Learning One-Variable Pattern Languages in Linear Average Time*

RÜDIGER REISCHUK
Med. Universität zu Lübeck
Institut für Theoretische Informatik
Wallstraße 40
23560 Lübeck, Germany
reischuk@informatik.mu-luebeck.de

THOMAS ZEUGMANN
Department of Informatics
Kyushu University
Kasuga 816
Japan
thomas@i.kyushu-u.ac.jp

Abstract. A new algorithm for learning 1-variable pattern languages is proposed and analyzed with respect to its average-case behavior. We consider the total learning time that takes into account all operations till an algorithm has converged to a correct hypothesis. For the expectation it is shown that for almost all meaningful distributions defining how the pattern variable is replaced by a string to generate random samples of the target pattern language this algorithm converges within a constant number of rounds with a *total learning time* that is *linear* in the pattern length. Thus, the algorithm is average-case optimal in a strong sense.

Though 1-variable pattern languages cannot be inferred finitely, our approach can also be considered as probabilistic finite learning with high confidence.

1. Introduction

The formal definition of patterns and pattern languages goes back to Angluin [1]. Since then, pattern languages and variations thereof have widely been investigated (cf., e.g., [11, 12, 14].

As far as learning theory is concerned, pattern languages are a prominent example of non-regular languages that can be learned in the limit from positive data. Gold [5] has introduced the corresponding learning model. Let L be any language; then a *text* for L is any infinite sequence of strings containing eventually all strings of L, and nothing else. The information given to the learner are successively growing initial segments of a text. Processing these segments, the learner has to output *hypotheses* about L. The hypotheses are chosen from a prespecified set called *hypothesis space*. The sequence of hypotheses has to *converge* to a correct description of the target language.

Looking at applications of limit learners, efficiency becomes a central issue. But defining an appropriate measure of efficiency for learning in the limit is a difficult problem (cf. [9]). Various authors have studied the efficiency of learning in terms of the update time needed for computing a new single hypothesis. But what counts in applications is the overall time needed by a learner until convergence, i.e., the *total learning time.* Since the total learning time is *unbounded* in the worst-case, we study the *expected* total learning time. Next, we shortly summarize what has been known in this regard.

Angluin [1] provides a learner for the class of all pattern languages that is based on the notion of *descriptive patterns.* Since no efficient algorithm is known for computing descriptive patterns, and finding a descriptive pattern of *maximum* length is $\mathcal{NP}$ -hard, its update time is practically *infeasible.*

Therefore, one has considered restricted versions of pattern language learning in which the number k of different variables is fixed, in particular the case $k = 1$. Angluin [1] gives a learner for one-variable pattern languages with update time $O(\ell^4 \log \ell)$, where ℓ is the sum of the length of all examples seen so far. Nothing is known concerning the expected total learning time of her algorithm.

Erlebach *et al.* [3, 4] presented a one-variable

* This work was performed while the first author was visiting the Department of Informatics at Kyushu University and was supported by the Japan Society for the Promotion of Science under Grant JSPS 29716102.

pattern learner achieving an average total learning time $O(|\pi|^2 \log |\pi|)$, where $|\pi|$ is the length of the target pattern. This result is also based on finding descriptive patterns quickly. This approach has the advantage that the descriptiveness of every hypothesis output is guaranteed, but it may have the disadvantage of preventing the learner to achieve a better expected total learning time. Thus, we ask whether there is a one-variable pattern language learner achieving a subquadratic expected total learning time. Clearly, the best one can get is a linear average total learning time. If this is really possible, then such a learner seems to be more appropriate for potential application than previously obtained ones, even if there are no guaranteed properties concerning the intermediately calculated hypotheses. Such a learner would have already finished his learning task with high probability before any of the known learner has computed a single guess.

What we like to present in this paper is such a one-variable pattern learner. Moreover, we prove that our learner achieves an expected linear total learning time for a very large class of distributions with respect to which the input examples are drawn.

2. Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of all natural numbers, and let $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. For all real numbers y we define $\lfloor y \rfloor$, the *floor function*, to be the greatest integer less than or equal to y. Let Σ be an alphabet with $s := |\Sigma| \geq 2$. By Σ^* we denote the free monoid over Σ, and we set $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$, where ε is the empty string. Let x be a symbol with $x \notin \Sigma$. Every string over $(\Sigma \cup \{x\})^+$ is called a one-variable pattern. We refer to x as the pattern variable. Let *Pat* denote the set of all one-variable patterns. We write $\#(\pi, x)$ for the number of occurrences of the pattern variable x in π.

