Patterns

Kai Salomaa
School of Computing, Queen's University
Kingston, Ontario K7L 3N6, Canada
Email: ksalomaa@cs.queensu.ca

July 30, 2003

Abstract

We review topics on formal language aspects of patterns. The main results on the equivalence and inclusion problems are presented. We discuss open problems, in particular, concerning pattern language decision problems and ambiguity in patterns.

1 Introduction

Besides using generative or recognition devices, we can define formal languages by giving a pattern, or patterns, that all strings in the language should follow. For example, a pattern xyx mandates that all strings in the language must have a prefix that occurs also as a suffix of the string and that, furthermore, the given prefix and suffix do not overlap. The formal definition of patterns and pattern languages is due to Angluin [2, 3]. Patterns are commonly used in the extended regular expressions in systems like Perl, Python or `egrep` that allow backreferencing [1, 5, 10]. Roughly speaking, backreferences act as variables in patterns.

A pattern is a string of terminals and variable symbols. The corresponding language is obtained by uniformly replacing the variables with arbitrary terminal strings in the case of E-patterns and by non-empty terminal strings in the case of NE-patterns (non-erasing patterns). Here uniformly means that all occurrences of the same variable are substituted by the same string.

The family of pattern languages is in some sense orthogonal to the Chomsky hierarchy. From a language theoretic point of view an interesting property is that while the equivalence of NE-patterns is decidable, inclusion is undecidable for both NE- and E-patterns. On the other hand, it remains an open question whether we can decide equivalence of E-patterns and only some partial results are known [18, 30].

In this paper we survey results and open problems concerning pattern languages. The second section presents basic definitions and properties of NE- and E-pattern languages.

Sections 3 and 4 discuss the equivalence and inclusion problem, in particular, some approaches to attack the open E-pattern equivalence problem. Section 5 discusses unambiguity and degrees of ambiguity for pattern languages, an area where several important questions remain open. In the last section we briefly describe other related topics, namely applications of pattern languages in computational learning theory, as well as, generalizations of pattern languages. In particular, there remains much unexplored territory and open questions within the realm of pattern systems [28, 29].

2 Basic Properties of Pattern Languages

In the following let Σ be a finite alphabet, elements of Σ are called *terminals.* The set of (non-empty) finite strings over Σ is Σ^* (Σ^+) and λ denotes the empty string.

By a *pattern* we mean a finite non-empty string over $\Sigma \cup V$ where V is an enumerable set of variables.

We make the convention that elements of Σ are denoted by symbols 0, 1 or lower-case letters from the initial part of the alphabet, and variables are denoted by lower-case letters from the end of the alphabet, possibly with subscripts (symbols $x, y, z, x', y', z', x_1, x_2, \ldots$). Patterns are typically denoted by lower-case Greek letters. The set of variables occurring in a pattern α is $\mathrm{var}(\alpha)$.

Let s be a function from V to Σ^*. The function s is extended to a morphism from $(\Sigma \cup V)^* \longrightarrow \Sigma^*$ that is *stable for terminals* by setting: $s(a) = a$ for all $a \in \Sigma$, and $s(\alpha\beta) = s(\alpha)s(\beta)$ for all $\alpha, \beta \in (\Sigma \cup V)^*$. We denote by $S_{\Sigma,V}$ the set of all morphisms $(\Sigma \cup V)^* \longrightarrow \Sigma^*$ that are stable for terminals.

In the above definition it is convenient to allow the set of variables V to be infinite since patterns defining languages over an alphabet Σ may require arbitrarily many variables (whereas the terminal alphabet is fixed). As we will observe, the complexity of certain questions concerning pattern languages will essentially change if we would restrict consideration to a fixed finite set of variables. On the other hand, any fixed pattern can contain only finitely many variables. When specifying a function in $S_{\Sigma,V}$ it is common that we specify only the values of the variables that occur in the pattern(s) under consideration.

Especially from the point of view of decidability properties, it turns out to be an essential distinction whether or not the variables can be substituted by the empty string. For NE-patterns we allow only non-empty strings substituted for the variables, whereas in E-patterns the variables can be replaced also by the empty string. The original definition of Angluin [2, 3] uses NE-patterns. The first paper to systematically consider E-patterns is probably [13].

Strictly speaking, the difference between NE- and E-patterns concerns the languages defined by the patterns, and not the actual patterns as syntactic constructs. However, following common terminology we speak about NE-patterns and E-patterns with the meaning that in NE-patterns the variables have to be substituted with a non-erasing (that is, λ-free) morphism in $S_{\Sigma,V}$ and in E-patterns the variables can be substituted by an arbitrary morphism in $S_{\Sigma,V}$.

