An Average-Case Optimal One-Variable Pattern Language Learner

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 816-8580
Japan
thomas@i.kyushu-u.ac.jp

Abstract

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

Though one-variable pattern languages can neither be finitely inferred from positive data nor PAC-learned, our approach can also be extended to a *probabilistic finite learner* that *exactly* infers all one-variable pattern languages from positive data with *high confidence.*

It is a long standing open problem whether pattern languages can be learned in case that substitutions of pattern variables by the empty string can also occur. Our learning strategy can be generalized to this situation as well.

Finally, we show some experimental results for the behaviour of this new learning algorithm in pratice.

1. INTRODUCTION

The formal definition of patterns and pattern languages goes back to Angluin [1]. Since then, pattern languages and variations thereof have been widely investigated (cf., e.g., [18, 19]). Patterns provide an intuitively appealing and natural way to define formal languages. Suppose

*This work was performed while this 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.

you want to define the set of all strings of even positive length such that the first half is identical to the second half. In that case, the wanted language follows the pattern $\pi = x\,x$. Here x denotes a pattern variable, and the language generated by π is obtained by substituting any nonempty string for x. Note that this language is not contextfree while the contextfree language of the palindromes (the second half is the reverse of the first half) cannot be represented as a pattern language. Thus, pattern languages are somehow orthogonal to the Chomsky hierarchy.

As far as learning theory is concerned, pattern languages have attracted considerable attention during the last two decades (cf., e.g., [20], and the references therein). In particular, pattern languages are a prominent example of nonregular languages that can be learned in the limit from positive data (cf. [1]). The corresponding learning model goes back to Gold [10]. 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. [17]). Various authors have studied the efficiency of learning in terms of the *update time* needed for computing a new *single* hypothesis. However, processing all initial segments quickly is by no means a guarantee to learn efficiently. What counts in applications is the overall time needed by a learner until convergence, i.e., the *total learning time*. Daley and Smith [5] developed general definitions for the complexity of inductive inference that essentially correspond to the total amount of computation time taken by a learner until successfully inferring the target. But if one allows the total learning time to depend on the length of all examples seen until convergence, then even a polynomially bounded total learning time says fairly nothing about the efficiency of learning, since one may delay convergence until sufficiently long examples have been seen. On the other hand, the total learning time cannot be recursively bounded if it shall exclusively depend on the length of the target, but one allows arbitrarily adverse input sequences.

Valiant's PAC model [21] has resolved this problem by requiring a learner to find, with high confidence, a sufficiently good approximation from any randomly drawn sample of *adequate* size. What is adequate depends on the approximation and confidence parameters as well as on the VC dimension of the target class (cf. [2]). However, recently it has been shown that even the class of one-variable pattern languages has infinite VC dimension (cf. [15]). Thus, these languages are not PAC-learnable. As far as one-variable patterns are concerned, Kearns and Pitt [11] have circumvented the problem of dealing with an infinite VC dimension by *a priori* bounding the length of substitution strings. This approach also works if the overall number of distinct variables occurring in a pattern is *a priori* bounded and if, additionally, the class of distributions is restricted to product distributions (cf. [11]).

This paper makes a rather different approach to design an efficient one-variable pattern language learner. Since the class of one-variable pattern languages is not PAC-learnable, we study their learnability in *the limit* and analyze the total learning time. Moreover, this complexity measure is taken with respect to the length of the target pattern. However, as the total learning time is *unbounded* in the worst-case, we concentrate on the *expected* total learning time.

Let us shortly summarize what has been known concerning the limit learnability of pattern languages. Angluin [1] provides a learner for the class of all pattern languages that is based on the notion of *descriptive patterns*. Here a pattern π is said to be descriptive (for the set S of strings contained in the input provided so far) if π can generate all strings contained in S and no other pattern with this property generates a proper subset of the language generated by π. 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 of a single variable. Angluin [1] gives an algorithm for computing one-variable descriptive patterns. The resulting learner for one-variable pattern languages has update time $O(\ell^4 \log \ell)$, where ℓ is the sum of the length of all different examples seen so far. Nothing is known concerning the expected total learning time of her algorithm.

Erlebach *et al.* [6, 7] have presented a one-variable 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. However, it is debatable whether descriptiveness of intermediate hypotheses should really aimed for, since this may complicate the the learning process and prevent an algorithm from processing the input sequences fast. Thus, we ask whether there are other strategies to learn one-variable pattern language with a significantly smaller expected total learning time — clearly, the best one can hope for is linear. Such a learner would be more appropriate for potential application than previously obtained ones, even if there are less properties guaranteed for the intermediately calculated hypotheses. With high probability, it will already have *finished* its learning task before any of the previously known learner has computed a *single* guess.

What we will present in this paper is an optimal 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. But there is still the problem that, whenever learning in the limit is considered, the learner itself cannot decide whether or not it has already found the correct target. If covergence were decidable one would achieve finite learning (cf. [10]). But one-variable pattern languages are not finitely learnable from positive data. We resolve this problem by establishing *exponentially shrinking tail bounds* for the expected total learning time. Then, requiring a bit prior knowledge about the underlying probability distributions, we naturally arrive at *stochastic finite learning with high confidence.* Now, the learner gets a confidence parameter δ as additional input. Depending on δ and the information about the possible probability distributions it requests a certain number of examples, computes a pattern π from them as its unique hypothesis, and stops thereafter. For a suitable modification of our learning strategy, we will show that with probability at least $1-\delta$ the hypothesis π is *exactly* correct for the target one-variable pattern language. The total amount of time taken is *linearly bounded* in the length of the target pattern and $\log(1/\delta)$.

Note that stochastically finite learning with high confidence is different from PAC-learning. First, it is not completely distribution independent. Thus, from that perspective, this variant is weaker than the PAC-model. Nevertheless, some kind of distribution dependence is inevitable, since one-variable pattern languages are not PAC-learnable. On the other hand, the hypothesis computed is *exactly correct* with high probability. Moreover, the learner receives *exclusively* positive data while the correctness of its hypothesis is measured with respect to *all* data. Hence, from that perspective, our model of stochastic finite learning with high confidence is clearly stronger than the PAC-model.

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 a real number 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. *Pat* denotes 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

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

and set $PAT = \{L(\pi) \mid \pi \in Pat\}$. Let, throughout this paper, the target pattern be grouped in the following form

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

Here the α_i denote 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. Since we exclusively deal with the learnability of one-variable pattern languages, we specialize our definition of learning to this particular case. For a general definition of learning in the limit, the reader is referred to Gold [10]. Note that we actually define a particular form of the model introduced by Freivalds *et al.* [8]. There the examples are presented *on-line*, and the learner has a *long term* and a *short term memory*. The new hypothesis is computed by using the current example presented and what has been stored in its long term memory. For computing its new guess, the learner may use its short term memory in addition. After outputting a hypothesis, the learner decides what to remember in his long term memory. Then, the short term memory is cleared before the next example is read.

DEFINITION 1. A sequence $(\psi_i)_{i\in\mathbb{N}^+}$ of patterns *converges* to a pattern π if $\psi_i = \pi$ for all but finitely many i.
A concept class like PAT is said to be *learnable in the limit* iff there is a learner such that for every $L \in PAT$ and any sequence $X_1, X_2, \dots$ of example strings from L the following holds. Having received X_g the learner computes a hypothesis $\psi_g \in Pat$ such that the sequence of guesses $\psi_1, \psi_2, \dots$ converges to a pattern ψ with $L(\psi) = L$.

Note that in the case of one-variable pattern languages, if $L = L(\pi)$, convergence to a correct hypothesis ψ implies that $\psi = \pi$. Some more remarks are mandatory here. Though our definition of learning resembles that one given in Gold [10] and Freivalds *et al.* [8], there is also a major difference. In [8, 10] the sequence $(X_i)_{i\in\mathbb{N}^+}$ is required to exhaust $L(\pi)$ in the limit, that is to fulfill $\{X_i \mid i \in \mathbb{N}^+\} = L(\pi)$. Nevertheless, in real applications this requirement will hardly be fulfilled. We therefore do not require this property here. Instead, we only assume that the sequence $(X_i)_{i\in\mathbb{N}}$ contains "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

$$n := n_w + n_x \quad \text{with} \quad n_w := \sum |w_i| \quad \text{and} \quad n_x := \sum \alpha_i \ .$$

This parameter n will be considered as the size of problem instances, and the complexity analysis will be done with respect to its value. 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. The inputs are read via a serial input device, and reading a string of length ℓ is assumed to require ℓ steps.

In contrast to previous work (cf., e.g., [1, 12, 22]), we evaluate the efficiency of a learning algorithm by estimating the overall time taken by the learner until convergence (cf. [5]). This time is referred to as the *total learning time.* We aim to measure the total learning time with respect to 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 with respect to distributions that occur in practice. Analyzing the expected total learning time of limit learners has been initiated by Zeugmann [23]. Since the average-case complexity in general depends highly on the distribution over the input space we like to perform our analysis for a large class of distributions.

Our main result, an optimal bound of linear expected total learning time, is achieved for basically all meaningful distributions. 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.

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

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 a single algorithm that efficiently learns every one-variable pattern on the average with respect to every distribution in $\mathcal{D}$. Its strategy is based on a detailed analysis of the combinatorics of words generated by one-variable patterns. It is not required that the algorithm itself has any information about the underlying distribution, this will only be used in the analysis. On the contrary, as already mentioned, for the extension of this algorithm to a stochastic *finite* learner, some information about the distribution is necessary in order to compute the number of examples necessary.

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.** $\quad E[|Z|] \; < \; \infty$.

