Learning One-Variable Pattern Languages in Linear Average Time

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

Thomas Zeugmann
Department of Informatics
Kyushu University
Fukuoka 812-81
Japan
thomas@i.kyushu-u.ac.jp

Abstract

A new algorithm for learning one-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.

1. Introduction

Patterns are a very natural way to define formal languages. Suppose you want to define the set of all strings having even length $2m$, for any m, such that the prefix of length m is identical to the suffix starting at position $m+1$. In that case, the wanted language follows the pattern $\pi = x\,x$. The formal definition of patterns and pattern languages goes back to Angluin [An80]. Since then, pattern languages and variations thereof have widely been investigated (cf., e.g., Salomaa [Sal94a, Sal94b], and Shinohara and Arikawa [ShA95] for an overview).

As far as learning theory is concerned, pattern languages are one of the most prominent examples of nonregular languages that can be learned in the limit from

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

positive data. Gold [Go67] has introduced the corresponding learning model. Let L be any formal language; then a *text* for L is any infinite sequence of strings that eventually contains all the 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 the target language L. Thereby the hypotheses are chosen from a prespecified set of *admissible hypotheses* called *hypothesis space*. Additionally, the sequence of hypotheses has to *converge* to a hypothesis correctly describing the target language (cf. Definition 1).

Looking at potential applications of limit learners, efficiency becomes a central issue. However, defining an appropriate measure of efficiency for learning in the limit is a difficult problem (cf. [Pi89]). Various authors have studied the efficiency of learning in terms of the so-called update time needed for computing a new single hypothesis. But what really counts in practical applications is the overall time needed by a learner until convergence, i.e., the so-called *total learning time*. Nevertheless, one can show that the total learning time is *unbounded* in the worst-case. Thus, we study the *expected* total learning time. For the purpose of motivation we shortly summarize what has been known in this regard.

Angluin [An80] describes an algorithm that learns the class of all pattern languages with an arbitrary number of different variables, which is based on the notion of *descriptive patterns*. However, no efficient algorithm is known for computing descriptive patterns. Hence, already the update time is practically infeasible. Moreover, finding a descriptive pattern of *maximum* possible length is known to be $\mathcal{NP}$-hard.

Therefore, one has also considered restricted versions of pattern language learning in which the number of different variables is fixed, in particular the case of a single variable. In the same paper, Angluin gives a learning algorithm for one-variable pattern languages with update time $O(L^4 \log L)$, where L denotes the sum of the length of all examples seen so far. Nothing is known concerning the expected total learning time of her algorithm.

Recently, Erlebach *et al.* [ERS96, ERS97] presented a one-variable learner that achieves an average total learning time $O(|\pi|^2 \log |\pi|)$, where $|\pi|$ is the length of the target pattern. Nevertheless, this result is also based on finding descriptive patterns quickly. While this approach has the advantage that at least the descriptiveness of every hypothesis produced can be guaranteed, it may also have the disadvantage of preventing the learner to achieve a better expected total learning time. Therefore, the question arises whether there exists a one-variable pattern language learner that achieves a subquadratic expected total learning. Clearly, the best one can hope to 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. We achieve this goal by studying combinatorial properties of strings generated by one-variable patterns. 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

By $\mathbb{N} = \{0, 1, 2, \ldots\}$ we denote the set of all natural numbers, and by $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. 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 $s := |\Sigma| \geq 2$ symbols. By Σ^* we denote the free monoid over Σ (cf. [HU69]). The set of all finite non-null strings of symbols from Σ is denoted by Σ^+, i.e., $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$, where ε denotes the empty string. Let x be a symbol not contained in Σ. Every string over $(\Sigma \cup \{x\})^+$ is called a one-variable pattern. We refer to x as the pattern variable. By *Pat* we denote the set of all one-variable patterns. Moreover, 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.,

$$w = w[1]\ w[2]\ \ldots\ w[\ell-1]\ w[\ell] = w[-\ell]\ w[-\ell+1]\ \ldots\ w[-2]\ w[-1].$$