Definition 2.1 *The language denoted by an E-pattern* $\alpha \in (V \cup \Sigma)^+$ *is*

$$L_{E,\Sigma}(\alpha) = \{w \in \Sigma^* \mid w = s(\alpha) \text{ for some } s \in S_{\Sigma,V}\}.$$

The language denoted by an NE-pattern $\alpha \in (V \cup \Sigma)^+$ *is*

$$L_{NE,\Sigma}(\alpha) = \{w \in \Sigma^+ \mid w = s(\alpha) \text{ for some non-erasing } s \in S_{\Sigma,V}\}.$$

If Σ *is known from the context we write simply just* $L_E(\alpha)$ *and* $L_{NE}(\alpha)$.

The class of Z*-pattern languages,* $Z \in \{E, NE\}$*, over the alphabet* Σ *is denoted* $\mathcal{L}_{Z,\Sigma}$.

We say that NE-patterns (respectively, E-patterns) α and β are *equivalent* if $L_{NE}(\alpha) = L_{NE}(\beta)$ (respectively, $L_E(\alpha) = L_E(\beta)$).

The following basic properties are established already in [2] for NE-pattern languages. The corresponding results for E-patterns are obtained by a simple modification of the same arguments.

Proposition 2.1 *Let* Σ *be an alphabet of at least two symbols and* $Z \in \{E, NE\}$*. Then*

1. $\mathcal{L}_{Z,\Sigma}$ *is incomparable with the classes of regular languages and context-free languages.* $\mathcal{L}_{Z,\Sigma}$ *is contained in the context-sensitive languages.*

2. $\mathcal{L}_{Z,\Sigma}$ *is not closed under any of the following operations: union, intersection, complement, Kleene plus and star, morphism or inverse morphism.*

3. $\mathcal{L}_{Z,\Sigma}$ *is closed under concatenation and reversal.*

Note that already the simple (E- or NE-) pattern xx denotes a non-context-free language when Σ has at least two symbols.

We define for patterns the following "less general than" and "equivalent to" relations.

Definition 2.2 *Let* α *and* β *be patterns over* Σ *and* V.

1. $\alpha \leq \beta$ *iff* $\alpha = h(\beta)$ *for some* h *stable for terminals.*

2. $\alpha \leq_+ \beta$ *iff* $\alpha = h(\beta)$ *for some non-erasing* h *that is stable for terminals.*

3. $\alpha \equiv \beta$ *iff* $\alpha = h(\beta)$ *for some* h *that is a renaming of the variables.*

If $\gamma = h(\beta)$ we say that γ is an *instance* of β. If, additionally, $\gamma \in \Sigma^*$ it is called a terminal instance. The above definitions will be useful for determining whether two given patterns denote the same language. The notions have obvious analogies in rewriting systems with variables, and connections to rewriting systems will be discussed in the next section [21].

The language defined by a Z-pattern α, $Z \in \{E, NE\}$, can now be expressed as

$$L_{E,\Sigma}(\alpha) = \{w \in \Sigma^* \mid w \leq \alpha\}, \;\; L_{NE,\Sigma}(\alpha) = \{w \in \Sigma^* \mid w \leq_+ \alpha\}$$

It is immediate that the relations $\leq_+$ and $\leq$ are transitive. For any patterns α and β, $\alpha \leq_+ \beta$ (respectively, $\alpha \leq \beta$) implies that $L_{NE}(\alpha) \subseteq L_{NE}(\beta)$ (respectively, $L_E(\alpha) \subseteq L_E(\beta)$). It is easy to see [2] that

$$\alpha \equiv \beta \text{ if and only if } \alpha \leq_+ \beta \text{ and } \beta \leq_+ \alpha. \tag{1}$$

It is important to note that, on the other hand, there is no known simple characterization of patterns α and β that satisfy $\alpha \leq \beta$ and $\beta \leq \alpha$. This will have implications when trying to decide the equivalence of E-patterns [18, 30].

The NP-completeness of the uniform membership problem for NE- and E-patterns is established respectively in [2] and [17].