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.* The bounds obtained by the average-case analysis of this algorithm, however, depend on certain properties of the distributions. This cannot be avoided because one can construct very biased distributions that hide information about the pattern as long as one likes, for example, by making it extremly likely that the substitution starts with a specific letter. Thus, unlike the PAC-model, the complexity bounds are not completely distribution-free. Nevertheless, the parameters necessary to characterize a distribution μ will turn out to be of a very simple nature. 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. As already mentioned 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 hypotheses computed 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. Otherwise, for some $\sigma \in \Sigma$ it would hold $\texttt{Range}(Z) \subseteq L(\sigma x)$ or $\texttt{Range}(Z) \subseteq L(x\sigma)$. To illustrate these quantities, consider the special situation of ***length-uniform distributions***, i.e., distributions where the lengths $|Z|$ of the substitutions may be arbitrary, but for each length ℓ all possible strings over Σ of that length have the same probability. Then it is easy to see that $p_0 \le 1/s$ and $p_a = p_e = 1/s$.

In general, define

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

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

$$F_g[\mathcal{Z}] \ := \ [\ (Z_1[1] = Z_2[1] = \cdots = Z_g[1]) \ \vee \ (Z_1[-1] = Z_2[-1] = \cdots = Z_g[-1])\ ] \ .$$

Then

$$\Pr[F_g] \ \le \ 2\, p^{\,g-1} - p^{\,2(g-1)} \ .$$

Moreover, we define $f(\mathcal{Z}) \ := \ \min\{g \mid \neg F_g[\mathcal{Z}]\}$.

LEMMA 1. *The expectation of $f(\mathcal{Z})$ can be bounded as*

$$E[f(\mathcal{Z})] \ \le \ \frac{2}{1-p} - \frac{p^2}{1-p^2} \ .$$

Proof. Clearly, if $m < \min\{g \mid \neg F_g[\mathcal{Z}]\}$ then $F_m[\mathcal{Z}]$ holds. Thus, we can estimate $\Pr[f(\mathcal{Z}) > m] \leq 2p^{m-1} - p^{2(m-1)}$, and a simple calculation yields

$$E[f(\mathcal{Z})] = \sum_{m=0}^{\infty} \Pr[f(\mathcal{Z}) > m] \leq 2 + 2 \cdot \sum_{m \geq 2} p^{m-1} + \sum_{m \geq 2} p^{2(m-1)} = \frac{2}{1-p} - \frac{p^2}{1-p^2} .$$

∎

4. SYMMETRY OF STRINGS

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

Definition 2. Let $y = y[1]y[2]\ldots y[\ell] \in \Sigma^+$ be a string of length ℓ. If for some k with $1 \leq k \leq \ell/2$ the k-length prefix and suffix of y are identical, that is $y[1\ldots k] = y[\ell - k + 1 \ \ldots \ell]$, we say that y has a ***k*–*symmetry*** $u = y[1\ldots k]$ (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 $c, d \in \mathbb{N}^+$ maximal such that $y = u^c \, v_0 \, u^d$, for some string v_0, 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 $c \geq d$ 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 $y = u^{c_0} \, v_1 \, u^{c_1} \, v_2 \, \ldots v_r \, u^{c_r}$ 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 $\boldsymbol{mls(y)}$, 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 $\boldsymbol{sym(y)}$ 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 possesses the factorization $y = u^c \, v_0 \, u^d$ then it has k'-symmetries for $k' = 2k,\ 3k,\ \ldots,\ \min\{c,d\}\, k$, too.*

(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) $sym(y) \;\leq\; |y| \,/\, 2\, mls(y)$.

Proof. If a symmetry u of a string y can be written as $u = u'\ v\ u'$ for a nonempty string u then obviously u' is a smaller symmetry of y. Hence, (1) follows.

Assertion (2) is obvious. If there were other symmetries in between then it is easy to see that u itself must have a symmetry and thus cannot be minimal. This proves (3).

If v_0 contains u as a substring there may be other larger symmetries. For this case there must be strings v_1, v_2 such that y can be written as

$$y \;=\; u^c\ v_1\ u^d\ v_2\ u^c\ v_1\ u^d$$

where v_1 does not contain u as substring. Then y has an additional symmetry for $k' = (c+d)\ k + |v_1|$. There may be even more symmetries if v_2 is of a very special form containing powers of u, but we will not elaborate on this further. The important thing to note is that the length of such symmetries grows at least by an additive term $k = mls(y)$. The bound on $sym(y)$ follows. ∎

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. Only strings over a single letter alphabet can achieve this bound. For particular distributions the bound is usually much better. To illustrate this, we again consider the length-uniform case. Then, the probability that a random string y has a minimal symmetry of length k is given by

$$\Pr[mls(y) = k] \;=\; \Pr[|y| \geq 2k] \cdot s^{-k} \;.$$

Furthermore, given that $mls(y) = k$ the probability that it has at least c symmetries is bounded by

$$\Pr[sym(y) \geq c \mid mls(y) = k] \leq s^{-k \cdot 2\,(c-1)} \cdot \frac{1}{1 - s^{-2c+1}} \;.$$

Thus, the probability of having at least c symmetries is at most

$$\sum_{k \geq 1} s^{-k \cdot (2c-1)} \;\leq\; \frac{s^{-2c+1}}{(1 - s^{-2c+1})^2} \;.$$

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

LEMMA 3. *In the length-uniform case*

$$E[sym(Z)] \;\leq\; \frac{s}{2(s-1)^2} \;.$$

Proof. Using the equality $\sum_{c\geq 1} c\cdot\alpha^c = \frac{\alpha}{(1-\alpha)^2}$ for $\alpha = s^{-2k}$ in the estimation below one gets

$$\begin{aligned}
E[sym(Z)] &= \sum_{k\geq 1}\Pr[mls(Z)=k]\cdot\sum_{c\geq 1} c\cdot\Pr[sym(Z)=c \mid mls(Z)=k]\\
&\leq \sum_{k\geq 1} s^{-k}\ \sum_{c\geq 1} c\cdot s^{-k\,2(c-1)}\cdot\frac{1}{1-s^{-2c+1}}\\
&\leq \sum_{k\geq 1} s^{k}\ \sum_{c\geq 1} c\cdot (s^{-2k})^c\cdot\frac{1}{1-s^{-1}}\\
&= \frac{1}{1-s^{-1}}\cdot\sum_{k\geq 1} s^k\cdot\frac{s^{-2k}}{(1-s^{-2k})^2}\ \leq\ \frac{s}{2(s-1)}\ \sum_{k\geq 1} s^{-k}\\
&= \frac{s}{2\,(s-1)^2}\ .
\end{aligned}$$

∎

Thus, in this case the number of symmetries only depends on the size s of the alphabet and therefore, it is independent of the length of the strings generated. Next, let us estimate the total length of all factorizations of a string y, which can be bounded by $|y|\cdot sym(y)$. In the length-uniform case, we get

$$\begin{aligned}
E[|Z|\cdot sym(Z)] \;=\; &\sum_{\ell\geq 1}\Pr[|Z|=\ell]\cdot\ell\ \cdot\ \sum_{k\geq 1}\Pr[mls(Z)=k \mid |Z|=\ell]\\
&\cdot\ \sum_{c\geq 1} c\cdot\Pr[sym(Z)=c \mid mls(Z)=k,\ |Z|=\ell]\ .
\end{aligned}$$

The estimation for $E[sym(Z)]$ above is independent of the length of Z, thus the terms

$$\sum_{k\geq 1}\Pr[mls(Z)=k \mid |Z|=\ell]\cdot\sum_{c\geq 1} c\cdot\Pr[sym(Z)=c \mid mls(Z)=k,\ |Z|=\ell]$$

can be replaced by the bound for

$$\begin{aligned}
&\sum_{k\geq 1}\Pr[mls(Z)=k]\cdot\sum_{c\geq 1} c\cdot\Pr[sym(Z)=c \mid mls(Z)=k]\\
&\quad\leq\ \sum_{k\geq 1} s^{-k}\ \sum_{c\geq 1} c\cdot s^{-k\,2(c-1)}\cdot\frac{1}{1-s^{-2c+1}}\ .
\end{aligned}$$

This gives for $E[|Z|\cdot sym(Z)]$ the same bound as for $E[|Z|]\ \cdot\ E[sym(Z)]$.

For arbitrary distributions, we require

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

Remember that we already had to assume $E[|Z|]$ to be 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 we also have $0 < E[sym(Z)] < \infty$. 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.

Proof. To find the minimal symmetry an iterative scanning of specific bit positions of y is done. Let ℓ denote the length of y.

***Algorithm* 1.**

For $j = 1, 2, \ldots, \ell - 1$, we will construct subsets I_j of $[1 \ldots \ell]$ with the property:

$$t \in I_j \iff y[t \ldots t+j-1] \; = \; y[1 \ldots j] \; .$$

The sets are initialized with $I_j = \{1\}$ for all j. Then $I_1 \; := \; \{t \mid y[t] = y[1]\}$.
Assume that I_{j-1} has been constructed.

if $j \in I_{j-1}$ then
    $y[j \ldots 2j-2] = y[1 \ldots j-1]$ implying
    $y[1 \ldots 2j-2] = y[1 \ldots j-1]^2$.
    $\forall \; t \in I_{j-1}$:
    if $t + j - 1 \in I_{j-1}$ then $t \hookrightarrow I_{2j-2}$
    if $2j - 1 \geq \ell$ then stop and output `FAILURE`   else $j := 2j - 1$

if $j \notin I_{j-1}$ then
    $\forall \; t \in I_{j-1}$:
    if $y[t + j - 1] = y[j]$ then $t \hookrightarrow I_j$
    if $\ell - j + 1 \in I_j$ then
      stop with success and output $y[1 \ldots j]$
    if $j \geq \ell$ then stop and output `FAILURE`   else $j := j + 1$

It can be shown that this procedure considers each bit position $y[j]$ at most a logarithmic number of times from which the bound $O(\ell \log \ell)$ follows easily. For most strings, however, the complexity is linear since more than linear time is needed only for strings of highly regular structure.

