obtained from S by substituting a to x in the term $f(x, x)$. Furthermore, as S_{lin} contains only linear patterns, a complement representation of S_{lin} can be constructed: $S'_{lin} = \{h(x), f(h(x), a), f(f(x, y), a)\}$. Theorem 1 asserts that this example is typical: if a finite complement exists, the set is "linearizable", that is non-linear variables can be replaced by terms without changing the set of instances. The decidability of this property, stated in Theorem 1, means that a bound on the size of terms replacing non-linear variables can be effectively computed.

Recently, the study of finite complement representations has received a new impulse [GP99,Pic99], motivated by its applications in different areas, and in particular in logic programming. In [Pic99], it has been proved that testing if a given set has a finite complement representation (see Theorem 1) is co-NP-complete.

Coming back to problem P2.1, to check if $\overline{Inst(S)}$ is finite, we first check, according to Theorem 1, if S has a finite complement representation. If the answer is positive, we compute such a representation. If all patterns in the representation are terms (i.e. do not contain variables), then $\overline{Inst(S)}$ is finite. Otherwise, if at least one pattern has a variable, $\overline{Inst(S)}$ is infinite. This shows that P2.1 is in co-NP. The NP-hardness of P2.1 follows from [KNRZ91], where it was proved that deciding if $\overline{Inst(S)} = \emptyset$, is co-NP-complete. An easy modification of the hardness part of this proof shows that P2.1 is co-NP-hard, and therefore co-NP-complete.

Theorem 1 gives actually an answer to problem P2.2 too. It is an easy exercise to prove that if a set S contains only linear patterns, $Inst(S)$ is a regular tree language [GS84,NP92]. Thus, when a set is "linearizable" in the sense of Theorem 1, the set of instances is regular. On the other hand, if a set is not linearizable, it can be shown using a pumping lemma argument that the set of instances is not regular. This is however not easy to prove, but follows from the work [Kuc91] that we will survey below. We summarize the discussion in the following statement.

Proposition 2. *In the tree case, P2.1 and P2.2 are co-NP-complete problems.*

Now let us skip problem P3 for a moment and turn to problem P4 which has now a more-than-ten-years history. The problem, known under the name of *ground reducibility* problem, has attracted a lot of attention in the area of *term rewriting* [DJ90] because of its application to automated inductive proofs [JK89]. The problem consists of testing if all instances of a given tree pattern p have a subtree matched by one of the patterns of a given set S. Once again, non-linear variables in patterns of S make the problem much more difficult. In the middle and late 80's, several authors observed that the problem is decidable if patterns of S only contain linear variables. The problem was first proved decidable in the general case by Plaisted [Pla85], and later by other authors independently [KNZ87,Com88]. Recently, the problem was shown to be EXPTIME-complete [CJ97].

Problem P3 can be expressed in terms of P4 in the following way. Assume we have a pattern p and a set of patterns S, and we want to test whether $Inst(p) \subseteq Inst(S)$. First delete from S those patterns which do not have the same root symbol as the root symbol of p (obviously, these patterns cover no instance of p). Then choose a new symbol α and replace the root symbol in p and in all remaining patterns in S by α. Let p' and S' be the resulting pattern and set respectively. It can be shown that $Inst(p) \subseteq Inst(S)$ iff $Inst(p') \subseteq Cont(S')$. The latter property, which is a special instance of ground reducibility, can be expressed as the so called *sufficient completeness* property for specifications with free constructors (see [KNRZ91]). Deciding this property has been proved co-NP-complete in [KNRZ91].

Proposition 3. *In the tree case, P3 and P4 are both decidable problems. P3 is co-NP-complete and P4 is EXPTIME-complete.*

Finally, let us turn to problems 5.1 and 5.2. Problem 5.1 has been proved decidable in [Pla85,KNZ87]. Concerning Problem 5.2, the following Theorem has been proved in [Kuc91].