The length of a string $w \in \Sigma^*$ and of a pattern $\pi \in Pat$ is denoted by $|w|$ and $|\pi|$, respectively. Let w be a string with $\ell = |w| \geq 1$, and let $i \in \{1, \ldots, \ell\}$; we use $w[i]$ and $w[-i]$ to denote the i-th symbol in w counted from left to right and right to left, respectively, i.e.,

$$\begin{aligned} w &= w[1]\, w[2]\, \ldots\, w[\ell - 1]\, w[\ell] \\ &= w[-\ell]\, w[-\ell + 1]\, \ldots\, w[-2]\, w[-1]. \end{aligned}$$

For $1 \leq i \leq j \leq \ell$ we denote the substring $w[i] \ldots w[j]$ of w by $w[i \ldots j]$. Let $\pi \in Pat$ and $u \in \Sigma^+$; we use $\pi[x/u]$ for the string $w \in \Sigma^+$ obtained by substituting all occurrences of x in π by u. The string u is called a *substitution*. For every $\pi \in Pat$ we define the *language generated by pattern* π by

$$L(\pi) \;:=\; \{y \in \Sigma^+ \mid \exists\, u \in \Sigma^+, \quad y = \pi[x/u]\}$$

For discussing our approach to learning all one-variable pattern languages we let

$$\pi \;=\; w_0 x^{\alpha_1} w_1 x^{\alpha_2} w_2 \ldots w_{m-1} x^{\alpha_m} w_m$$

be the target pattern throughout this paper. Here the α_i denotes positive integers (the multiplicity by which x appears in a row), and $w_i \in \Sigma^*$ the separating constant substrings, where for $1 \leq i < m$ the w_i are assumed to be nonempty.

The learning problem considered in this paper is exact learning in the limit from positive data. A sequence $(\psi_i)_{i \in \mathbb{N}^+}$ of patterns is said to *converge* to a pattern π if $\psi_i = \pi$ for all but finitely many i.

DEFINITION 1. Given a target pattern π, the learner gets a sequence of example strings $X_1, X_2, \ldots$ from $L(\pi)$. Having received X_g he has to compute as hypothesis a one-variable pattern ψ_g. The sequence of guesses $\psi_1, \psi_2, \ldots$ eventually has to converge to a pattern ψ such that $L(\psi) = L(\pi)$.

Note that in the case of one-variable pattern languages this implies that $\psi = \pi$. Some more remarks are mandatory here. Though our definition of learning resembles that one given in Gold [5], there is also a major difference. In [5] the sequence $(X_i)_{i \in \mathbb{N}^+}$ is required to fulfill $\{X_i \mid i \in \mathbb{N}^+\} = L(\pi)$. Nevertheless, this requirement will be hardly fulfilled. We therefore omit this assumption here. Instead, we only require the sequence $(X_i)_{i \in \mathbb{N}}$ to contain "enough" information to recognize the target pattern π. What is meant by "enough" will be made precise when discussing the set of all admissible distributions with respect to which the example sequences are allowed to be randomly drawn.

We continue with the complexity measure considered in this paper. The length of the pattern π to be learned is given by $n := n_w + n_x$ with $n_w := \sum |w_i|$ and $n_x := \sum \alpha_i$. This parameter will be considered as the size of problem instances, and the complexity analysis will be done with respect to this value n. We assume the same model of computation and the same representation of patterns as Angluin [1], i.e., in particular a random access machine that performs a reasonable menu of operations each in unit time on registers of length $O(\log n)$ bits, where n is the input length. The inputs are read via a serial input device, and reading a string of length n is assumed to require n steps.

In contrast to previous work [1, 6, 13, 15], we measure the efficiency of a learning algorithm by estimating the overall time taken by the learner until convergence. This time is referred to as the *total learning time.* We aim to determine the total learning time in dependence on the length of the target pattern. Of course, if examples are provided by an adversary the number of examples one has to see before being able to converge is unbounded in general. Thus analyzing the total learning time in such a worst-case setting will not yield much insight. But such a scenario is much too pessimistic for many applications, and therefore, one should consider the average-case behavior. Analyzing the expected total learning time of limit learners has been initialized by Zeugmann [16]. Average-case complexity in general depends very much on the distribution over the input space. We perform our analysis for a very large class of distributions. An optimal result of linear expected total learning is achieved by carefully analyzing the combinatorics of words generated by a one-variable pattern. This linear bound can even be shown to hold with high probability. Let

$$\mu \; : \; \Sigma^+ \to [0,1]$$

be the probability distribution specifying how given a pattern π the variable x is replaced to generate random examples $\pi[x/Z]$ from $L(\pi)$. Here $Z = Z_\mu$ is a random variable with distribution μ.