***Algorithm* 2.**

The second, and in the worst-case more efficient algorithm first computes a **maximal overlap** of y, that is a substring w of maximal length such that

$$y \; = \; w \, \nu_1 \; = \; \nu_2 w$$

for some nonempty strings ν_1, ν_2. If $|w| \leq |y|/2$ then y can be written in the form

$$y \; = \; w \, \nu \, w$$

for some string ν, that means w is a symmetry of y. Since w was chosen maximal it is even the maximal symmetry. If $|w| > |y|/2$ then in the representation $y = w\ \nu_1 = \nu_2 w$ the string w overlaps with itself, thus it cannot be used as a symmetry. However, let κ denote the length of the ν_i, $r = \ell - \kappa$ the length of w, and $r' := \ell \mod \kappa$. Then

$$w[\kappa+1 \dots r] \;=\; w[1 \dots r-\kappa] \;,$$

which implies in particular $w[1 \dots \kappa] \;=\; w[\kappa+1 \dots 2\kappa]$. In the same way, $w[\kappa+1 \dots 2\kappa] \;=\; w[2\kappa+1 \dots 3\kappa]$, thus w can be written as

$$w \;=\; w[1 \dots \kappa]^{\lfloor r/\kappa \rfloor}\ w[1 \dots r'] \qquad \text{and} \qquad y \;=\; w[1 \dots \kappa]^{\lfloor r/\kappa \rfloor + 1}\ w[1 \dots r'],$$

where $w[1 \dots r']$ is empty for $r' = 0$. Now define

$$w_0 \;:=\; \begin{cases} w, & \text{if} \quad |w| \le \ell/2, \\ w[1 \dots \kappa], & \text{if} \quad r' = 0, \\ w[1 \dots r'], & \text{otherwise} . \end{cases}$$

Note that w_0 is a symmetry of y. As already mentioned it is the maximal one in the first case, whereas in the other cases the maximal one is $w[1 \dots \kappa]^{\ell/2\kappa}$ and $w[1 \dots \kappa]^{(\ell-\kappa)/2\kappa}\ w[1 \dots r']$, if $r' = 0$ and for $r' > 0$, respectively. Symmetries of size between w_0 and the maximal one are of the form $w[1 \dots \kappa]^L\ w[1 \dots r']$ for some $1 \le L < \ell/2\kappa$. Having obtained w_0, iteratively in the same way we first compute the maximal overlap of this string and from this a substring w_1, and so forth until for the first time w_j has zero overlap. Then w_j is the minimal symmetry of y.

A maximal overlap (sometimes also called a maximal border) can be computed in linear time, see for example [4], Chapter 3.1, where an algorithm of complexity $2\ |y| - 3$ is described.

Given the maximal overlap, the string w_0 can easily be obtained in a linear number of steps. Since for all j the length of w_j is at most half the length of w_{j-1} the whole iterative procedure stays linearly bounded.

Once we have found a symmetry u, computing the complete u–factorization of y is just a simple pattern matching of u against y, which can be done by well established methods in linear time.

From a complete minimal factorization based on u_1 other symmetries can be deduced by checking powers of u_1 and the equality of substrings between these powers. This can be done in a linear number of operations. ∎

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. $\quad p_{\mathrm{sym}} < 1$.

Now consider the event

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

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

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

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

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

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. Let $m_{\text{pre}}(A)$ and $m_{\text{suf}}(A)$ 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 $\mathtt{PRE}(\{X, \text{PRE}\}) \neq \text{PRE}$ or $\mathtt{SUF}(\{X, \text{SUF}\}) \neq \text{SUF}$ 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 $Y = \text{PRE}\ u_1^{c_1}\ v_1\ u_1^{d_1}\ \text{SUF}$ 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 $Y = \text{PRE}\ u_i^{c_i}\ v_i\ u_i^{d_i}\ \text{SUF}$ 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* $Y = \text{PRE}\ u_1^{c_1}\ v_1\ u_1^{d_1}\ \text{SUF}$ *be the minimal* (PRE, SUF)*–factorization of* Y. *Then, for every string* $\tilde{Y}$ *of the form* $\tilde{Y} = \text{PRE}\ u_1\ \tilde{v}\ u_1\ \text{SUF}$ *for some string* $\tilde{v}$, *the minimal* (PRE, SUF)*–factorization of* $\tilde{Y}$ *is based on* u_1, *too.*

Proof. That u_1 gives rise to a factorization is obvious. There cannot be one of smaller length because this implies that u_1 has a symmetry and contradicts that u_1 is minimal for Y. ∎

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

LEMMA 7. *Let* $\pi = w_0\, x\, v\, x\, w_m$ *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 $\boldsymbol{\psi = \psi(Y, \tilde{Y})}$ can be derived from which both strings can be generated. More precisely, it has to hold:

$$Y \;=\; \mathrm{PRE}\; u_1^{c_1}\; v_1\; u_1^{d_1}\; \mathrm{SUF} \qquad \text{and} \qquad \tilde{Y} \;=\; \mathrm{PRE}\; \tilde{u}_1^{\tilde{c}_1}\; \tilde{v}_1\; \tilde{u}_1^{\tilde{d}_1}\; \mathrm{SUF}\;,$$

and for

$$Y_{\mathrm{mid}} \;:=\; u_1^{c_1 - 1}\; v_1\; u_1^{d_1 - 1} \qquad \text{and} \qquad \tilde{Y}_{\mathrm{mid}} \;:=\; \tilde{u}_1^{\tilde{c}_1 - 1}\; \tilde{v}_1\; \tilde{u}_1^{\tilde{d}_1 - 1}$$

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* $(\mathrm{PRE}, \mathrm{SUF}) = (w_0, w_m)$ *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.

Proof. It is easy to see that for a nonsymmetric string u the string

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

has u as the basis for its minimal (w_0, w_m)–factorization. That u gives rise to a factorization is obvious, and if there were a smaller one it would imply that u has a symmetry. Since the constant substrings $w_1, \ldots, w_{m-1}$ may contain u as a substring the actual factorization may show more powers of u, but it is unique since occurrences of u cannot overlap – again because u is nonsymmetric. If the constant substring w_i of π has a decomposition with respect to u of the form $u^{\beta_{i,0}}\, \omega_{i,1}\, u^{\beta_{i,1}} \ldots \omega_{i,ni}\, u^{\beta_{i,ni}}$, where the $\beta_{i,j}$ are integers and the $\omega_{i,j}$ are substrings not containing u, then the middle part Y_{mid} of Y without prefix, suffix and first and last occurrence of u looks like

$$u^{\alpha_1 - 1}\; u^{\beta_{1,0}}\; \omega_{1,1}\; u^{\beta_{1,1}} \ldots \omega_{1,n1}\; u^{\beta_{1,n1}}\; u^{\alpha_2}\; u^{\beta_{2,0}}\; \omega_{2,1}\; u^{\beta_{2,1}} \ldots \omega_{2,n2}\; u^{\beta_{2,n2}}\; u^{\alpha_3}\; \ldots\; u^{\alpha_m - 1}\;.$$

When checking direct compatibility of Y against π it becomes obvious whether a substring u in Y corresponds to a variable or not. ∎

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 $\tilde{Y}_{\text{mid}}$, 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 $\tilde{u}$ 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.

Proof. Let u be nonsymmetric, $Y = \pi[x/u]$, and let

$$Y_{\text{mid}} = u^{\alpha_1 - 1}\ u^{\beta_{1,0}}\ \omega_{1,1}\ u^{\beta_{1,1}} \ldots \omega_{1,n1}\ u^{\beta_{1,n1}}\ u^{\alpha_2}\ u^{\beta_{2,0}}\ \omega_{2,1}\ u^{\beta_{2,1}} \ldots\ u^{\alpha_m - 1}\ .$$

If

$$\tilde{Y} = \pi[x/\tilde{u}]\ =\ w_0\ \tilde{u}^{\alpha_1}\ w_1\ \tilde{u}^{\alpha_2}\ w_2\ \ldots\ w_{m-1}\ \tilde{u}^{\alpha_m}\ w_m$$

then $\tilde{Y}$ has a (w_0, w_m)–factorization based on $\tilde{u}$. Note that this factorization will not be minimal if $\tilde{u}$ itself has symmetries. Since $w_1, \ldots, w_{m-1}$ may contain $\tilde{u}$ the actual factorization may show more powers of $\tilde{u}$.

By assumption, u is not a symmetry for $\tilde{u}$ and since one may either work from left to right or right to left we may assume that u is not a prefix of $\tilde{u}$. When comparing $Y_{\rm mid}$ to $\tilde{Y}_{\rm mid}$ after the first $\alpha_1 - 1$ occurrences of u in $Y_{\rm mid}$ have been read and matched against occurrences of $\tilde{u}$ in $\tilde{Y}_{\rm mid}$ the next occurrence of u in the substring $u^{\beta_{1,0}}$ will be detected as a constant. This is because this substring also occurs in $\tilde{Y}_{\rm mid}$ and u is not a prefix of $\tilde{u}$. The same holds for the other occurrences of u in Y.

Given the corresponding factorizations, checking whether $Y_{\rm mid}$ and $\tilde{Y}_{\rm mid}$ match can be done by a single pass over the strings and has linear time complexity. However, one has to find that factorization of $\tilde{Y}$ that matches the one of Y. Considering the symmetries of $\tilde{Y}$ in increasing length this will be symmetry $sym(\tilde{u})$. In the worst-case, if $\tilde{u}$ contains only one symbol $sym(\tilde{u})$ can be as large as $|\tilde{u}|/2$, but such a case will be easier to handle.