Theorem 4. *For a set of patterns S, $Cont(S)$ is a regular tree language iff there exists a set of* linear *patterns S_{lin} such that $Cont(S) = Cont(S_{lin})$. Moreover,*

– *if such a set S_{lin} exists, it can be obtained by instantiating the non-linear variables in the patterns of S by terms.*

Theorem 4 is a lifting of Theorem 1 from the set of instances $Inst(S)$ to the set $Cont(S)$ of terms containing instances of S as subterms. The latter case is however much more difficult, and the proof of Theorem 4 used a non-constructive combinatorial argument, based on Ramsey Theorem. Therefore, no effective bound on the size of terms to be substituted for the non-linear variables, resulted from the proof, and the decidability of the regularity of $Cont(S)$ remained an open problem. This problem, considered important in the area of rewriting, has appeared in the list of major open problems in rewriting in [DJK91]. Soon after, the regularity of $Cont(S)$ has been proved decidable by three groups of authors [KT92,VG92,HH92]. The results of [KT95] provided also a new proof of the decidability of problem 5.1, and even gave an effective bound on the size of $\overline{Cont(S)}$ in the case it is finite. We then conclude this section with the following

Proposition 5. *In the tree case, P5.1 and P5.2 are both decidable problems.*

5 The Word Case

The overview of the tree case given in the previous section shows that all the problems are decidable, though the complexity of some of them appears to be high. In this section we study these problems in the word case and see that most of them, and even some restricted versions of them, turn out to be undecidable. We also analyze the complexity of these problems in the case of linear patterns.

We first remark that in the tree case, $Cont(S)$ is a "meta-notion" with respect to $Inst(S)$, due to the fact that the notion of subtree cannot be expressed by means of patterns, as only first-order variables are allowed in patterns. In contrast, in the word case $Cont(S)$ can be expressed in terms of $Inst(S)$:

$$Cont(S) = Inst(\{xpy, xp, py, p | p \in S \text{ and } x, y \text{ do not occur in } p\})$$

This implies that, in contrast to the tree case, the problem for $Cont(S)$ is simpler than its counterpart for $Inst(S)$. In particular, if a problem is decidable for $Inst(S)$, it is also decidable for $Cont(S)$. On the other hand, if a problem is undecidable for $Inst(S)$, the undecidability of its counterpart for $Cont(S)$ may be harder to prove. We will face this situation later in this section.

Note another difference with the tree case: in contrast to trees, we may allow variables in word patterns to be substituted by the empty word. This gives rise to two cases depending of whether this possibility is allowed or not. Following Kari et al. [KMPS95], we call these cases *erasing* (*E-case* for short), if substituting by the empty word is allowed, and *non-erasing* (*NE-case*), if it is not allowed. We will generally speak about the NE-case, unless the E-case is explicitly mentioned.

An early result of Angluin [Ang80] asserts that problem P1.1 is NP-complete. This implies that P1.2 is also NP-complete, as $w \in Inst(p)$ iff $\#w\# \in Cont(\#p\#)$ where $\#$ is a fresh letter. This NP-completeness result immediately shows that the word case appears to be much more difficult, as P1.1 and P1.2 are polynomial problems in the tree case, of low polynomial degree. However, if pattern p is linear, P1.1 and P1.2 can be solved in linear time, as they actually reduce to the well-known string matching problem, and can be solved, e.g., by the Knuth-Morris-Pratt algorithm [CR95]. In the general case, the naive algorithm solving P1.1 is in $O(|w|^{\Delta})$ (respectively $O(|w|^{\Delta+2})$ for P1.2), where Δ is the number of distinct variables in p. Néraud [Nér95] showed how this complexity can be slightly reduced (roughly, the exponent can be decreased by 2) and obtained some specialized efficient algorithms for P1.2 for the cases of low Δ (1 or 2).