Theorem 2.1 *Let $Z \in \{E, NE\}$. The problem of deciding whether $w \in L_Z(\alpha)$, for a given terminal word $w \in \Sigma^*$ and pattern α, is NP-complete.*

Intuitively, the difficulty of the membership problem is caused by the fact that if w is a terminal instance of pattern α, $w = h(\alpha)$, the morphism h need not be unique. Questions of this type relate to the notion of ambiguity in patterns that will be discussed later. It is not difficult to see that the membership problem can be solved in polynomial time for a fixed pattern α (the polynomial depending on α). Also, if α is a linear pattern (any variable has at most one occurrence), the uniform membership problem reduces to the string matching problem and it can be solved in linear time by the well-known Knuth-Morris-Pratt algorithm [22].

Besides the language denoted by a pattern, we can consider also patterns that in a certain sense approximate a "sample" set of strings. The samples considered in computational learning theory are typically finite.

Let Σ be an alphabet and F a subset of Σ^+. We say that a Z-pattern, $Z \in \{E, NE\}$, α is *descriptive of* the sample F if $F \subseteq L_{Z,\Sigma}(\alpha)$, and there is no Z-pattern β such that $F \subseteq L_{Z,\Sigma}(\beta) \subset L_{Z,\Sigma}(\alpha)$.

Theorem 2.2 *Let $Z \in \{E, NE\}$. Every finite sample F has a descriptive Z-pattern and the pattern can be constructed algorithmically.*

The result was originally proved in [2] for NE-patterns, however the algorithm obtained from the proof is quite inefficient. A polynomial time algorithm for finding a pattern (of maximum length) that is descriptive within the class of all NE-patterns containing variables from an a priori fixed finite set is given in [16]. The construction of an E-pattern descriptive of a finite sample [17] is different and it is not known whether this can be done efficiently, for example, in polynomial time. Also it is not known whether infinite samples (e.g., given as a regular or a context-free language) necessarily have a descriptive sample, see [17].

In computational learning theory it is common that we have a set of positive samples that should be included in the language and a set of negative samples that should be excluded from the language. The *pattern consistency problem* asks whether for given finite samples F_1 and F_2 there exists a pattern α such that $F_1 \subseteq L_{NE}(\alpha)$ and $F_2 \cap L_{NE}(\alpha) = \emptyset$.

Theorem 2.3 [20] *The pattern consistency problem is complete for the class* Σ_2^P.

Above $\Sigma_2^P = \mathrm{NP}^{\mathrm{NP}}$ denotes the second level in the polynomial time hierarchy. The proof requires that the set of variables is not fixed.

3 The Equivalence and the Inclusion Problem

In formal language theory pattern languages are one of the few known naturally defined families for which equivalence is decidable (in the case of NE-patterns) and inclusion is undecidable (for both NE- and E-patterns). What is even more surprising is that establishing the decidability of NE-pattern equivalence is not too difficult (below we outline the proof following [2]), and because of this it was generally conjectured that also inclusion should be decidable. On the other hand, the decidability of equivalence for E-patterns remains an open question. After reviewing the known (un)decidability results we will discuss some special cases of this problem.

In the following we assume that Σ has cardinality at least 2 and let $a, b \in \Sigma$ be two fixed distinct symbols. (In the case $\mathrm{card}(\Sigma) = 1$, pattern languages over Σ are regular and equivalence can be determined easily.) Let $V = \{x_1, x_2, \ldots\}$. We define a particular sample that is useful for deciding the equivalence of two NE-patterns.

Consider the following set of morphisms stable for terminals, where g_j is defined for all $j \geq 1$:

$$f_a(x_i) = a, \quad f_b(x_i) = b, \qquad g_j(x_i) = \begin{cases} a \text{ if } i = j \\ b \text{ if } i \neq j \end{cases}, \qquad i \geq 1.$$

Then for a given pattern α we define $F(\alpha) = \{f_a(\alpha), f_b(\alpha)\} \cup \{g_i(\alpha) \mid i \geq 1\}$.

Clearly $F(\alpha)$ is a finite language, and we have $F(\alpha) \subseteq L(\alpha)$. Note that if α contains no variables, then $F(\alpha) = L(\alpha) = \{\alpha\}$, if α contains only one variable then $\mathrm{card}(F(\alpha)) = 2$, and if α contains $n \geq 2$ variables then $\mathrm{card}(F(\alpha)) = n + 2$.

The usefulness of the sample $F(\alpha)$ is based on the following lemma from [2].

Lemma 3.1 *Let* α *and* β *be patterns such that* $|\alpha| = |\beta|$ *and* $F(\alpha) \subseteq L_{NE}(\beta)$. *Then* $\alpha \leq_+ \beta$.