This can even be sped-up. One observation is that a string with c symmetries yields a least by a factor c more occurrences of its minimal symmetry in the minimal factorization. Thus, once one output pattern ψ has been computed, which also gives the number of occurrences of the pattern variable, strings $\tilde{Y}$ with a much larger number of occurrences in the minimal factorization based on a string $\tilde{u}_1$ can simply be discarded unless ψ itself contains lots of substrings $\tilde{u}_1$. More precisely, let

$$\#(Y,u) \quad := \quad \text{maximal number of nonoverlapping occurrences of } u \text{ in } Y.$$

Since u is nonsymmetric and $Y = \pi[x/u]$

$$\#(Y,u) \;=\; \#(\pi,x) \;+\; \#(\pi,u)\;.$$

For $\tilde{Y} = \pi[x/\tilde{u}]$ and a symmetry $\tilde{u}_i$ of a factorization of $\tilde{Y}$ such that $\tilde{u}_i$ is a substring of $\tilde{u}$ it holds,

$$\#(\tilde{Y},\tilde{u}_i) \;=\; \#(\tilde{u},\tilde{u}_i)\cdot\#(\pi,x) \;+\; \#(\pi,\tilde{u}_i)\;.$$

Let ψ a pattern that is supposed to equal the pattern π to be learned. Thus, to find the right factorization of a string $\tilde{Y}$ to check upwards compatibility against ψ from the minimal factorization one can compute

$$\frac{\#(\tilde{Y},\tilde{u}_1) \;-\; \#(\psi,\tilde{u}_i)}{\#(\psi,x)}$$

to get an estimate on $\#(\tilde{u},\tilde{u}_1)$. When all symmetries of $\tilde{Y}$ are known it is then easy to find that string $\tilde{u}$ directly that matches this value. However, when checking upwards compatibility of a string $\tilde{Y}$ to a string Y, we do not have a precise estimate on $\#(\pi,x)$, there is only the upper bound $\#(Y,u)$ available from the factorization of Y. This implies a lower bound on $\#(\tilde{u},\tilde{u}_1)$ of the form

$$\#(\tilde{u},\tilde{u}_1) \;\geq\; \frac{\#(\tilde{Y},\tilde{u}_1) \;-\; \#(Y,\tilde{u}_i)}{\#(Y,u)}\;.$$

Thus, unless $\#(\pi,u)$ is relatively large compared to $\#(\pi,x)$ this gives a good approximation which symmetry of $\tilde{Y}$ should be used. ∎

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. LEARNING ONE-VARIABLE PATTERNS

Now, we are ready to present our one-variable pattern language learners. First, we describe the average-case optimal learning algorithm. Its correctness is established in Subsection 6.2. Next, we analyze its expected total learning time.

6.1. *The Algorithm*

The learner may not store all example 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 examples 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 example 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 examples does not have a factorization then the output will be the *default pattern*

$$\psi_0 := \mathbf{PRE}_g\ \boldsymbol{x}\ \mathbf{SUF}_g\,.$$

If a non-default pattern has been generated as a hypothesis further examples 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 of the ones seen so far. Instead of a set A we will use a single variable Y.

The One-Variable Pattern Learning Algorithm

$Y := X_1$;
PRE $:= X_1$;
SUF $:= X_1$;
output X_1;

for $g = 2, 3, 4, \ldots$ **do**
PRE$'$:= PRE; SUF$'$:= SUF;
$\psi :=$ output of previous round;

read the new example X_g ;

if $X_g = \psi$ **then** output ψ , **else**
PRE := $\texttt{PRE}(\{\text{PRE}, X_g\})$;
SUF := $\texttt{SUF}(\{\text{SUF}, X_g\})$;
if PRE $\neq$ PRE$'$ or SUF $\neq$ SUF$'$ **then**
compute the (PRE, SUF) –factorization of Y ;
ψ_0 := PRE x SUF ;
$\psi := \psi_0$ **endif**;

compute the (PRE, SUF) –factorization of X_g ;

case 1: Y does not have a factorization
then output ψ_0 :
case 2: X_g does not have a factorization
then output ψ_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 ψ_0 ,
if X_g is not larger than Y **then** $Y := X_g$;

case 4: X_g is upwards compatible to ψ
then output ψ ;
case 5: X_g is downwards compatible to ψ
then output $\psi(X_g, \psi, \cdot)$ and $Y := X_g$;
else if X_g is not larger than Y **then** $Y := X_g$;
output ψ_0 .

6.2. *Proof of Correctness*

Since the example strings are generated at random it might happen that only "bad examples" occur in which case no learning algorithm can eventually come up with a correct hypothesis. Therefore, the following claims cannot hold absolutely in a probabilistic setting, but they will be true with probability 1. Remember that $\pi = w_0 x^{\alpha_1} w_1 x^{\alpha_2} w_2 \ldots w_{m-1} x^{\alpha_m} w_m$ is the pattern to be learned. Since not all substitutions start with the same symbol or end with the same symbol (remember that we have assumed $p < 1$) with probability 1 a sequence $\mathcal{X}$ contains strings X_i, X_j, X_k , where $X_\kappa = \pi[x/u_\kappa]$ such that

$$u_i[1] \neq u_j[1] \qquad \text{and} \qquad u_i[-1] \neq u_k[-1] \ .$$

Note that j may be equal to k . Let g be the maximum of i, j, k and consider a triple for which g is minimal. By the construction of the sets PRE and SUF round g will start a new phase in which now the variables $\text{PRE}_g = w_0$ and $\text{SUF}_g = w_m$ have the correct values.

We do not care about the output of the algorithm before this final phase has been reached. It remains to show that the algorithm will converge in the final phase. For this purpose, let us distinguish whether the pattern contains the variable only once, in which case there will be examples without any symmetry, or more than once (the case that the pattern does not contain any variable is obvious).

If $\pi = w_0 x w_1$ then with probability 1 there will be an example X_g obtained from a substitution $[x/u]$ with a nonsymmetric string u. Then X_g does not have a $(\mathrm{PRE}_g, \mathrm{SUF}_g)$–factorization and thus case 2 occurs. Since Y is set equal to X from then on always case 1 occurs. The algorithm will always choose case 1 and output ψ_0, which in this case is the correct answer.

Otherwise, the pattern contains the variable at least twice and any example does have a $(\mathrm{PRE}_g, \mathrm{SUF}_g)$–factorization. Lemma 11 shows that a nonsymmetric substitution generates a string that is downwards compatible to any other string in $L(\pi)$. Thus, as soon as X_g is such a string, which again happens with probability 1, the output ψ_g will equal the pattern π. Furthermore the algorithm will never change its output from this round on since case 4 "$X_{g'}$ is upwards compatible to ψ" will hold for any $g' > g$. Let us summarize these properties in the following

LEMMA 12. *After the algorithm has detected the correct prefix and suffix it will converge immediately to the correct hypothesis π as soon it gets the first example generated by a nonsymmetric substitution.*

For the case that substitutions with large symmetries occur very frequently the algorithm can be modified to achieve convergence even before seeing a nonsymmetric substitution. For this purpose we perform a complete compatibility test in cases 3 to 5 between the new example X_g and the string Y remembered, resp. the hypothesis ψ. This may increase the computational effort within one round, but reduces the number of rounds. This modification complicates the complexity analysis, therefore we will stick to the original version in the estimations below. For the experimental tests described in section 8, however, we have used an implementation of this faster version in order to achieve convergence even for distributions with large symmetries, that is for the case $p_{\mathrm{sym}} = 1$.

6.3. *Pattern Languages with Empty Substitutions*

It is a long standing open problem whether pattern languages with empty substitutions can be learned in the classical sense. Angluin's approach computing descriptive patterns does not work in this case. Our algorithm, however, is flexible enough to disregard certain examples temporarily and thus can solve this more difficult problem by the same strategy. The idea is to postpone the shortest example among all seen so far. This will guarantee that an example obtained by an empty substitution will never be processed, which otherwise would confuse the learner and prevent him from finding the correct pattern.

The modification uses an additional variable Z which stores the shortest string seen so far. For initializaton the first two distinct examples X_1 and X_j with $j \geq 2$ are needed. Z is initialized to the shorter one, resp. to X_1 if both have the same length (in this case neither X_1 nor X_j are generated from an empty substitution and thus one could use both as well). Y, PRE and SUF are all initialized to the longer string, and then the for-loop starts with the next example X_{j+1}. At the beginning of each round the length of Z is compared with the new example, and they are exchanged in case that the new example is shorter than Z. Up to round $j-1$, X_1 will serve as output. The hypothesis of the j-th round is Y.

The correctnes of this modified version follows easily from the analysis above. Since example obtained from empty substitutions are never considered the modified algorithm will behave in the same way as the original one for a slightly modified sequence of examples, namely $X_1, Y, X'_{j+1}, X'_{j+2}, \ldots$, where X'_{j+1} denotes the result after comparing X_{j+1} with Z. Since this is a possible sequence for the case without empty substitutions, for which we have proven correct convergence, this holds for the modified version as well. The round of convergence, however, may be later. If the first nonsymmetric substitution X_g happens to be the shortest example seen so far it will be postponed. Thus we may have to wait for the second nonsymmetric substitution or a shorter string than X_g. Thus one can expect the modified algorithm to require at least one more round.

6.4. *Complexity Analysis of the Basic Algorithm*

Let ψ_g denote the output of round g, and Y_g the value of Y at the end of that round. Let $Time_g(\mathcal{X})$ denote the number of bit operations in round g on example sequence $\mathcal{X}$, and recall that Z and X are defined as random variables for the substitutions and examples, respectively.