Proposition 6. *In the word case, problems P1.1 and P1.2 are NP-complete. Both problems can be solved in linear time if pattern p is linear.*

The difficulty of matching problems P1.1 in the case of words can be also illustrated by the fact that if a word w is matched by a pattern p, that is $w = \sigma(p)$, then substitution σ does not have to be unique. For example, pattern xy can match a word w in $(|w| - 1)$ different ways, corresponding to the factorizations of w into two parts. It is easy to see that many patterns admit this situation (e.g. all linear patterns), but not all of them – for example, patterns x, xx (and more generally, one-variable patterns) have a unique way to match a word. Formally, a pattern p is called *non-ambiguous* if there is a unique way for p to match each word of $Inst(p)$, and *ambiguous* otherwise. The ambiguity of patterns was studied by Mateescu and Salomaa [MS94]. They introduced the notion of *degree of ambiguity* of a pattern p defined as the maximal number of ways for p to match a word from $Inst(p)$ provided this number is finite; otherwise the degree of ambiguity is ∞. It is easy to exhibit patterns with the degree of ambiguity 1 or ∞, and much more difficult with a finite degree of ambiguity different from 1. In [MS94], it was shown that pattern $p = xabxbcayabcy$ has the degree of ambiguity 2. For example, there are two ways for p to match the word $caabcabcaabcbcabcabcbc$, and any word from $Inst(p)$ is matched by p in at most two ways. The authors also found a pattern of degree of ambiguity 3, and by some composition technique, patterns of any degree $2^m 3^n$. However, they state it as an open question if *every* finite degree of ambiguity is realizable by some pattern. The decidability status of determining if the degree of ambiguity of a pattern is finite, is also open.

Let us now turn to problem P3. A striking result has been proved in [JSSY93]: inclusion $Inst(p) \subseteq Inst(S)$ is undecidable even if S consists of a single pattern. This contrasts to the fact that the equivalence problem $Inst(p_1) = Inst(p_2)$ is trivial: the equivalence holds iff p_1 and p_2 are equal modulo a variable renaming. The latter is however true only in the NE-case, and for the E-case the decidability status of the equivalence problem $Inst(p_1) = Inst(p_2)$ is open. We also point out to paper [Fil88] for some results about the inclusion problem $Inst(p_1) \subseteq Inst(p_2)$ in the E- and NE-case.

Proposition 7. *In the word case, problem P3 is undecidable even if S consists of a single pattern.*

Formally, the undecidability result of [JSSY93] for problem P3 does not imply the undecidability of problem P4 (see the discussion in the beginning of this section). Problem P4 has been studied in [KR95b], where it has been proved undecidable.

Proposition 8. *In the word case, problem P4 is undecidable.*

An interesting feature of the proof of [KR95b] is that it implies that the problem $Inst(p) \subseteq Cont(S)$ remains undecidable if p has a very simple form, namely the form axa, where a is a letter and x a variable. It seems very difficult (if at all possible) to further simplify p. We will come back to this issue below.

Based on the proof of the result of [KR95b], we now establish a new result.

Theorem 9. *In the word case, problem P2.1 is undecidable.*

Proof. We give a very general idea of the proof. To reconstruct the details, the reader is referred to [KR95b].

First, we review the proof of [KR95b] of Proposition 8. To show that $Inst(axa) \subseteq Inst(S)$ is undecidable, the construction of S is based on the following idea. The instances of $p = axa$ are assumed to encode runs of a given deterministic Minsky (two-register) machine M on a given data d. Patterns of S are designed in such a way that every instance of p which does not encode a correct run of machine M on data d, contains some pattern from S. To put it in another way, an instance of p which does not contain any pattern of S, must encode a correct finite run of machine M on data d. Therefore, there exists an instance of p which does not contain an instance of S iff M halts on d, which is an undecidable property.