$$\texttt{Range}(Z) \;:=\; \{w \in \Sigma^+ \mid \mu(w) > 0\}$$

denotes the range of Z, i.e., the set of all substitution strings that may actually occur. From this we get a probability distribution

$$\mu_\pi \; : \; \Sigma^+ \to [0,1]$$

for the random strings generated by π based on μ. Let $X = X_{\pi,\mu}$ denote a random variable with distribution μ_π. The random examples are then generated according to X, thus the relation between X and Z is given by $X = w_0 \; Z^{\alpha_1} \; w_1 \; Z^{\alpha_2} \; w_2 \; \ldots \; w_{m-1} \; Z^{\alpha_m} \; w_m$. Note that μ is fixed, and in particular *independent* of the special target pattern to be learned.

What we consider in the following is a large class $\mathcal{D}$ of distributions μ that is defined by requiring only very simple properties. These properties basically exclude the case where only a small subset of all possible example strings occur and this subset does not provide enough information to reconstruct the pattern. We show that there exists an algorithm that efficiently learns every one-variable pattern on the average with respect to every distribution in $\mathcal{D}$.

By $\boldsymbol{E}[|\boldsymbol{Z}|]$ we denote the expectation of $|Z|$, i.e., the average length of a substitution. Then the expected length of an example string X for π is given by $E[|X|] = n_w + n_x \cdot E[|Z|] \leq n \cdot E[|Z|]$. Obviously, if one wants to analyze the bit complexity of a learning algorithm with respect to the pattern length n one has to assume that $E[|X|]$, and hence $E[|Z|]$, is finite, otherwise already the expected length of a single example will be infinite.

***Assumption* 1.** $E[|Z|] < \infty$.

Let $\mathcal{X} = X_1, X_2, X_3, \ldots$ denote a sequence of random examples that are independently drawn according to μ_π. Note that the learner, in general, does not have information about μ_π *a priori.* On the other hand, the average-case analysis of our learning algorithm presupposes information about the distribution μ. Thus, unlike the PAC-model, our framework is not completely distribution-free. Nevertheless, we aim to keep the information required about μ as small as possible. Finally, let

$$L(\pi,\mu) \ := \ \{y \in \Sigma^+ \mid \mu_\pi(y) > 0\}$$

be the language of all example strings that may actually occur.

3. Probabilistic Analysis of Substitutions

For obtaining most general results we would like to put as little constraints on the distribution μ as possible. Note that one cannot learn a target pattern if only example strings of a very restricted form occur. This will be in particular the case if $\texttt{Range}(Z)$ itself is contained in a nontrivial one-variable pattern language. For seeing this, suppose there exists a pattern $\phi \in Pat \setminus \{x\}$ such that $\texttt{Range}(Z) \subseteq L(\phi)$. Clearly, then the languages generated by $\pi = w_0 x^{u_0} w_1 x^{u_1} w_2 \dots w_{m-1} x^{u_m} w_m$ and $\pi' = w_0 \phi^{u_0} w_1 \phi^{u_1} w_2 \dots w_{m-1} \phi^{u_m} w_m$ cannot be distinguished, since $L(\pi,\mu) \subseteq L(\pi')$. Thus, even from an information theoretic point of view the learner has no chance to distinguish this case from the one where the pattern to be learned is actually π' and the examples are generated by the corresponding projection μ' of μ. Hence, such a problem instance (π,μ) should be regarded as the instance (π',μ'). To exclude this case, let us define

$$p_0 \ := \ \max_{\phi \text{ pattern}, \ |\phi|>1} \Pr[Z \in L(\phi)] \ .$$

and let us make

***Assumption* 2.** $\quad p_0 < 1$.

An alternative approach would be to consider the correctness of the learning hypotheses with respect to the distribution μ. The learner solves the learning problem if he converges to a pattern ψ for which $L(\psi,\mu) = L(\pi,\mu)$. This model is equivalent, but conceptually more involved and complicates the algorithm. Therefore we stick to the original definition. If $p_0 < 1$ then the following quantities

$$\begin{aligned} p_a \ &:= \ \max_{\sigma\in\Sigma} \Pr[Z[1] = \sigma] \ , \\ p_e \ &:= \ \max_{\sigma\in\Sigma} \Pr[Z[-1] = \sigma] \ , \end{aligned}$$

are smaller than 1, too. Next, define

$$p \ := \ \max\{p_a, p_e\} < 1 \ ,$$

and for sequences of substitutions $\mathcal{Z} = Z_1, Z_2, Z_3, \dots$ the event

$$\begin{aligned} F_g[\mathcal{Z}] \ &:= \ \big[\, (Z_1[1] = Z_2[1] = \dots = Z_g[1]) \\ &\quad \vee \ (Z_1[-1] = Z_2[-1] = \dots = Z_g[-1]) \, \big] \ . \end{aligned}$$