LEMMA 13. *For each round g it holds*

$$\begin{aligned} E[Time_g(\mathcal{X})] &\le O\Big(E[|X|] \cdot (1 + E[sym(Z)])\Big) \\ &\le O\Big(n \cdot E[|Z|] \cdot (1 + E[sym(Z)])\Big). \end{aligned}$$

Proof. By Lemma 10 and 11 in each round g the number of bit operations can be estimated by

$$\begin{aligned} Time_g(\mathcal{X}) \le\ & O(|Y_{g-1}|) + O(|X_g|) + \\ & \max\{O((|Y_{g-1}| + |X_g|) \cdot (1 + sym(Y_{g-1})), \\ & \quad O((|\psi_{g-1}| + |X_g|) \cdot (1 + sym(X_g)), O(|\psi_{g-1}| + |X_g|)\} \\ \le\ & O\Big(|Y_{g-1}| + |X_g| + |\psi_{g-1}| + |Y_{g-1}| \cdot sym(Y_{g-1}) + \\ & \quad |X_g| \cdot sym(Y_{g-1}) + |\psi_{g-1}| \cdot sym(X_g) + |X_g| \cdot sym(X_g)\Big). \end{aligned}$$

By construction of the algorithm and the fact that a pattern is never longer than an example string it generates we can bound $E[|X_g|]$ as well as $E[|Y_{g-1}|]$ and $E[|\psi_{g-1}|]$ by $E[|X|]$. Moreover, Assumption 3 directly implies that $E[|X_g| \cdot sym(X_g)]$ and $E[|Y_{g-1}| \cdot sym(Y_{g-1})]$ are both bounded by $O(E[|X|] \cdot E[sym(Z)])$. Note, that X_g is independent of Y_{g-1} and ψ_{g-1}. Thus

$$\begin{aligned} E[|X_g| \cdot sym(Y_{g-1})] &= E[|X_g|] \cdot E[sym(Y_{g-1})] = E[|X|] \cdot E[sym(Z)] \qquad \text{and} \\ E[|\psi_{g-1}| \cdot sym(X_g)] &= E[|\psi_{g-1}|] \cdot E[sym(X_g)] \le E[|X|] \cdot E[sym(Z)]. \end{aligned}$$

This simplifies the expectation to

$$\begin{aligned} E[Time_g(\mathcal{X})] \le\ & O\Big(E[|Y_{g-1}|] + E[|X_g|] + E[|\psi_{g-1}|] \\ & + E[|Y_{g-1}| \cdot sym(Y_{g-1})] + E[|X_g| \cdot sym(Y_{g-1})] \\ & + E[|\psi_{g-1}| \cdot sym(X_g)] + E[|X_g| \cdot sym(X_g)]\Big) \\ \le\ & O\Big(E[|X|] + E[|X|] \cdot E[sym(Z)]\Big). \end{aligned}$$

$\blacksquare$

Now we can also bound the total learning time.

LEMMA 14. *The expected total learning time is bounded by*

$$
\begin{aligned}
& O\left(E[|X|]\cdot(1+E[sym(Z)])\cdot\Big(\frac{1}{1-p}+\frac{1}{1-p_{\mathrm{sym}}}\Big)\right)\\
\le\ & O\left(n\cdot E[|Z|]\cdot(1+E[sym(Z)])\cdot\Big(\frac{1}{1-p}+\frac{1}{1-p_{\mathrm{sym}}}\Big)\right)\ .
\end{aligned}
$$

Since $E[|Z|])$, $E[sym(Z)]$, p, and p_{sym} are characterized by the distribution for substituting the pattern variable they are all independent of the problem size. This means the complexity grows linear with the size of the problem.

Proof. The number of rounds can be bounded by the number of rounds to reach the final phase plus the number of rounds in the final phase till $\psi_g = \pi$. By Lemma 1 and 5 the expectation of both is a constant that only depends on the probabilities p and p_{sym}. Let G be a random variable that counts the number of rounds till convergence. Then,

$$
E[G]\ \le\ O\left(\frac{1}{1-p}+\frac{1}{1-p_{\mathrm{sym}}}\right)\ . \tag{1}
$$

Let $Time_{total}(\mathcal{X})$ denote the total number of operations on example sequence $\mathcal{X}$. Then

$$
\begin{aligned}
Time_{total}(\mathcal{X}) &= \sum_{g=1}^{G} Time_g(\mathcal{X}) \ = \sum_{t\ge 2}\Pr[G=t]\ \cdot\ \sum_{g=1}^{t} Time_g(\mathcal{X}) \qquad \text{and}\\
E[Time_{total}(\mathcal{X})] &= E\Big[\sum_{t\ge 2}\Pr[G=t]\cdot\sum_{g=1}^{t} Time_g(\mathcal{X})\Big]\\
&\le O\Big(E\Big[\sum_{t\ge 2}\Pr[G=t]\cdot t\cdot E[|X|]\cdot(1+E[sym(Z)])\Big]\Big)\\
&\le O\left(E[|X|]\cdot(1+E[sym(Z)])\cdot E\Big[\sum_{t}\Pr[G=t]\cdot t\Big]\right)\\
&= O\Big(E[|X|]\cdot(1+E[sym(Z)])\cdot E[G]\Big)\\
&\le O\left(n\cdot E[|Z|]\cdot(1+E[sym(Z)])\cdot\Big(\frac{1}{1-p}+\frac{1}{1-p_{\mathrm{sym}}}\Big)\right)\\
&= O(n)\ .
\end{aligned}
$$

$\blacksquare$

Summarizing, we state the first main result of this paper.

THEOREM 1. *One-variable pattern languages can be inferred in linear expected total learning time for all distributions that fulfill the Assumptions 1 through 4 made above.*

Clearly, the expected value of a random variable is only one aspect of its distribution. Looking at potential applications of our learning algorithm, a hypothetical user might be interested in knowing how often the total learning time exceeds its average substantially. For answering this question we could compute the variance of the total learning time. Then Chebyshev's inequality provides the desired tail bounds. However, in our particular setting, there is an easier way to figure out how good the distribution of the total learning time is centered around its expected value, that is, proving *tail bounds.*

THEOREM 2. *For all* $\tau \in \mathbb{N}$ *it holds:*

$$\Pr\Big[Time_{total} \;\geq\; 2 \cdot \tau \cdot E[Time_{total}]\Big] \;\leq\; 2^{-\tau}. \tag{2}$$

Proof. Our algorithm converges immediately when an example 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 has not happened no matter which bad examples have occurred, again there will be convergence in the next $2\ E[G]$ rounds with probability at least $1/2$. ∎

Since the distribution of $Time_{total}$ decreases exponentially, all higher moments of it exist. In particular, we may conclude that the variance of $Time_{total}$ is small.

7. STOCHASTIC FINITE LEARNING

In this section, we convert the learning algorithm presented in Subsection 6.1 into a learner that identifies all one-variable pattern languages from positive data in a bounded number of rounds *stochastically finite with high confidence.* The additional ingredient needed is certain amount of *additional knowledge* concerning the underlying class of probability distributions. Therefore, the resulting learning model is not distribution-free, and hence in this respect weaker than Valiant's [21] PAC model. On the other hand, one has to make certain assumptions on the class of probability distributions, since the one-variable pattern languages are not PAC-learnable (cf. [15]). But on the other hand, our model is stronger than the PAC model by requiring the output to be *exactly correct* with high probability. Moreover, the learner has to infer its hypotheses from positive data only, while the correctness of the output is measured with respect to all data, positive and negative. We continue with the formal definition.

DEFINITION 7. Let $\mathcal{D}$ be a set of probability distributions on the learning domain, let $\mathcal{C}$ a concept class, $\mathcal{H}$ a hypothesis space for $\mathcal{C}$, and let $\delta \in (0,1)$. $(\mathcal{C},\mathcal{D})$ is said to be *stochastically finite learnable with* δ*-confidence* from positive data with respect to $\mathcal{H}$ iff there is a learner that for every $c \in \mathcal{C}$ and every $D \in \mathcal{D}$ performs as follows. Given a random presentation $\mathcal{X} = (X_j)_{j\in\mathbb{N}}$ for c generated according to D, the learner stops after having seen a finite number of examples and outputs a single hypothesis $h \in \mathcal{H}$. With probability at least $1-\delta$ (with respect to distribution D) h has to be correct, that is $h = c$.

If stochastic finite learning can be achieved with δ-confidence for every $\delta > 0$ then we say that $(\mathcal{C},\mathcal{D})$ can be learned stochastically finite *with high confidence.*

Next, we specify the additional knowledge the learner must possess for learning stochastically finite with high confidence. Recall that the number G of rounds depends only on p and $p_{\rm sym}$. Clearly, p and $p_{\rm sym}$ themself are usually not known. But it is *reasonable* to assume the knowledge of *upper bounds* for both parameters. We therefore define the class $\mathcal{D}[p^*, p^*_{\rm sym}]$ of admissible probability distributions to be the set of all distributions fullfilling Assumptions 1 through 4 in a way such that $p \le p^*$ and $p_{\rm sym} \le p^*_{\rm sym}$. Then, the following can be shown.

THEOREM 3. *Let $p^*, p^*_{\rm sym} < 1$, and let $\mathcal{D}[p^*, p^*_{\rm sym}]$ be a class of admissible probability distributions. Then $(PAT, \mathcal{D}[p^*, p^*_{\rm sym}])$ is stochastically finitely learnable with high confidence from positive data using $O(\log(1/\delta) \cdot |\pi|)$ many examples.*

Proof. First note that the learner gets δ as additional input. In addition to the limit learner, it uses a counter for memorizing the *number* of examples already seen. The expected number of rounds is estimated by evaluating Formula 1 for p^* and $p^*_{\rm sym}$. Let $\tilde{G}$ be this estimate. Furthermore, the learner computes the least m such that $2^{-m} \le \delta$, and runs teh basic algorithm for $2 \cdot m \cdot \tilde{G}$ rounds. While doing this, no output is provided. After having finished these rounds, the learner outputs the last guess π made by the original algorithm, and stops thereafter. Now, using the same argument as above for proving (2), one easily sees that π will be the correct target with probability at least $1 - \delta$. By construction the total learning time remains linear in the length of the pattern and $\log_2(1/\delta)$. ∎

Finally, it should be noted that the number of rounds performed by our stochastic finite learner does *not* depend on the actual target to be learned but only on $p^*, p^*_{\rm sym}$ and $\log_2(1/\delta)$. Thus, though our definition of stochastic finite learning with high confidence is not requiring this additional feature; it can be achieved for the one-variable pattern languages. Thus, we have a further resemblance to the PAC model.

8. Test Results

Based on an implementation of the learning algorithm described and analysed above and variants of this algorithm we have run a large amount of tests. For each choice of an instance of the 1-variable pattern language learner, that is for each pair of pattern and probability distribution for substituting the pattern variable, 100 experiments have been conducted.