To prove Theorem 9, we modify the proof as follows. We modify the set of patterns S in such a way that S encodes only a Minsky machine M, and does not specify any input data d. Assume that S' is the modified set of patterns. Consider now the set of patterns

$$\begin{aligned}\tilde{S} = \{\alpha x | \alpha \in A,\ \alpha \neq a\} \cup \{x\alpha | \alpha \in A,\ \alpha \neq a\} \cup \\ \{xpy | p \in S' \text{ and } x, y \text{ do not occur in } p\},\end{aligned} \tag{1}$$

where a is the same letter as in the pattern p above. From the previous discussion, it is clear that the words which are in $\overline{Inst(\tilde{S})}$ are words of the form awa, which are not instances of S'. By construction of S', these are words which encode a correct finite run of the machine M on some input data. Since it is undecidable if a machine stops on a finite number of input data, it is undecidable if the set $\overline{Inst(\tilde{S})}$ is finite or not.

The decidability status of Problem P2.2 is open [KMPS95]. The inverse problem, whether a given regular language is expressible as $Inst(S)$ is also not known to be decidable. It is also open if it is decidable for a language $Inst(S)$ to be context-free. However, it was proved in [KMPS95] that it is undecidable if a given context-free language is expressible as $Inst(S)$.

Let us now consider problem P5.1. The proof of Theorem 9 above may suggest that P5.1 is not so much different from P2.1 and must be also undecidable by a similar proof. Indeed, all "important" patterns occur in the third set of (1), and patterns in the first and the second sets are extremely simple – they consists of a single letter followed or preceded by a variable. However, these "extremely simple" patterns play a crucial role as they actually specify the first and last letter in the words of the language, which is necessary for an undecidability proof (see [KR95b]).

The decidability status of Problem 5.1 is open. Actually, it is the most general version of the famous *avoidability* problem. The avoidability problem was studied in the word combinatorics under a very restricted form – when S contains a single pattern p, and moreover, p contains only variables and no letters. However, even in this restricted form the problem turns out to be extremely difficult.

It is not known if testing the finiteness of $\overline{Cont(p)}$ is decidable or not. The author of [Cur93] offered 100 US dollars[1] for a solution of this problem.

[1] 2278.78 russian rubles as for February 12, 1999

A pattern p is called *unavoidable* (*blocking* according to the terminology of [Zim84]) if $\overline{Cont(p)}$ is finite, and *avoidable* otherwise. Clearly, p is avoidable iff there exists an infinite word which does not contain (finite) subwords which are instances of p.

Interestingly, a study of avoidability is historically at the origin of word combinatorics and formal language theory. Back to the beginning of the century, Axel Thue obtained his famous construction of an infinite *square-free* word on the three-letter alphabet and an infinite *cube-free* word on the two-letter alphabet. In the terminology of pattern avoidance, a square-free and cube-free word is a word which does not contain respectively the pattern xx and xxx. Trivially, xx is unavoidable on two letters and xxx is unavoidable on one letter. A pattern which is avoidable on four letters but not on three letters has been described in [BEM79]. No pattern is known which is avoidable on k letters but unavoidable on $k-1$ letters for $k > 4$.

The above discussion shows that the size of the alphabet may be crucial in avoiding patterns. We refer to [Cas94] for a survey of the state-of-the-art in pattern avoidance. A key result in the area is an algorithm proposed independently in [BEM79,Zim84], which decides if *there exists an alphabet* on which a given pattern can be avoided. However, as was mentioned above, it is not known if *for a fixed alphabet* one can decide, given a pattern, if it is avoidable on this alphabet.

The rest of the paper is devoted to analyzing some of our problems in case the set S consists of *linear* patterns. We already mentioned that problems P1.1 and P1.2 can be efficiently solved if p is a linear pattern. For the other problems we will see that although they become decidable in the linear case, they remain untractable.