Then $\Pr[F_g] \ \le \ 2p^{\,g\,-\,1}$. Define $f(\mathcal{Z}) \ := \ \min\{g \mid \neg F_g[\mathcal{Z}]\}$.

LEMMA 1. *The expectation of* $f(\mathcal{Z})$ *can be bounded as* $E[f(\mathcal{Z})] \ \le \ 2/(1-p)$.

4. Symmetries of Strings

We now come to the main technical tool that will help us to detect the pattern variable and its replacements in sample strings, respectively.

DEFINITION 2. Let $y = y[1]y[2]\dots y[\ell] \in \Sigma^+$ be a string of length ℓ. If for some k with $1 \le k \le \ell/2$ the k-length prefix and suffix of y are identical, that is $y[1\dots k] = y[\ell-k+1 \ \dots \ell]$, we say that y has a ***k*-*symmetry*** $u = y[1\dots k]$ (or symmetry, for short).

A symmetry u of y is said to be the smallest symmetry if $|u| < |\hat{u}|$ for every symmetry $\hat{u}$ of y with $\hat{u} \neq u$.

DEFINITION 3. Let u be a symmetry of y and choose positive integers c, d maximal such that for some string v_0 $y = u^c \, v_0 \, u^d$, i.e., u is neither a prefix nor a suffix of v_0. This includes the special case $v_0 = \varepsilon$. In this case, since c and d are not uniquely determined, we choose $c \ge d$ such that their difference is at most 1. This unique representation of a string y will be called ***factorization of y with respect to u*** or simply ***u*-*factorization***, and u the ***base*** of this factorization.

If all occurrences of u are factored out including also possible ones in v_0 one gets a representation $y = u^{c_0} \, v_1 \, u^{c_1} \, v_2 \, \dots v_r \, u^{c_r}$ with positive integers c_i ($c_0 = c$, $c_r = d$) and strings v_i that do not contain u as substring. This will be called a ***complete u*-*factorization*** of y.

Of particular interest for a string y will be its symmetry of minimal length, denoted by

$mls(y)$, which gives rise to the ***minimal factorization*** of y. For technical reasons, if y does not have a symmetry then we set $mls(y) := |y| + 1$. Let ***$sym(y)$*** denote the number of all different symmetries of y.

The following properties will be important for the learning algorithm described later.

LEMMA 2. *Let $k \in \mathbb{N}^+$ and let $u, y \in \Sigma^+$ be any two strings such that u is a k-symmetry of y. Then we have*

(1) *u is a smallest symmetry of y iff u itself has no symmetry.*

(2) *If y has the factorization $y = u^c\ v_0\ u^d$ then it also has k'-symmetries for $k' = 2k, 3k, \ldots, \min\{c,d\}\ k$.*

(3) *If $u^c\ v_0\ u^d$ is the minimal factorization of y then, for all $k' \in \{1, \ldots, \max\{c,d\} mls(y)\}$, y does not have other k'-symmetries.*

(4) *$sym(y) \leq |y|\ /\ 2\ mls(y)$.*

Assertion (4) of the latter lemma directly implies the simple bound $sym(y) \leq |y|/2$, which in most cases, however, is far too large.

Now, we consider the expected number of symmetries. To motivate our Assumption 3, we first take a look at the length uniform case.

LEMMA 3. *In the length uniform case*

$$E[sym(Z)] \leq s/(2(s-1)^2)\ .$$

Thus, in this case the number of symmetries only depends on the size s of the alphabet. Let us now estimate the total length of all factorizations of a string y, which can be bounded by $|y| \cdot sym(y)$. For the length uniform case,

$$E[|Z| \cdot sym(Z)] \leq E[|Z|] \cdot E[sym(Z)]\ .$$

can be shown, but for arbitrary distributions, we have to require

Assumption 3. $E[|Z| \cdot sym(Z)] < \infty$.