The experimental results are given as tables. These should be read as follows. Column 1 specifies the pattern. The 2. column gives the size s of the alphabet Σ used to replace the pattern variable. Σ was chosen as $\{a, b, \ldots\}$.

The 3. column specifies the details of the distribution. For *length-uniform distributions* the notation $[\nu, \mu]$ means that from the interval of natural numbers ranging from ν to μ each number was chosen with equal probability to be the length ℓ of the substitution. Then, according to the definition of length-uniform, with equal probability for all strings in Σ^ℓ one was selected.

For distributions with symmetries $[\nu, \mu]$ means that for the pattern variable x a string

$$w = y^k \, z \, y^k \in \Sigma^*$$

was randomly generated. Hereby, k is uniformly distributed in the interval $[\nu, \mu]$. Independently, the substrings y and z are selected length-uniformly. For the length of y the values from 1 to 5 are chosen with equal probability, for the length of z the values range from 0 to 5. However, in the case $k = 0$, that is $w = z$, z has to be nonempty in order to avoid empty

pattern	$\|\Sigma\|$	**distribution**	**average**	bound	dominance	maximum
		length-uniform				
varying the pattern length						
axb	4	[1,10]	2,56	4,17	2 (57%)	5 (4%)
$axbxab$	4	[1,10]	2,80	4,17	2 (48%)	6 (2%)
$abxbxxcxxdxaxxxbx$	4	[1,10]	2,50	4,17	2 (59%)	6 (1 %)
$axbaxb$	2	[1,10]	3,73	8,00	3 (29%)	9 (1 %)
$axxbxxxax$	2	[1,10]	4,16	8,00	4 (29%)	10 (1%)
$axbxxbaxbaxbaxx$ - $baxbxxbaxbaxb$	2	[1,10]	3,74	8,00	3 (36%)	8 (1 %)
varying the alphabet size						
$axxbxxxax$	3	[1,10]	3,04	5,00	2 (39%)	8 (1%)
$axxbxxxax$	4	[1,10]	2,74	4,17	2 (56%)	6 (2%)
$axxbxxxax$	10	[1,10]	2,34	3,35	2 (76%)	6 (1%)
varying the example length						
$axxbxxxax$	2	[5,5]	3,57	8,00	3 (38%)	12 (1%)
$axxbxxxax$	2	[10,10]	4,00	8,00	3 (27%)	8 (4%)
$axxbxxxax$	2	[20,20]	3,39	8,00	2 (33%)	9 (1%)
$axxbxxxax$	2	[11,20]	3,97	8,00	3 (35%)	12(1%)
very regular pattern						
$ax^{12}b$	2	[5,5]	3,92	8,00	3 (30%)	15 (1%)
$ax^{12}b$	2	[0,10]	3,66	8,00	3 (27%)	9 (1%)
$ax^{12}b$	4	[5,5]	2,68	4,17	2 (56%)	8 (1%)
$ax^{12}b$	4	[0,10]	2,63	4,17	2 (54%)	6 (1%)

Table 1: *Number of rounds for the learning algorithm to converge:*
length-uniform distributions

substitutions. The notation [3;4;6] means that the number k of symmetries is chosen equally likely among the values 3, 4 and 6.

Finally, for Markov chains, the substitution string w is generated by a random walk in the alphabet. In the uniform case the first symbol is chosen with equal probability $1/s$ among the elements of Σ. In the following steps of the random walk with probability $1/(s+1)$ either another letter from Σ is chosen or the walk terminates. It is easy to see the expected length of w equals $s+1$ in this uniform setting. We have also tested nonuniform random walks where some letters are much more likely to be the first, resp. the last symbol of w, while others may not occur at all. For the distributions named *skewed 1 to 4* see the appropriate figures below.

The distribution in figure 3, for example, has the property that substitutions beginning with the letters ac are very likely, while letter c will never occur at the beginning. At the end letter b is most likely, whereas a does not appear. Furthermore, a b will never be followed by a c.

In figure 7, in addition, the subsequence abc is most likely, that is a in most cases will be followed by a b and a b will be followed by a c. The distribution in figure 8 has the property that most substitutions start with an a, a b will appear somewhere in the middle with high

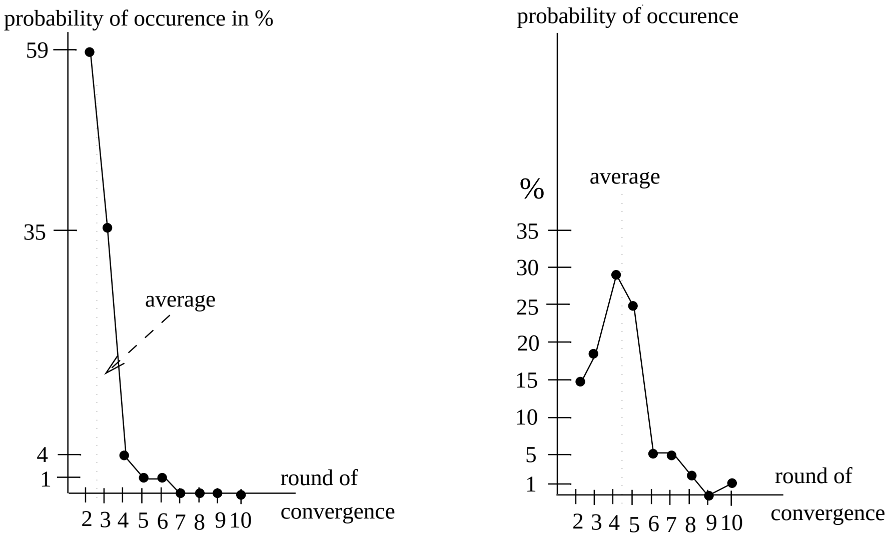

Figure 1: *Complete distribution for test instance 3 and 5 of table 1: on the left pattern $abxbxxcxxdxaxxxbx$ with the [1,10]-length-uniform distribution over the 4 letter alphabet, on the right pattern $axxbxxxax$ with the [1,10]-length-uniform distribution over 2 letters.*

probability, if a d occurs than this most likely terminates the substitution string, and for the letter c there is a good chance that it is followed by another c.

The 4. column gives the average number of examples the learning algorithm reads until its hypotheses have converged to the correct pattern, that is the average when observing the random variable G as defined in the proof of Lemma 14. Excluding the case of a trivial pattern without any variable, G is at least 2, and this value typically also occurs with high frequency.

The column labelled *bound* gives the numerical value of the upper bound estimation on G provided by Lemma 1 and 5. The 6. column labelled *dominance* gives the most frequent value of G that has occured and its frequency in paranthesis, while the last column shows the maximal value of G and its frequency among the 100 test runs for this particular instance of the problem. If two values have occured with the same frequency both values are listed (see the entries 2;3 in the table). For some cases we also give the complete distribution of G in the following figures.

These data allow the following interpretation. First of all, the average number of rounds tend to be less than our estimated guarantees, in some cases much less. This can be explained by the fact that considering the probability p_{sym} is quite pessimistic. To receive an example with a nonsymmetric substitution is a sufficient condition for the algorithm to converge, but not a necessary one. If $1/p_{\mathrm{sym}}$ is large the actual behaviour is significantly better than the upper bound derived from p_{sym}.

For alphabet size at least 3, the average of G is around 3 or less for uniform distributions. This value is relatively independent of the pattern – its structure and length – and whether short examples (obtained by single letter substitutions) occur or not – compare the length uniform case where substitutions of length 20 only perform better compared to the case [1,10] and [5,5] for example. Even highly symmetric patterns like $ax^{12}b$ are recognized very fast. Increasing the alphabet reduces the number of rounds slightly as should be expected.

For alphabets of size 2, however, about 1 additional example is needed, in particular for

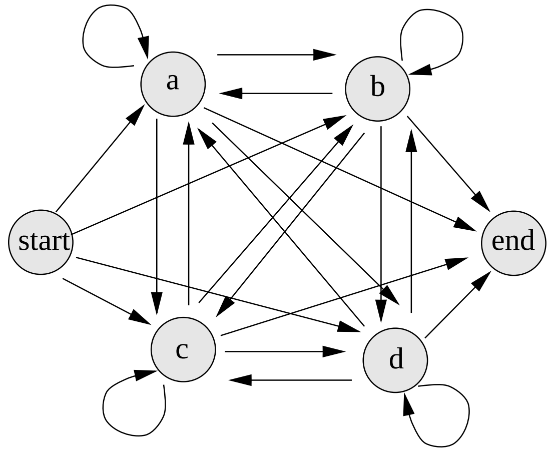

Figure 2: *Uniform Markov chain distribution for alpabet size 4*
In every node all outgoing edges have the same probability of being chosen.

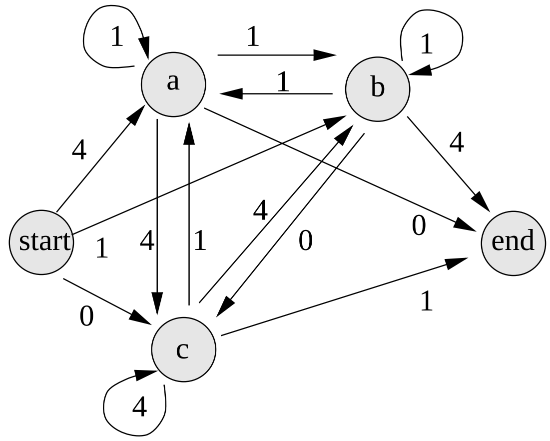

Figure 3: *Markov chain distribution of type skewed 1.*
The probability in a given node to chose a particular edge is given by its weight divided by the sum of all weights of edges leaving this node.

patterns like $axxbxxxax$, where it is difficult to locate the positions that represent constants in the pattern. From the testing of many different patterns we got the impression that this pattern with an alphabet of size 2 belongs to the most difficult ones for the learning algorithm.