Note that if S consists of linear patterns, the languages $Inst(S)$ and $Cont(S)$ are regular languages specified by a regular expression of the form

$$\cup_{i=1}^{n}(A^*)w_{i1}A^*w_{i2}\dots A^*w_{ik_i}(A^*), \tag{2}$$

where w_{ij}'s are words and parenthesis indicate that A^* may or may not occur in the beginning and the end of the expression. Thus, problems P2.2 and P5.2 are always positively answered. Note also that inclusion and equivalence of regular languages specified by general regular expressions is a PSPACE-complete problem (cf [GJ79]).

In [KR95a] problems P4 and P5.1 have been studied under the condition that the patterns of S are linear. As for P4, it has been proved that it is decidable in this case, regardless if p is linear or not. If p is restricted to be linear too, the problem has been proved to be co-NP-complete [KR95a]. The exact complexity of the case when patterns of S are linear but pattern p is not, is not known to us. However, if the maximal number of occurrences of a variable is bounded, the problem remains co-NP-complete.

Proposition 10. *Problem P4 of testing $Inst(p) \subseteq Cont(S)$ is decidable if S consists of linear patterns. If p is linear in addition, the problem is co-NP-complete.*

It was also proved in [KR95a] that if S is restricted to contain linear patterns only, problem P5.1 is co-NP-complete too.

To move on, we need to sketch the co-NP-completeness proofs from paper [KR95a]. Consider problem P4 for the case that pattern p and all patterns of S are linear. The co-NP-hardness of this problem is easy to show. We refer to [KR95a] for the reduction from MONOTONE-ONE-IN-THREE-SAT. However, proving the membership in co-NP represents a non-trivial part. It amounts to show that if $Inst(p) \not\subseteq Cont(S)$, there is an instance of *p of size polynomial on* $(|S| + |p|)$ which does not contain any pattern from S. Of course, the language $Cont(S)$ and its complement $\overline{Cont(S)}$ are regular, as $Cont(S)$ has form (2). The proof of [KR95a] consisted of defining a compact *deterministic finite automaton* (DFA) for these languages verifying the following key property: although the total size (number of states) of this automaton is exponential in $|S|$, the length of the longest loop-free path from the initial to the finite state is of polynomial length. We refer to [KR95a] for further details.

This property of the automaton allowed to show that in case $Inst(p) \not\subseteq Cont(S)$, the minimal size of an instance of p which is not in $Cont(S)$ has a size polynomial on $|S|$. Similarly, if $\overline{Cont(S)}$ is finite (problem P5.1), we can give a polynomial bound on the length of words in $\overline{Cont(S)}$. This provides a key argument in the co-NP-completeness proof.

Here we use this argument to show the co-NP-completeness of two other problems – P3 (in case p is a linear pattern) and P2.1.

Since P3 is a more general problem than P4 in the word case, P4 is co-NP-hard if p is a linear pattern. Similarly, P2.1 is more general than P5.1 and is then also co-NP-hard. To prove that both of them are in co-NP, we use an adaptation of the deterministic automaton construction from [KR95a] from the language $Cont(S)$ to $Inst(S)$. We skip the details of the construction which would require us too much space, and summarize the results in the following statement.

Theorem 11. *Assuming a linear pattern p and a set of linear patterns S, problems P2.1, P3, P4 and P5.1 are co-NP-complete.*

Finally, for a linear pattern p, following [Shi82], we can build a DFA recognizing $Inst(p)$ in polynomial (linear) time: if $p = (x_0)u_1x_1 \dots x_{n-1}u_n(x_n)$ $(u_i \in A^+, x_i \in X)$, the idea is to build DFA's $D_1, \dots, D_n$ recognizing respectively $Cont(u_1), \dots, Cont(u_n)$, and then to identify the final state of D_i with the initial state of D_{i+1}. This construction implies, in particular, that for the special case of P3 and P4 where p is linear and S consists of a single linear pattern, a solution can be obtained in polynomial time: the question $Inst(p) \subseteq Inst(p')$ is equivalent to the emptiness of the language $Inst(p') \cap \overline{Inst(p)}$ whose DFA is easily derived in polynomial (quadratic) time [HU79].