Remember that we already had to assume that $E[|Z|]$ is finite. Trivially, the expectation of $|Z| \cdot sym(Z)$ is guaranteed to be finite if $E[|Z|^2] < \infty$, that means the variance of $|Z|$ is finite, but in general weaker conditions suffice.

If $0 < E[|Z| \cdot sym(Z)] < \infty$ then also $0 < E[sym(Z)] < \infty$. Thus we can find a constant c such that

$$E[|Z| \cdot sym(Z)] \leq c \cdot E[|Z|] \cdot E[sym(Z)] = O(1).$$

Symmetries and factorizations should be computed fast; we thus show:

LEMMA 4. *The minimal symmetry of a string y can be found in $O(|y|)$ operations. Given the minimal symmetry, all further symmetries can be generated in linear time. From a symmetry, the corresponding factorization can be computed in linear time as well.*

Let Σ^+_{sym} denote the set of all strings in Σ^+ that possess a symmetry and let

$$p_{\mathrm{sym}} := \Pr[Z \in \Sigma^+_{sym}]\ .$$

We require that the distribution is not restricted to substitutions with symmetries – with positive probability also nonsymmetric substitutions should occur.

Assumption 4. $p_{\mathrm{sym}} < 1$.

Now consider the event

$$Q_g[\mathcal{Z}] := [\{Z_1, \ldots, Z_g\} \in \Sigma^+_{sym}]\ .$$

$Q_g[\mathcal{Z}]$ means that among the first g substitutions all have a symmetry. Obviously,

$$\Pr[Q_g[\mathcal{Z}]] \leq p_{\mathrm{sym}}^g\ .$$

Define $q(\mathcal{Z}) := \min\{g \mid \neg Q_g[\mathcal{Z}]\}$. Similarly to Lemma 1, one can show

LEMMA 5. $E[q(\mathcal{Z})] \leq 1/(1 - p_{\mathrm{sym}})$.

5. Basic Subroutines: Factorizations - and Compatibility

For a subset A of Σ^* let $\mathtt{PRE}(A)$ and $\mathtt{SUF}(A)$ denote the maximal common prefix and suffix of all strings in A, respectively. Furthermore, let $m_{\mathrm{pre}}(A)$ and $m_{\mathrm{suf}}(A)$ be their lengths. The first goal of the algorithm is to recognize the prefix w_0 and suffix w_m before the first and last occurrence of the variable x, respectively, in the pattern π. In order to avoid confusion, x will be called the *pattern variable*, where *variable* simply refers to any data variable used by the learning algorithm.

The current information about the prefix and suffix is stored in the variables PRE and SUF. The remaining pattern learning is done with respect to the current value of these variables. If the algorithm sees a new string X such that $\mathrm{PRE}(\{X, \mathrm{PRE}\}) \neq \mathrm{PRE}$ or $\mathrm{SUF}(\{X, \mathrm{SUF}\}) \neq \mathrm{SUF}$ then these variables will be updated. We will call this *the begin of a new phase.*

DEFINITION 4. For a string $Y \in \Sigma^+$ a (PRE, SUF)*-**factorization*** is defined as follows. Y has to start with prefix PRE and end with suffix SUF. For the remaining middle part Y' we select a symmetry u_1. This means Y can be written as $Y = \mathrm{PRE}\, u_1^{c_1}\, v_1\, u_1^{d_1}\, \mathrm{SUF}$ for some strings u_1, v_1 and $c_1, d_1 \in \mathbb{N}^+$.

If such a representation is not possible for a given pair (PRE, SUF) then Y is said to have no (PRE, SUF)-factorization.

Moreover, Y' may have other symmetries $u_2, u_3, \ldots$ giving rise to factorizations $Y = \mathrm{PRE}\, u_i^{c_i}\, v_i\, u_i^{d_i}\, \mathrm{SUF}$ for $c_i, d_i \in \mathbb{N}^+$. For simplicity, we assume that the symmetries u_i are ordered by increasing length, in particular u_1 always denotes the minimal symmetry with corresponding minimal factorization.

LEMMA 6. *Let* $Y = \mathrm{PRE}\, u_1^{c_1}\, v_1\, u_1^{d_1}\, \mathrm{SUF}$ *be the minimal* (PRE, SUF)*-factorization of* Y. *Then, for every string* $\tilde{Y}$ *of the form* $\tilde{Y} = \mathrm{PRE}\ u_1\ \tilde{v}\ u_1\ \mathrm{SUF}$ *for some string* $\tilde{v}$, *the minimal* (PRE, SUF)*-factorization of* $\tilde{Y}$ *is based on* u_1, *too.*

Though the following lemma is easily verified, it is important to establish the correctness of our learner presented below.