For heavily skewed distributions the number of rounds increases, which has to be expected. Even then, among the several thousands of runs conducted the maximal value having ever occured was about 35 rounds, unless extremly biased distributions were chosen.

For this testing we have used an implementation of the algorithm that performs complete compatibility checking between pairs of strings. This increases the time within each round slightly if examples are highly symmetric, but will reduce the number of rounds. Even for certain distributions that generate only patterns with symmetries (the case [1,3] and [3;4;6]) this version is able to learn the pattern. For such cases our analysis above could not give any guarantee for convergence – indicated by the value ∞ in the column labelled *bound*.

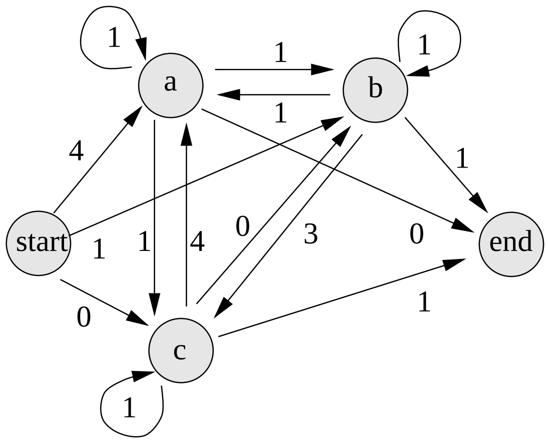

Figure 4: *Markov chain distribution of type skewed 2.*

pattern	$\lvert\Sigma\rvert$	**distribution**	**average**	bound	dominance	maximum
		symmetries				
$axxbxxxax$	2	[0,2]	3,74	7,00	3 (26%)	9 (1%)
$axxbxxxax$	2	[0,5]	3,57	10,00	3 (34%)	8 (1%)
$axxbxxxax$	2	[1,3]	4,03	∞	2;3 (23%)	11 (1%)
$axxbxxxax$	2	[3;4;6]	3,80	∞	3 (33%)	11 (1%)
$axxbxxxax$	4	[0,2]	3,74	7,00	3 (26%)	9 (1%)
$axxbxxxax$	4	[0,5]	3,57	10,00	3 (34%)	8 (1%)
$axxbxxxax$	4	[3;4;6]	2,69	∞	2 (56%)	8 (1%)
$ax^{12}b$	2	[0,2]	5,72	7,00	2 (21%)	21 (1%)
$ax^{12}b$	2	[0,5]	5,93	10,00	3 (25%)	25 (1%)
$ax^{12}b$	2	[1,3]	5,44	∞	3 (18%)	16 (2%)

Table 2: *Number of rounds till convergence:* **distributions with symmetries**

pattern	$\lvert\Sigma\rvert$	**distribution**	**average**	bound	dominance	maximum
		Markov chain				
$axxbxxxax$	2	uniform	3,84	8,00	3 (27%)	13 (1%)
$axxbxxxax$	4	uniform	2,70	4,17	2 (51%)	5 (4%)
$abxbxxcxxdxaxxxbx$	4	uniform	2,56	4,17	2 (56%)	4 (12%)
$axxbxxxax$	3	skewed 1	6,52	11,02	2 (24%)	33 (1%)
$axxbxxxax$	3	skewed 2	5,92	11,36	3 (17%)	16 (1%)
$axxbxxxax$	3	skewed 3	3,84	6,00	3 (24%)	9 (2%)
$axxbxxxax$	4	skewed 4	4,50	9,10	2 (29%)	14 (2%)

Table 3: *Number of rounds till convergence:* **Markov chain distributions**

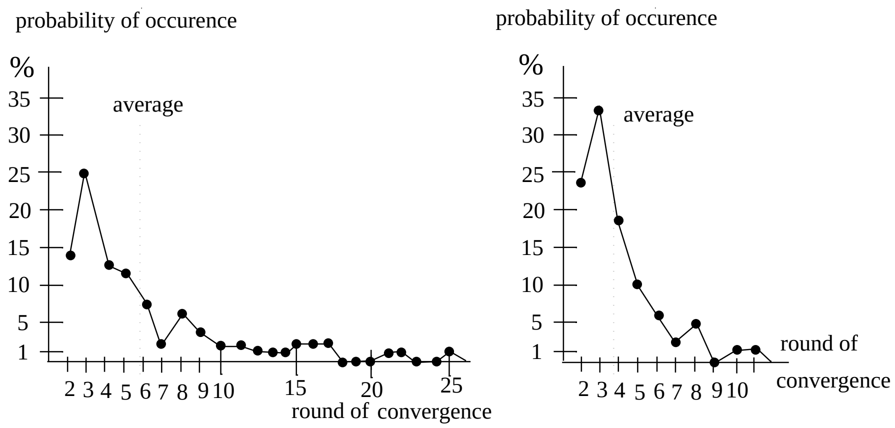

Figure 5: *Complete distribution for test instance 9 and 4 of table 2: on the left pattern* $ax^{12}b$ *for the distribution with a [0,5]-symmetry over the 2 letter alphabet, on the right pattern* $axxbxxxax$ *with a [3;4;6]-symmetry over 2 letters.*

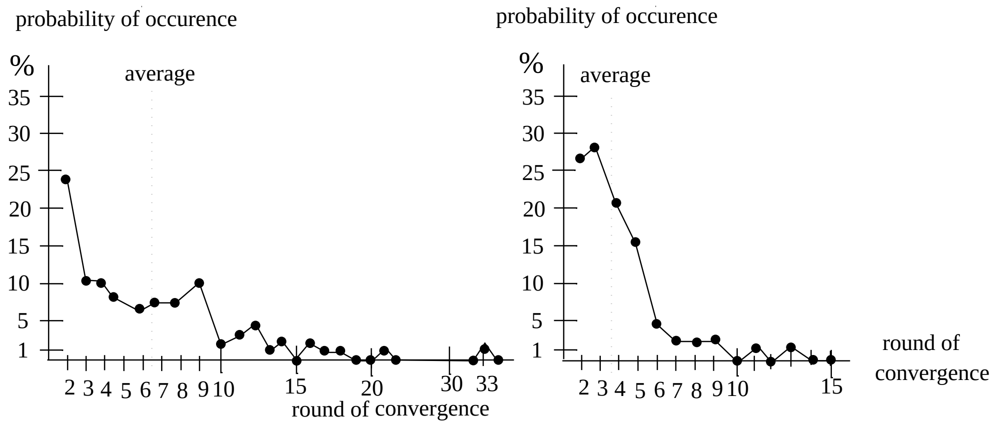

Figure 6: *Complete distribution for test instance 4 and 1 of table 3: for pattern* $axxbxxxax$ *on the left the uniform Markov chain over the 2 letter alphabet, on the right the distribution given in figure 5 for 3 letters.*

The last table lists experimental results for a modified version of the learning algorithm that can also handle empty substitutions. Due to disregarding the shortest example this modified version requires at least one more round. In addition, one has to take into account the probability for the learner to receive the empty substitution. We have included this delay in the upper bound estimations.

Now, in the length-uniform case the value 0 for the length parameter ℓ is also possible, and similarly for symmetric distributions. In the Markov chain model we add an edge that leads directly from the start node to the end node. The numerical data shown in table 4 shows the expected behaviour when comparing it to the case without empty substitutions.

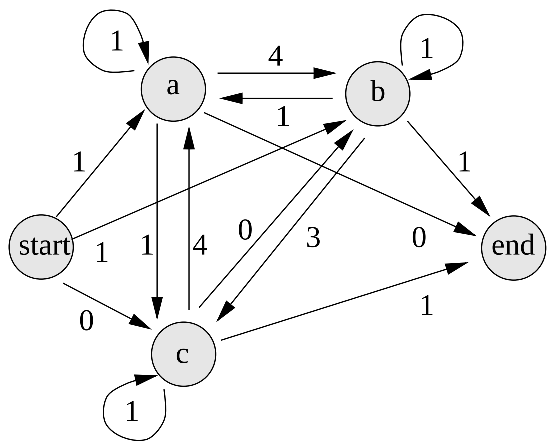

Figure 7: *Markov chain distribution of type skewed 3.*

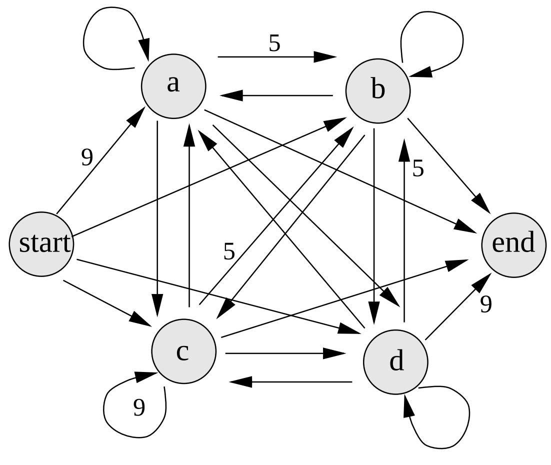

Figure 8: *Markov chain distribution of type skewed 4. Edges without a number have weight 1.*