Proposition 12. *Assuming a linear pattern p and $S = \{p'\}$ with p' a linear pattern, P3 and P4 can be checked in polynomial time.*

6 Conclusions

In this paper we formulated several language-theoretic problems which are meaningful for any combinatorial structure equipped with a notion of pattern and a substructure relation. We then studied the algorithmic complexity of those problems for two particular structures – trees over a finite signature and words over a finite alphabet. It turns out that the instances of these problems for words and trees cover a large area of research, including seemingly quite unrelated subareas. Some problems on trees have been studied in term rewriting theory, with relation to the theory of tree languages. Some other problems, such as tree matching, have received much attention in the area of algorithm development. Applied to words, those problems have been studied in the area of word combinatorics and formal language theory, including the recent research stream on pattern languages. Again, the matching problem for words has been subject of intensive studies in the algorithmics area. We found it interesting that all these problems can be expressed uniformly as classical problems on languages specified by patterns.

We attempted to give a brief survey of considered problems, putting the stress on comparing the tree and the word case. Moreover, we gave several new results for the word case. We showed that all problems are easier on the tree case than their counterparts for the word case. In particular, except for the matching problem, all problems are decidable in the tree case and undecidable in the word case. For the word case, we gave a special attention to the linear case, where the problems become decidable but, as we have showed, remain of high algorithmic complexity.

References

[Ang80] D. Angluin. Finding patterns common to a set of strings. *J. Comput. System Sci.*, 21:46–62, 1980.

[BEM79] D.R. Bean, A. Ehrenfeucht, and G.F. McNulty. Avoidable patterns in strings of symbols. *Pacific J. Math.*, 85(2):261–294, 1979.

[Cas94] J. Cassaigne. *Motifs évitables et régularités dans les mots.* Thèse de doctorat, Université Paris VI, 1994.

[CHI99] R. Cole, R. Hariharan, and P. Indyk. Tree pattern matching and subset matching in deterministic $o(n \log^3 n)$-time. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltymore, Maryland, January 17-19, 1999*, pages 245–254. ACM, SIAM, 1999.

[CJ97] H. Comon and F. Jacquemard. Ground reducibility is EXPTIME-complete. In *Proceedings, Twelth Annual IEEE Symposium on Logic in Computer Science*, pages 26–34, Warsaw, Poland, 29 June–2 July 1997. IEEE Computer Society Press.

[Com88] H. Comon. *Unification et disunification. Théories et applications.* Thèse de Doctorat d'Université, Institut Polytechnique de Grenoble (France), 1988.

[CR95] M. Crochemore and W. Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13:405–425, 1995.

[Cur93] J. Currie. Open problems in pattern avoidance. *American Mathematical Monthly*, 100:790–793, 1993.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.

[DJK91] N. Dershowitz, J.-P. Jouannaud, and J. W. Klop. Open problems in rewriting. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 445–456. Springer-Verlag, 1991.

[Fil88] G. Filè. The relation of two patterns with comparable languages. In R. Cori M. Wirsing, editor, *Proceedings of the 5th Annual Symposium on Theoretical Aspects of Computer Science (STACS '88)*, volume 294 of *Lecture Notes in Computer Science*, pages 184–192, Bordeaux, France, February 1988. Springer.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability. A guide to the theory of NP-completeness.* W. Freeman and Compagny, New York, 1979.

[GP99] G. Gottlob and R. Pichler. Working with ARMs: Complexity results on atomic representations of Herbrand models. In *Proceedings of LICS'99*, 1999. to appear, available from `http://www.dbai.tuwien.ac.at/staff/gottlob/arms.ps`.

[GS84] F. Gécseg and M. Steinby. *Tree automata.* Akadémiai Kiadó, Budapest, Hungary, 1984.

[HH92] D. Hofbauer and M. Huber. Computing linearizations using test sets. In M. Rusinowitch and J.-L. Rémy, editors, *Proceedings 3rd International Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson (France)*, pages 145–149. CRIN and INRIA-Lorraine, 1992.