Lemma 3.2 *Let* α *and* β *be patterns such that* $|\alpha| = |\beta|$. *Then*

$$L_{NE}(\alpha) \subseteq L_{NE}(\beta) \text{ iff } \alpha \leq_+ \beta$$

Proof. If $L_{NE}(\alpha) \subseteq L_{NE}(\beta)$ then $F(\alpha) \subseteq L_{NE}(\beta)$ and Lemma 3.1 gives $\alpha \leq_+ \beta$. The implication in the other direction is immediate. □

Theorem 3.1 [2] *For any patterns* α *and* β:

$$L_{NE}(\alpha) = L_{NE}(\beta) \text{ iff } \alpha \equiv \beta$$

Proof. If $L_{NE}(\alpha) = L_{NE}(\beta)$ then necessarily $|\alpha| = |\beta|$ since the morphisms used for substituting the variables are non-erasing. Then Lemma 3.2 gives $\alpha \leq_+ \beta$ and $\beta \leq_+ \alpha$, and further by (1), $\alpha \equiv \beta$. The converse implication is immediate. □

The proof of Theorem 3.1 actually gives a linear time algorithm for testing language equivalence of NE-patterns. Consequently the following result may seem surprising.

Theorem 3.2 [18] *Let Z be either E or NE. Given a terminal alphabet Σ and two patterns $\alpha, \beta \in (\Sigma \cup V)^*$ it is undecidable whether or not*

$$L_{Z,\Sigma}(\alpha) \subseteq L_{Z,\Sigma}(\beta).$$

We omit the proof of Theorem 3.2 which is quite long and can be found in [18]. The construction uses a reduction from the problem of determining whether a nondeterministic 2-counter automaton has an accepting computation. The latter question is known to be undecidable [15].

We may ask why is the inclusion problem difficult. Recall that $\alpha \leq_+ \beta$ implies that $L_{NE}(\alpha) \subseteq L_{NE}(\beta)$. Also, from Lemma 3.2 we know that the converse holds in the special case when $|\alpha| = |\beta|$. However, the converse does not need to hold for patterns of different lengths.

Example 3.1 Let $\Sigma = \{0, 1\}$ and consider patterns $\alpha = 0x10xx1$ and $\beta = xxy$. Consider an arbitrary $w \in L_{NE}(\alpha)$, $w = 0u10uu1$ where $u \in \Sigma^+$. If $u = 0v$, we can write $w = 0{\cdot}0{\cdot}v100v0v1$ which is in $L_{NE}(\beta)$. Similarly, if $u = 1v$, we can write $w = 01v1{\cdot}01v1{\cdot}v1$ which again is in $L_{NE}(\beta)$. On the other hand, clearly $\alpha \not\leq_+ \beta$.

The above example shows that the inclusion $L_{NE}(\alpha) \subseteq L_{NE}(\beta)$ can hold for two patterns α and β that seemingly do not have any connection.

The inclusion problem is known to be decidable in certain special cases.

Theorem 3.3 [13, 18] *The inclusion problem is decidable for terminal-free E-patterns, (that is, patterns $\alpha, \beta \in V^+$).*

On the other hand, it should be noted that the decidability of the inclusion problem for terminal-free NE-patterns is still open.

The method of defining words by using a substitution for the variables has an obvious similarity to the way in which variables are used in rewriting systems. It turns out that the pattern language inclusion problem has an interesting connection to questions on rewriting systems. We present here only the basic ideas, for details of the connection between pattern languages and rewriting systems we refer the reader to [21, 22], and [4] is a general reference for rewriting systems.

A *word rewriting system with variables* (WRSV) is a finite set $\mathcal{R}$ of rewrite rules $u \to v$, where $u, v \in (V \cup \Sigma)^*$. Here Σ is an alphabet V is a finite set of variables. *Ground instances* (or simply, *instances*) of the left sides of the rules are obtained by subsituting strings of Σ^+ for the variables. In our terminology, a ground instance of $w \in (V \cup \Sigma)^*$ is any string $h(w)$ where $h \in S_{\Sigma,V}$ is non-erasing. A string $z \in \Sigma^*$ is said to be *reducible* by $\mathcal{R}$ if z contains contains a subword that is an instance of the left side of some rule in $\mathcal{R}$.

The *ground reducibility problem* for WRSV's [21, 22] asks whether all ground instances of a fixed string $w \in (V \cup \Sigma)^*$ are reducible by a given WRSV $\mathcal{R}$.