pattern	$\lvert\Sigma\rvert$	**distribution**	**average**	bound	dominance	maximum
		length-uniform				
axb	2	[0,10]	4,92	9,09	4 (30%)	10 (2%)
$axbxab$	2	[0,10]	4,75	9,09	5 (27%)	9 (2%)
$axxbxxxax$	2	[0,10]	4,98	9,09	4 (25%)	12 (1%)
$axxbxxxax$	2	[0,5]	5,34	9,17	4 (25%)	17 (1%)
$axxbxxxax$	2	[0,20]	4,57	9,05	4 (34%)	12 (1%)
$axxbxxxax$	2	[10,10]	4,61	9,00	4 (37%)	11 (2%)
$axbxab$	4	[0,10]	3,58	5,26	3 (55%)	6 (2%)
$axxbxxxax$	4	[0,10]	3,82	5,26	3 (49%)	12 (1%)
$ax^{12}b$	2	[5,5]	4,97	9,00	5 (33%)	9 (4%)
$ax^{12}b$	2	[0,10]	4,86	9,09	4 (33%)	13 (1%)
$ax^{12}b$	4	[5,5]	3,68	5,17	3 (51%)	8 (1%)
$ax^{12}b$	4	[0,10]	3,71	5,26	3 (53%)	7 (1%)
		Markov chain				
$axxbxxxax$	2	uniform	5,30	9,33	4 (28%)	13 (1%)
$axxbxxxax$	4	uniform	3,72	5,37	3 (55%)	7 (2%)
$ax^{12}b$	4	uniform	3,73	5,37	3 (47%)	6 (2%)
$abxbxxcxxdxaxxxbx$	4	uniform	3,89	5,37	3 (42%)	7 (3%)
$axxbxxxax$	4	skewed 4	6,23	10,53	3;4 (26%)	22 (1%)
		symmetries				
$axxbxxxax$	2	[0,2]	4,68	8,07	4 (32%)	9 (2%)
$axxbxxxax$	2	[0,3]	4,71	11,05	3 (26%)	10 (1%)
$axxbxxxax$	2	[1,3]	4,78	∞	4 (32%)	12 (2%)
$axxbxxxax$	2	[3;4;6]	4,75	∞	3 (27%)	10 (2%)
$ax^{12}b$	2	[0,2]	5,03	8,07	4 (24%)	10 (1%)
$ax^{12}b$	2	[1,3]	4,62	∞	4 (29%)	12 (1%)

Table 4: *Number of rounds till convergence of the extended algorithm with complete compatibility checking and empty substitutions.*

9. CONCLUSIONS

We have shown that one-variable pattern languages are learnable for basically all meaningful distributions within an optimal linear total learning time on the average. The algorithm obtained is quite simple and is based on symmetries that occur in such languages. Thus, our approach to minimize the expected total learning time turned out to be quite satisfactory.

Additionally, our learner requires only space for its long and short term memory that is linear in the length of the target pattern. Therefore, it is not only faster than the algorithms presented by Angluin [1] and Erlebach *et al.* [6] but also more space-efficient. The only known algorithm using even less space is Lange and Wiehagen's [12] learner. But their algorithm is only successful for a much smaller class of probability distributions, since it requires shortest examples in order to converge. As a matter of fact, our algorithm does not need shortest examples at all to achieve convergence. Its convergence is quite independent of the substitution length, which is further confirmed by our experiments.

On the other hand, our learner can easily be modified to maintain the *incremental* behavior of Lange and Wiehagen's [12] algorithm. Instead of memorizing the pair $(\mathtt{PRE}, \mathtt{SUF})$, it can also store just the two or three examples from which $(\mathtt{PRE}, \mathtt{SUF})$ has been computed. While it is no longer *iterative*, it is still a *bounded example memory* learner. A learner is called iterative, if it uses only its last guess and the next example in the sequence of example strings for computing its actual hypothesis. A bounded example memory learner is additionally allowed to memorize an *a priori* bounded number of examples. For more information concerning these learning models, we refer the reader to Lange and Zeugmann [14].

Moreover, our algorithm does not only possess an expected linear total learning time, but also very good tail bounds. Note that, whenever learning in the limit is considered one cannot decide whether or not the learner has already converged to a correct hypothesis. If convergence is decidable, we arrive at *finite* learning. It is easy to see that one-variable pattern languages are *not* finitely learnable. On the other hand, a bit of prior knowledge about the underlying probability distributions nicely buys a *stochastically finite learner with high confidence* (cf. Theorem 3).

Note that stochastically finite learning with high confidence is different from PAC-learning. First, it is not completely distribution independent. Thus, from that perspective, this variant is weaker than the PAC-model. On the other hand, since the one-variable pattern languages are not PAC learnable (cf. [15]), one has to restrict the class of admissible probability distributions in one way or the other. Our restriction emerged quite naturally and comprises a huge class of probability distributions. Furthermore, the hypothesis computed is *exactly correct* with high probability. Moreover, the learner receives *exclusively* positive data while the correctness of its hypothesis is measured with respect to all data. Hence, from that perspective, our model of stochastically finite learning with high confidence is stronger than the PAC-model.

Our approach also differs from U-learnability introduced by Muggleton [16]. First of all, our learner is fed with positive examples only, while in Muggleton's [16] model examples labeled with respect to their containment in the target language are provided. Next, we do not make any assumption concerning the distribution of the target patterns. Furthermore, we do *not* measure the expected total learning time with respect to a given class of distributions over the targets and a given class of distributions for the sampling process, but exclusively in dependence on the length of the target. Finally, we require exact learning and not approximately correct learning.

Our implementation of the algorithm is available for public use through the WEB. The reader is referred to

http://www.itheoi.mu-luebeck.de/pages/reischuk/Algorithmen/PatLearn.html

for getting access to the resulting Java-applets.

Next, we shortly discuss possible directions of further research. An obvious extension would be to consider k-variable pattern languages for small fixed $k > 1$. Already for $k = 2$ the situation becomes considerably more complicated and requires additional tools.

Another direction to pursue would be to learn languages that are the union of at most ℓ one-variable pattern languages for some fixed ℓ.

Finally, the approach presented in this paper seems to be quite suited to tolerate errors in the example data. Let us assume that there is some (small) probability ϵ that

error model 1: in an example string $X[1] \ldots X[l]$ a symbol $X[i]$ is changed to a different one,

error model 2: $X[i]$ is changed to a different symbol or removed or replaced by two symbols $X[i]\sigma$ for some $\sigma \in \Sigma$.

A property of the pattern language like the common prefix of all strings now is only accepted if it is supported by a large percentage of examples. The details and modification of the algorithm will be given in another paper.

References

[1] D. Angluin, Finding Patterns common to a Set of Strings, *J. Comput. System Sci.* **21** (1980), 46–62.

[2] A. Blumer, A. Ehrenfeucht, D. Haussler and M. Warmuth, Learnability and the Vapnik-Chervonenkis Dimension, *J. ACM* **36** (1989), 929–965.

[3] J. Case, S. Jain, S. Lange and T. Zeugmann, Incremantal Concept Learning for Bounded Data Mining, *Information and Computation*, to appear.

[4] M. Crochmore and W. Rytter, "Text Algorithms," Oxford University Press, 1994.

[5] R. Daley and C.H. Smith. On the Complexity of Inductive Inference. *Inf. Control* **69** (1986), 12–40.

[6] T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger and T. Zeugmann, "Efficient Learning of One-Variable Pattern Languages from Positive Data," DOI-TR-128, Kyushu University, Fukuoka, Japan, 1996.

[7] 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, *in* "Proc. 8th Int. Workshop on Algorithmic Learning Theory" (M. Li and A. Maruoka, Eds.), Lecture Notes in Artificial Intelligence, Vol. 1316, pp. 260–276, Springer-Verlag, Berlin 1997.

[8] R. Freivalds, E. Kinber and C.H. Smith, On the Impact of Forgetting on Learning Machines, *J. ACM* **42** (1995), 1146–1168.

[9] M. Fulk, S. Jain and D. N. Osherson, Open Problems in Systems That Learn, *J. Comput. System Sci.* **49** (1994), 589–604.

[10] E. Gold, Language identification in the limit, *Inform. Control* **10** (1967), 447–474.

[11] M. Kearns and L. Pitt, A Polynomial-Time ALgorithm for Learning k-Variable Pattern Languages from Examples, *in* "Proc. 2nd Annual ACM Workshop on Computational Learning Theory" (R. Rivest, D. Haussler and M. K. Warmuth, Eds.), pp. 57–71, Morgan Kaufmann, San Mateo, CA, 1989.

[12] S. Lange and R. Wiehagen, Polynomial-Time Inference of Arbitrary Pattern Languages, *New Generation Computing* **8** (1991), 361–370.

[13] S. Lange and T. Zeugmann, Set-driven and Rearrangement-independent Learning of Recursive Languages, *Mathematical Systems Theory* **29** (1996), 599–634.

[14] S. Lange and T. Zeugmann, Incremental Learning from Positive Data, *J. Comput. System Sci.* **53**(1996), 88–103.

[15] A. Mitchell, A. Sharma and T. Scheffer, "Language Identification for Tourists on Polynomially Bounded Holidays," unpublished manuscript.

[16] S. Muggleton, Bayesian Inductive Logic Programming, *in* "Proc. 7th Annual ACM Conference on Computational Learning Theory" (M. Warmuth, Ed.), pp. 3–11, ACM Press, New York, 1994.

[17] L. Pitt, Inductive Inference, DFAs and Computational Complexity, *in* "Proce. 2nd Int. Workshop on Analogical and Inductive Inference" (K.P. Jantke, Ed.), Lecture Notes in Artificial Intelligence, Vol. 397, pp. 18–44, Springer-Verlag, Berlin, 1989.

[18] A. Salomaa, Patterns, (The Formal Language Theory Column), *EATCS Bulletin* **54** (1994), 46–62.

[19] A. Salomaa, Return to Patterns, (The Formal Language Theory Column), *EATCS Bulletin* **55** (1994), 144–157.

[20] T. Shinohara and S. Arikawa, Pattern Inference, *in* "Algorithmic Learning for Knowledge-Based Systems" (K. Jantke and S. Lange, Eds.), Lecture Notes in Artificial Intelligence, Vol. 961, pp. 259–291, Springer-Verlag, Berlin, 1995.

[21] L.G. Valiant, A Theory of the Learnable, *Commun. ACM* **27** (1984), 1134-1142.

[22] R. Wiehagen and T. Zeugmann, Ignoring Data may be the only Way to Learn Efficiently, *J. Experimental and Theoretical Artificial Intelligence* **6** (1994), 131–144.

[23] 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, to appear in *Annals of Mathematics and Artificial Intelligence.*