LEMMA 7. *Let* $\pi = w_0\, x\, v\, w_m$ *be any pattern with* $\#(\pi, x) \geq 2$, *let* $u \in \Sigma^+$, *and let* $Y = \pi[x/u]$. *Then* Y *has a* (w_0, w_m)*-factorization with base* u *and its minimal* (w_0, w_m)*-factorization is based on the minimal symmetry* u_1 *of* u.

The results of Lemma 4 directly translate to

LEMMA 8. *The minimal base for a* (PRE, SUF) *- factorization of a string* Y *can be computed in time* $O(|Y|)$. *All additional bases can be found in linear time. Given a base, the corresponding* (PRE, SUF) *-factorization can be computed in linear time as well.*

DEFINITION 5. Two strings $Y, \tilde{Y}$ are said to be ***directly compatible*** with respect to a given pair (PRE, SUF) if from their minimal (PRE, SUF)-factorizations a single pattern $\psi = \psi(Y, \tilde{Y})$ can be derived from which both strings can be generated. More precisely, it has to hold:

$Y = \mathrm{PRE}\, u_1^{c_1}\, v_1\, u_1^{d_1}\, \mathrm{SUF}$ and

$\tilde{Y} = \mathrm{PRE}\, \tilde{u}_1^{\tilde{c}_1}\, \tilde{v}_1\, \tilde{u}_1^{\tilde{d}_1}\, \mathrm{SUF}$, and for $Y_{\mathrm{mid}} := u_1^{c_1 - 1}\, v_1\, u_1^{d_1 - 1}$ and $\tilde{Y}_{\mathrm{mid}} := \tilde{u}_1^{\tilde{c}_1 - 1}\, \tilde{v}_1\, \tilde{u}_1^{\tilde{d}_1 - 1}$ every occurrence of u_1 in Y_{mid} – including further ones in v_1 – is matched in $\tilde{Y}_{\mathrm{mid}}$ either by an occurrence of $\tilde{u}_1$ (which indicates that at this place π has a pattern variable) or by u_1 itself (indicating that the constant substring u_1 occurs in π). In all the remaining positions Y_{mid} and $\tilde{Y}_{\mathrm{mid}}$ have to agree.

We extend this compatibility notion to pairs consisting of a string Y and a pattern π. Y is directly compatible to π with respect to (PRE, SUF) if for the minimal symmetry u_1 of the (PRE, SUF)-factorization of Y holds $\pi[x/u_1] = Y$.

The following lemma is easily verified.

LEMMA 9. *Assume that* (PRE, SUF) $= (w_0, w_m)$ *has the correct value. If a string* Y *is generated from* π *by substituting the pattern variable by a nonsymmetric string* u *then the string* u_1 *on which its minimal* (PRE, SUF)*-factorization is based equals* u. *Thus,* Y *is directly compatible to* π.

If one of the substitutions $u, \tilde{u}$ for $Y = \pi[x/u]$, resp. $\tilde{Y} = \pi[x/\tilde{u}]$ is a prefix of the other, let us say $\tilde{u} = u\, u'$ for some nonempty string u' then there may be an ambiguity if $u\, u'$ appears as a constant substring in Y_{mid}. If this is not followed by another occurrence of u' it can easily be detected. In general, if $u\ u'$ is a constant in π then the number of occurrences following this substring will be the same in the corresponding positions in Y_{mid} and $\tilde{Y}_{\mathrm{mid}}$, otherwise it has to be one more in $\tilde{Y}$.

Using this observation it is easy to see that even in such a case testing of direct compatibility is easy.

LEMMA 10. *Let the minimal factorizations of two strings $Y, \tilde{Y}$ be given. Then by a single joint scan one can check whether they are directly compatible, and if yes construct their common pattern $\psi(Y, \tilde{Y})$. The scan can be performed in $O(|Y| + |\tilde{Y}|)$ bit operations.*
Moreover, for a pattern π it can be checked in time $O(|Y| + |\pi|)$ whether Y is directly compatible to π.

The extra effort in the degenerated case of u being a prefix of $\tilde{u}$ can be omitted if in this case the pattern matching is done from right to left since the procedure is completely symmetric. This will only fail if u is both prefix and suffix of $\tilde{u}$, implying that $\tilde{u} = u\, u'\, u$. But this means that $\tilde{u}$ has a symmetry and thus cannot derive from a minimal factorization of $\tilde{Y}$.