Theorem 3.4 [21] *Ground reducibility is undecidable for WRSV's.*

The result of Theorem 3.4 appears quite similar to the inclusion problem for NE-pattern languages, however, the questions are not precisely the same since inclusion of the language of an NE-pattern α in the language defined by pattern β means that any instance of α has to be an instance of β (as opposed to just having a subword that is an instance of β). However, this distinction can be overcome by noting that if $a \in \Sigma$ and variables x, y do not appear in the pattern β, then $L_{NE}(a\alpha a) \subseteq L_{NE}(x\beta y)$ if and only if all instances of α are reducible with a rule having β as the left side. Thus, Theorem 3.2 implies that ground reducibility is undecidable even for WRSV's having a single rule.

Finally it may be noted that the ground reducibility problem for term rewriting systems [4] is decidable. When viewed as term rewriting systems, WRSV's are rewriting systems over a signature consisting of one binary associative operator (concatenation) and finitely many constant symbols, and the presence of associative functions makes the problem more complex.

4 Equivalence of E-pattern Languages

We have seen that two NE-patterns are language equivalent if and only if they are identical up to a renaming of variables. However, no such characterization is known for E-patterns and it remains an open question whether E-pattern equivalence is decidable. In order to deal with E-pattern equivalence we introduce the following definition.

Definition 4.1 *Let $\alpha \in (\Sigma \cup V)^+$ be a pattern. By the* standard representation *of α we mean the decomposition*

$$\alpha = \alpha_0 u_1 \alpha_1 u_2 \cdots \alpha_{m-1} u_m \alpha_m \tag{2}$$

where $\alpha_0, \alpha_m \in V^*$, $\alpha_i \in V^+$ for $1 \leq i \leq m-1$, $u_j \in \Sigma^+$, $1 \leq j \leq m$.

The terminal segment *of the standard representation (2) is the m-tuple of terminal words $(u_1, u_2, \ldots, u_m)$. Patterns α and β are said to be* similar *if the standard representations of α and β have the same terminal segment.*

The notion of similarity is well-defined because the standard representation of a pattern is unique. The following result appears in [13] (case 1.) and in [17, 18].

Theorem 4.1 *Let $\alpha, \beta \in (\Sigma \cup V)^+$ and assume that $L_{E,\Sigma}(\alpha) = L_{E,\Sigma}(\beta)$.*

1. *If* $\mathrm{card}(\Sigma) \geq 3$ *then α and β are necessarily similar.*

2. *Assume that* $\mathrm{card}(\Sigma) \geq 4$ *and let*

$$\alpha = \alpha_0 u_1 \alpha_1 u_2 \cdots \alpha_{m-1} u_m \alpha_m, \quad \beta = \beta_0 u_1 \beta_1 u_2 \cdots \beta_{m-1} u_m \beta_m.$$

(Note that by 1. above we know that α and β have to be similar.)

Then we have further $L_{E,\Sigma}(\alpha_i) = L_{E,\Sigma}(\beta_i)$, $0 \leq i \leq m$.

However, conversely if α and β are similar and satisfy $L_{E,\Sigma}(\alpha_i) = L_{E,\Sigma}(\beta_i)$, $0 \leq i \leq m$ (using the above notations), this is not sufficient to guarantee that $L_{E,\Sigma}(\alpha) = L_{E,\Sigma}(\beta)$ [18]. Also, the result of Theorem 4.1 does not hold if the terminal alphabet has only two letters, as is seen by the below example.

Example 4.1 *Let $\Sigma = \{0, 1\}$ and choose $\alpha = x01y0z$, $\beta = x0y10z$. It is easy to verify (both languages are, in fact, regular) that $L_{E,\Sigma}(\alpha) = L_{E,\Sigma}(\beta)$, however α and β are not similar.*

We can prove that for similar E-patterns the equivalence problem, and even the inclusion problem, is decidable provided that the terminal alphabet contains two letters not occurring in the patterns.

Definition 4.2 *Let $V = \{x_1, \ldots, x_n\}$ be a set of variables and Σ an alphabet with* $\mathrm{card}(\Sigma) \geq 2$. *For every pair of distinct letters $a, b \in \Sigma$ and an integer $k > 0$ we define a morphism $\tau_{k,a,b} : V \longrightarrow \{a, b\}^*$ by*

$$\tau_{k,a,b}(x_i) = ab^{k \cdot i+1} aab^{k \cdot i+2} a \cdots ab^{k \cdot (i+1)} a, \quad 1 \leq i \leq n.$$

In the above definition and the following results in this section we restrict the set of variables V to be finite. This does not cause loss of generality since V can always be chosen large enough to contain all variables occurring in the patterns under consideration.