[HO82] C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Publishing Company, Reading, Mass., USA, 1979.

[JK89] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82:1–33, 1989.

[JSSY93] T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Inclusion is undecidable for pattern languages. In Svante Carlsson Andrzej Lingas, Rolf G. Karlsson, editor, *Automata, Languages and Programming, 20th International Colloquium*, volume 700 of *Lecture Notes in Computer Science*, pages 301–312, Lund, Sweden, 5–9 July 1993. Springer-Verlag.

[KMPS95] L. Kari, A. Mateescu, G. Paun, and A. Salomaa. Multi-pattern languages. *Theoretical Computer Science*, 141:253–268, 1995.

[KNRZ91] D. Kapur, P. Narendran, D. J. Rosenkrantz, and H. Zhang. Sufficient completeness, ground-reducibility and their complexity. *Acta Informatica*, 28:311–350, 1991.

[KNZ87] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[KP98] G. Kucherov and D. Plaisted. The complexity of some complementation problems. submitted, 1998.

[KR95a] G. Kucherov and M. Rusinowitch. Complexity of testing ground reducibility for linear word rewriting systems with variables. In *Proceedings 4th International Workshop on Conditional and Typed Term Rewriting Systems, Jerusalem (Israel)*, volume 968 of *Lecture Notes in Computer Science*, pages 262–275. Springer-Verlag, 1995.

[KR95b] G. Kucherov and M. Rusinowitch. Undecidability of ground reducibility for word rewriting systems with variables. *Information Processing Letters*, 53:209–215, 1995.

[KT92] G. Kucherov and M. Tajine. Decidability of regularity and related properties of ground normal form languages. In M. Rusinowitch and J.-L. Rémy, editors, *Proceedings 3rd International Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson (France)*, pages 150–156. CRIN and INRIA-Lorraine, 1992.

[KT95] G. Kucherov and M. Tajine. Decidability of regularity and related properties of ground normal form languages. *Information and Computation*, 118(1):91–100, April 1995.

[Kuc91] G. A. Kucherov. On relationship between term rewriting systems and regular tree languages. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 299–311. Springer-Verlag, April 1991.

[LM87] J.-L. Lassez and K. Marriot. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–318, 1987.

[MS94] A. Mateescu and A. Salomaa. Nondeterminism in patterns. In P. Enjalbert, E.W. Mayr, and K.W. Wagner, editors, *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science (STACS'94), Caen, France, February 1994*, volume 775 of *Lecture Notes in Computer Science*, pages 661–668. Springer-Verlag, 1994.

[Nér95] J. Néraud. Detecting morphic images of a word: On the rank of a pattern. *Acta Informatica*, 32:477–489, 1995.

[NP92] M. Nivat and A. Podelski, editors. *Tree Automata and Languages*. Studies in Computer Science and Artificial Intelligence 10. North-Holland, 1992.

[Pic99] R. Pichler. The explicit representability of implicit generalizations. submitted, april 1999.

[Pla85] D. Plaisted. Semantic confluence and completion method. *Information and Control*, 65:182–215, 1985.

[RR92] R. Ramesh and I.V. Ramakrishnan. Nonlinear pattern matching in trees. *Journal of the ACM*, 39(2):295–316, April 1992.

[Shi82] T. Shinohara. Polynomial time inference of pattern langages and its applications. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, Mathematical Theory of Computations/The Complexity of Algorithms*, pages 191–209, 1982.

[VG92] S. Vágvölgyi and R. Gilleron. For a rewriting system it is decidable whether the set of irreducible ground terms is recognizable. *Bulletin of European Association for Theoretical Computer Science*, 48:197–209, 1992.

[Zim84] A.I. Zimin. Blocking sets of terms. *Math. USSR Sbornik*, 47:353–364, 1984. Original version in russian published in 1982, 119 (3), 363–375.