DEFINITION 6. A string Y is ***downwards compatible*** to a string $\tilde{Y}$ with respect to a given pair (PRE, SUF) if for some $\kappa \geq 1$, from the minimal (PRE, SUF)-factorization of Y and the κ-th (PRE, SUF)-factorization of $\tilde{Y}$ a single pattern $\psi = \psi(Y, \tilde{Y}, \kappa)$ can be derived from which both strings can be generated. We also say that $\tilde{Y}$ is ***upwards compatible*** to Y.

Again, these notions are extended to pairs consisting of a string and a pattern.

LEMMA 11. *Assume* $(\text{PRE}, \text{SUF}) = (w_0, w_m)$ *having the correct value. Let $Y = \pi[x/u]$ for a nonsymmetric string u. Any other string $\tilde{Y}$ in $L(\pi)$ obtained by substituting the pattern variable by a string $\tilde{u}$ for which u is not a symmetry is upwards compatible to Y with respect to* (PRE, SUF).
The pattern $\psi(Y, \tilde{Y})$ equals the pattern π to be learned.

Given the (PRE, SUF)*-factorization of both strings, $\psi(Y, \tilde{Y})$ can be constructed in time at most $O((1 + sym(\tilde{Y})) \cdot (|Y| + |\tilde{Y}|))$, where $sym(\tilde{Y}) := sym(\tilde{u})$ denotes the number of symmetries of the string $\tilde{Y}$ that generates $\tilde{Y}$.*
Furthermore, given a pattern ψ and the factorization of a string Y it can be checked in time $O(|Y| + |\psi|)$ whether Y is upwards compatible to ψ. For Y, downwards compatibility to ψ can be checked and $\psi(Y, \psi, \cdot)$ can be constructed in linear time, too.

Note that one cannot decide whether a string Y was generated by substitution with a nonsymmetric string by counting the number of its factorizations – which is likely to be one. However, there are rare cases with more factorization than the one induced by the substitution – for example, if α_1 and α_m have a common nontrivial divisor or even if $\alpha_1 = \alpha_m = 1$, but by chance $w_1 = v\ u\ v'$ and $w_{m-1} = v''\ u\ v$ for some arbitrary strings v, v', v''.

6. The Algorithm

The learner may not store all sample strings he has seen so far. Therefore let $A = A_g = A_g(\mathcal{X})$ denote the set of examples he remembers after having got the first g samples of the random sequence $\mathcal{X} = X_1, X_2, \ldots$, and, similarly, let PRE_g and SUF_g be the values of the variables PRE and SUF at that time. We will call this *round* g of the learning algorithm.

Let us first describe the global strategy of the learning procedure. When the pattern is a constant $\pi = w$ all sample strings are equal to w and the variables PRE and SUF are not defined. Thus, as long as the algorithm has seen only one string, it will output this string.

Otherwise, we try to generate a pattern from 2 compatible strings received so far. If this is not possible or if one of the samples does not have a factorization then the output will be the *default pattern* $\psi_0 := \text{PRE}_g\ x\ \text{SUF}_g$.

If a non-default pattern has been generated as a hypothesis further samples are tested for compatibility with respect to this pattern. As long as the test is positive the algorithm will stick to this hypothesis, else a new pattern will be generated. In the simplest version of the algorithm we remember only a single example. Instead of a set A we will use a single variable Y.