Theorem 4.2 [30] *Let $\alpha, \beta \in (\Sigma \cup V)^+$ be similar patterns. If Σ contains two distinct letters a and b not occurring in α and β, then the following statements are equivalent:*

1. $L_{E,\Sigma}(\alpha) \subseteq L_{E,\Sigma}(\beta)$
2. $\tau_{|\beta|,a,b}(\alpha) \in L_{E,\Sigma}(\beta)$
3. *There exists a morphism* $h : \mathrm{var}(\beta) \longrightarrow \mathrm{var}(\alpha)^*$ *such that* $h(\beta) = \alpha$.

Corollary 4.1 [30] *For two similar patterns α, β and an alphabet Σ containing two distinct terminals not appearing in α and β, it is decidable whether or not $L_{E,\Sigma} \subseteq L_{E,\Sigma}(\beta)$.*

Theorems 4.1 and 4.2 give also the following:

Corollary 4.2 [30] *If* $\mathrm{card}(\Sigma) \geq 3$, *the equivalence problem for E-pattern languages is decidable whenever the underlying alphabet Σ contains two terminals that do not occur in the patterns.*

Note that inclusion is decidable for terminal-free E-patterns (Theorem 3.3). In spite of this, in Corollary 4.2 we need the assumption $\mathrm{card}(\Sigma) \geq 3$, since in the case $\mathrm{card}(\Sigma) = 2$ E-pattern language equivalence does not imply similarity of the patterns, as observed in Example 4.1.

The above results seem to indicate that E-pattern equivalence becomes "easier" if we introduce new letters to the terminal alphabet. However, it remains an open question whether E-pattern equivalence is preserved under alphabet extensions when the original alphabet has at least three symbols [30], that is, does the equivalence

$$L_{E,\Sigma}(\alpha) = L_{E,\Sigma}(\beta) \quad \Leftrightarrow \quad L_{E,\Sigma'}(\alpha) = L_{E,\Sigma'}(\beta)$$

hold for $\mathrm{card}(\Sigma) \geq 3$, $\alpha, \beta \in (\Sigma \cup V)^*$, and $\Sigma' = \Sigma \cup \{a\}$, $a \notin \Sigma$. We know from Example 4.1 that E-pattern equivalence does not have the above extension property when $\mathrm{card}(\Sigma) = 2$.

The fact that an E-pattern may contain many superfluous variables (that are not "needed" to define the language) seems to be a major cause of difficulty in deciding whether two E-patterns are equivalent. For example, if a terminal-free E-pattern α contains some variable that occurs exactly once, then α is equivalent to the pattern x. Thus, very "different looking" E-patterns may be equivalent. Ohlebusch and Ukkonen have introduced a general variable elimination technique that is defined by a computable reduction relation and can be used to solve E-pattern equivalence in certain special cases. For details we refer the reader to [30].

5 Ambiguity

The classical language theoretic notions of unambiguity and degrees of ambiguity have natural counterparts in languages defined by patterns. Consider the pattern $\alpha = xyx$ and $w = a^3ba^3$. There exist three different non-erasing homomorphisms h such that $w = h(\alpha)$, that is, the degree of *ambiguity* of w with respect to the NE-pattern α is three. Below we present the main definitions and some results without proofs concerning ambiguity in patterns. The reader is referred to [25, 26] for more information.

Let α be a pattern and w a terminal string. We denote by $S_Z(\alpha, w, \Sigma)$, $Z \in \{E, NE\}$, the set of morphisms (in case $Z = NE$, non-erasing morphisms) stable for terminals with domain $\mathrm{var}(\alpha)$ such that $h(\alpha) = w$. Note that for the purposes of ambiguity, morphisms that agree on variables occurring in the pattern should be considered identical.

Definition 5.1 *Let α be a Z-pattern, $Z \in \{E, NE\}$. The* degree of ambiguity *of α is $k \geq 1$ if*

1. $\mathrm{card}(S_Z(\alpha, w, \Sigma)) \leq k$ *for all* $w \in \Sigma^*$, *and*

2. $\mathrm{card}(S_Z(\alpha, w, \Sigma)) = k$ *for some* $w \in \Sigma^*$.

If there is no $k \geq 1$ as above, then the degree of ambiguity of α is infinite.

A Z-pattern language L is ambiguous of degree $k \geq 1$ if $L = L_Z(\alpha)$ for some pattern α with ambiguity k and L is not denoted by any Z-pattern of ambiguity degree less than k. If no k as above exists we say that the ambiguity degree of L is infinite. If L has ambiguity degree $k = 1$ it is said to be *unambiguous.* Otherwise L is said to be *inherently ambiguous.*

From Theorem 3.1 we get immediately the following.

Theorem 5.1 *The degree of ambiguity of an NE-pattern α equals the degree of ambiguity of the language $L_{NE}(\alpha)$.*

Note that E-patterns do not have the same property. For example, Σ^* is denoted by the E-pattern x. Hence Σ^* is unambiguous, however, it is denoted also by the E-pattern xy that has infinite ambiguity degree.

Theorem 5.2 [25] *Given a Z-pattern α, $Z \in \{E, NE\}$, and a natural number k, we can effectively decide whether α is ambiguous of degree at least k.*

Consequently, we can effectively decide whether a pattern has ambiguity equal to given $k \geq 1$, has ambiguity strictly less than given k, or is unambiguous. However, the decidability of the question whether the degree of ambiguity of a pattern is infinite remains open [25]. The proof of Theorem 5.2 relies on Makanin's algorithm that can be extended to solve finite systems of equations and inequations [8]. The condition that the degree of ambiguity of a pattern is infinite cannot, at least not straightforwardly, be expressed in this way.

Theorems 5.2 and 5.1 can be combined to decide algorithmically whether an NE-pattern *language* is ambiguous of any fixed degree k. The corresponding question remains open for E-pattern languages. Here the situation is essentially more complicated since even the decidability of E-pattern equivalence remains open [30].

While it is easy to construct patterns of ambiguity degree 1 or of ambiguity degree infinite, it is generally difficult to prove that a given pattern has a finite degree of ambiguity greater than one. It is shown in [25, 26] that the pattern $xabxbcayabcy$ has ambiguity exactly 2 (as an NE- or E-pattern). Following our convention for the notation, here x, y are variables and a, b, c are terminals. The "best" known example of a pattern that is ambiguous of degree 3 has length 324 and the shortest word that has three different decompositions according to this pattern has length 1018 [25, 26]. More generally we have the following result.

Theorem 5.3 [25, 26] *For any $m \geq 0$ and $n \geq 0$, a Z-pattern, $Z \in \{E, NE\}$, with the degree of ambiguity exactly $2^m 3^n$ can be effectively constructed.*

The main open question is whether for all $k \geq 1$ there is a pattern α_k that is ambiguous of degree exactly k.

6 Related Topics

Here we briefly mention and give pointers to further topics related to patterns that have been omitted above due to space limitations. A general overview of pattern languages and further references can be found also in [28, 32].

Inductive inference and computational learning theory

Applications in learning theory were the original motivation for studying pattern languages [2, 3]. As we briefly discussed in Section 2, patterns are a very natural model for representing a sample set of strings that is, for instance, growing through a learning process. As examples of non-regular languages that can be learned in the limit from positive data, pattern languages have gained much interest in the machine learning community. Pattern languages can be learned in polynomial time if the set of variables is fixed or if we drop the requirement of consistency [23]. Inductive inference results for patterns using combinatorics on words type techniques are presented in [24]. Recent references on learning pattern languages are [12, 14, 31].

Unavoidable patterns

Here we have used patterns to define languages. In some situations it is more relevant to ask questions like: does there exist arbitrarily long words (that is, an infinite word) over Σ that avoids a given pattern? For example, the pattern xx is avoidable if Σ has three letters but unavoidable on a binary alphabet. This direction of research was initiated by A. Thue almost a hundred years ago and recent references can be found in [6, 7].

Pattern systems

Generative devices based on patterns have been considered in [9, 27] and a more general model of *pattern systems* is introduced in [29]. In a pattern system we are given a finite set of terminal words for each variable, and variables occurring in patterns are uniformly replaced by these terminal words, yielding a new set of terminal words associated with each variable. The above process is iterated and, furthermore, the process can be done *synchronously* or *non-synchronously.* Pattern systems define a considerably larger language family than ordinary E- or NE-patterns. Results on the comparison of languages defined by pattern systems with known language families can be found in [11]. There remain many interesting open questions in the area of pattern systems.

Acknowledgement

I thank Alexander Okhotin for carefully reading the paper and for useful suggestions.

References

[1] A.V. Aho, Algorithms for finding patterns in strings. In: *Handbook of Theoretical Computer Science* Vol. A, (J. van Leeuwen, Ed.), Elsevier 1990, pp. 255–300.

[2] D. Angluin, Finding patterns common to a set of strings. *Journal of Computer and System Sciences* 21 (1980) 46–62.

[3] D. Angluin, Inductive inference of formal languages from positive data. *Information and Control* 45 (1980) 117–135.

[4] F. Baader and T. Nipkow, *Term Rewriting and All That.* Cambridge University Press, 1998.