The One-Variable Pattern Algorithm

```
$Y := X_1$ ;     PRE $:= X_1$ ;     SUF $:= X_1$ ;
output $X_1$ ;
for $g$ = 2, 3, 4, ... do
```

```
$\mathrm{PRE}' := \mathrm{PRE}$;  $\mathrm{SUF}' := \mathrm{SUF}$;
$\psi :=$ output of previous round;
read the new sample $X_g$;
**if** $X_g = \psi$ **then** output $\psi$, **else**
$\mathrm{PRE} := \mathtt{PRE}(\{\mathrm{PRE}, X_g\})$;
$\mathrm{SUF} := \mathtt{SUF}(\{\mathrm{SUF}, X_g\})$;
**if** $\mathrm{PRE} \neq \mathrm{PRE}'$ or $\mathrm{SUF} \neq \mathrm{SUF}'$ **then**
compute the $(\mathrm{PRE}, \mathrm{SUF})$-factorization of $Y$;
    $\psi_0 := \mathrm{PRE}\, x\, \mathrm{SUF}$;
    $\psi := \psi_0$ **endif**;
compute the $(\mathrm{PRE}, \mathrm{SUF})$-factorization of $X_g$;
**case 1:** $Y$ does not have a factorization
    **then** output $\psi_0$:
**case 2:** $X_g$ does not have a factorization
    **then** output $\psi_0$ and $Y := X_g$;
**case 3:** $\psi = \psi_0$
    **if** $X_g$ is downwards compatible to $Y$
        **then** output $\psi(X_g, Y, \cdot)$,
        **else** output $\psi_0$,
    **if** $X_g$ is shorter than $Y$ **then** $Y := X_g$;
**case 4:** $X_g$ is upwards compatible to $\psi$
    **then** output $\psi$;
**case 5:** $X_g$ is downwards compatible to $\psi$
    **then** output $\psi(X_g, \psi, \cdot)$ and $Y := X_g$;
**else** output $\psi_0$.
```

Next, we state the main result of this paper.

THEOREM 1. *One-variable pattern languages can be learned in linear expected time for all distributions that with nonzero probability generate a sample strings by a nonsymmetric replacement of the pattern variable.*

For a proof we refer the reader to [10]. An additional nice feature of the algorithm is the immediate convergence in the final phase when a sample with a nonsymmetric replacement occurs. The expectation of this event is $E[G]$, hence with probability at least $1/2$ the algorithm converges within $2\ E[G]$ rounds. If this did not happen, no matter which bad samples have occurred, again there will be convergence in the next $2\ E[G]$ rounds with probability at least $1/2$. Thus, the probability of failure decrease exponentially with the number of rounds:

$$\Pr\Big[Time_{total} \ \geq\ 2\,k \cdot E[Time_{total}]\Big] \ \leq\ 2^{-k}\ ,$$

which implies that the variance is small.

References

[1] D. Angluin. *Finding Patterns Common to a Set of Strings,* J.CSS 21:46–62, 1980.

[2] M. Crochmore, W. Rytter. *Text Algorithms,* Oxford University Press, 1994.

[3] T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger, T. Zeugmann. *Efficient learning of one-variable pattern languages from positive data,* DOI-TR-128, Kyushu University, Fukuoka, Japan, 1996.

[4] T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger, T. Zeugmann. *Learning One-Variable Pattern Languages Very Efficiently on Average, in Parallel, and by Asking Queries,* Proc. 8. ALT, 1997, LNAI 1316, 260–276.

[5] E. Gold. *Language identification in the limit,* Inf. & Control 10:447–474, 1967.

[6] M. Kearns, L. Pitt. *A polynomial-time algorithm for learning k -variable pattern languages from examples,* Proc. 2. COLT, 1989, 57–71.

[7] S. Lange, R. Wiehagen. *Polynomial-time inference of arbitrary pattern languages,* New Generation Computing 8:361–370, 1991.

[8] S. Lange, T. Zeugmann. *Set-driven and rearrangement-independent learning of recursive languages,* Mathematical Systems Theory 29(6):599–634, 1996.

[9] L. Pitt. *Inductive inference, DFAs and computational complexity,* in K. Jantke (Ed.), Proc. AII, 1989, LNAI 397, 18–44.

[10] R. Reischuk and T. Zeugmann. *Learning One-Variable Pattern Languages in Linear Average Time,* DOI-TR-140, Kyushu University, Fukuoka, Japan, 1997. http://www.i.kyushu-u.ac.jp/ thomas/treport.html

[11] A. Salomaa. *Patterns,* EATCS Bulletin 54:46–62, 1994.

[12] A. Salomaa. *Return to patterns,* EATCS Bulletin 55:144–157, 1994.

[13] R. Schapire. *Pattern languages are not learnable,* Proc. 3. COLT, 1990, 122–129.

[14] T. Shinohara, S. Arikawa. *Pattern inference,* in "Algorithmic Learning for Knowledge-Based Systems," (K. Jantke and S. Lange (Eds.)) LNAI 961, 1995, 259–291.

[15] R. Wiehagen, T. Zeugmann. *Ignoring data may be the only way to learn efficiently,* J. Experimental and Theoretical Artificial Intelligence 6:131–144, 1994.

[16] T. Zeugmann. *Lange and Wiehagen's pattern language learning algorithm: An average-case analysis with respect to its total learning time,* RIFIS-TR 111, Kyushu University, Fukuoka, Japan, 1995.