[5] C. Câmpeanu, K. Salomaa and S. Yu, Regex and extended regex. *Proc. of CIAA'02,* Springer Lect. Notes Comput. Sci., to appear.

[6] J. Cassaigne, Unavoidable patterns. In: *Algebraic Combinatorics on Words, (M. Lothaire),* Cambridge University Press, 2002.

[7] C. Choffrut and J. Karhumäki, Combinatorics on words. In: Handbook of Formal Languages, Vol. I, (G. Rozenberg, A. Salomaa, Eds.), Springer-Verlag 1997, pp. 329–438.

[8] K. Culik II and J. Karhumäki, Systems of equations over a free monoid and Ehrenfeucht's conjecture. *Discrete Mathematics* 43 (1983) 139–153.

[9] J. Dassow, Gh. Păun and A. Salomaa, Grammars based on patterns. *International Journal of Foundations of Computer Science* 4 (1993) 1–14.

[10] G. Della Penna, B. Intrigila, E. Tronci and M. Venturini Zilli, Synchonized regular expressions. *Acta Informatica* 39 (2003) 31–70.

[11] J. Duske and R. Parchmann, Non-synchronized pattern languages are IO-macro languages. *Journal of Automata, Languages and Combinatorics* 2 (1997) 143–150.

[12] T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger and T. Zeugmann, Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science* 261 (2001) 119–156.

[13] G. Filè, The relation of two patterns with comparable languages. *Proc. STACS'88,* Lect. Notes Comput. Sci. 294, 1988, pp. 184–192.

[14] S. Goldman and S. Kwek, On learning unions of pattern languages and tree patterns in the mistake bound model. *Theoretical Computer Science* 288 (2002) 237–254.

[15] O. Ibarra, Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM* 25 (1978) 116–133.

[16] K. Jantke, Polynomial time inference of general pattern languages. *Proc. STACS'84,* Lect. Notes Comput. Sci. 166, 1984, pp. 314–325.

[17] T. Jiang, E. Kinber, A. Salomaa, K. Salomaa and S. Yu, Pattern languages with and without erasing. *Intern. J. Computer Math.* 50 (1994) 147–163.

[18] T. Jiang, A. Salomaa, K. Salomaa and S. Yu, Decision problems for patterns. *Journal of Computer and System Sciences* 50 (1995) 53–63.

[19] L. Kari, A. Mateescu, G. Păun and A. Salomaa, Multi-pattern languages. Theoretical Computer Science 141 (1995) 253–268.

[20] K.-I Ko and W.-G. Tzeng, Three Σ_2^P-complete problems in computational learning theory. *Computational Complexity* 1 (1991) 269–310.

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

[22] G. Kucherov and M. Rusinowitch, Patterns in words versus patterns in trees: a brief survey and new results. International Conference "Perspectives of Systems Informatics", July 1999, Novosibirsk, Russia. Lect. Notes Comput. Sci. 1755, Springer, pp. 280–293.

[23] S. Lange and R. Wiehagen, Polynomial time inference of arbitrary pattern languages. *New Generation Computing* 8 (1991) 361–370.

[24] A. Marron and K.-I Ko, Identification of pattern languages from examples and queries. *Information and Computation* 74 (1987) 91–112.

[25] A. Mateescu and A. Salomaa, Nondeterminism in patterns. *Proc. STACS'94,* Lect. Notes Comput. Sci. 775, Springer, 1994, pp. 661–668.

[26] A. Mateescu and A. Salomaa, Finite degrees of ambiguity in pattern languages. *Theoretical Informatics and Applications* 28 (1994) 233-253.

[27] V. Mitrana, Iterated pattern languages. *Journal of Automata, Languages and Combinatorics* 1 (1996) 305–311.

[28] V. Mitrana, Patterns and languages: An overview. *Grammars* 2 (1999) 149–173.

[29] V. Mitrana, G. Păun, G. Rozenberg and A. Salomaa, Pattern systems. *Theoretical Computer Science* 154 (1996) 183–201.

[30] E. Ohlebusch and E. Ukkonen, On the equivalence problem for E-pattern languages. *Theoretical Computer Science* 186 (1997) 231–248.

[31] R. Reischuk and T. Zeugmann, An average-case optimal one-variable pattern language learner. *Journal of Computer and System Sciences* 60 (2000) 302–335.

[32] A. Salomaa, Patterns and Return to patterns. *Bull. EATCS* 54 (1994) 194–206 and *Bull. EATCS* 55 (1995) 144–157.