Moreover, for $1 \leq i \leq j \leq \ell$ we denote the substring $w[i] \ldots w[j]$ of w by $w[i \ldots j]$. For $\pi \in Pat$ and $u \in \Sigma^+$ we denote by $\pi[x/u]$ 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

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

Furthermore, 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 defined as follows. 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 [Go67], there is also a major difference. In [Go67] the sequence $(X_i)_{i\in\mathbb{N}^+}$ is required to exhaust $L(\pi)$ in the limit, that is $\{X_i \mid i \in \mathbb{N}^+\} = L(\pi)$. 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 throughout this paper. The length of the pattern π to be learned is given by

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

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 [An80], 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 [An80, KP89, Sch90, WZ94], we like to 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 parameter n introduced above, i.e., with respect to the length of the target pattern. From a practical point of view we think this to be a more suitable concept than considering the *update time* and measuring the effort with respect to the examples received. Obviously, one is interested to learn an unknown pattern as fast as possible. If efficiency can be achieved by doing nothing and simply waiting till a long list of examples has been received from which one then can produce a hypothesis quickly this indicates that something must be wrong with the underlying complexity measure (cf. [Pi89]). 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. On the other hand, such a scenario is much too pessimistic for many applications, and hence a worst-case analysis is not appropriate. Therefore, one should consider the average-case behavior. Average-case complexity in general depends very much on the distribution over the input space. Restricting to some sort of uniform distributions might make the analysis easier, but such results may not have any implications for typical distributions that occur in practice.

Analyzing the expected total learning time of limit learners has been initialized by Zeugmann [Ze95] who studied Lange and Wiehagen's [LW91] pattern language learning algorithm. Their algorithm has an expected total learning time that is exponential in the number k of different variables occurring in the target pattern (cf. Zeugmann [Ze95]). Moreover, the point of convergence definitely depends on the appearance of sufficiently many *shortest* strings of the target language. On the one hand, shortest strings give in a certain sense maximal information about the pattern to be learned. On the other hand, assuming that variables are replaced independently by strings over the alphabet such shortest samples are rare or do not occur at all. Therefore, this algorithm seems to be of little practical use unless in the special situation when shortest samples are guaranteed to occur frequently.

For one-variable patterns, [ERS96] make a new approach by trying to estimate the number of occurrences of the pattern variable and the length of the substitution. This way, a total average learning time of order $n^2 \log n$ is achieved.

In this paper, we will more carefully analyze the combinatorics of words generated by a one-variable pattern to achieve an optimal result of linear total learning time on the average. It can even be shown that this linear bound holds with high probability. Let

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

denote the probability distribution that specifies 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.

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

denotes the range of Z, 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. Otherwise, if μ is chosen after the target pattern π has been specified, an adversary can always construct a distribution that makes it extremely difficult or even impossible to recognize this special pattern π.

What we will actually 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 will show that there exists an algorithm that efficiently learns every one-variable pattern on the average with respect to every distribution in $\mathcal{D}$.

Let $\boldsymbol{E}[|\boldsymbol{Z}|]$ denote the expectation of $|Z|$, that is 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

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

denote the language of all example strings that may actually occur, i.e., in π only substitutions are allowed that replace the variable x by a string from $\mathtt{Range}(Z)$.

3. Probabilistic Analysis of Substitutions

For obtaining most general results we would like to put as little constraints on the distribution μ as possible. For example, in previous work it was assumed that for every length all possible substitution strings are equally likely (cf. [ERS96]) or that all substitutions have a non-zero probability to occur (cf. [Ze95]). However, in real applications like genome analysis this will be hardly the case.

On the other hand, one cannot learn a target pattern if only example strings of a very restricted form occur. This will be the case if substitutions of the pattern variable are severely limited, in particular if $\mathtt{Range}(Z)$ 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 $\mathtt{Range}(Z) \subseteq L(\phi)$. Clearly, then the languages generated by

$$\pi \;=\; w_0 x^{u_0} w_1 x^{u_1} w_2 \ldots w_{m-1} x^{u_m} w_m$$

and

$$\pi' \ = \ w_0 \phi^{u_0} w_1 \phi^{u_1} w_2 \ldots w_{m-1} \phi^{u_m} w_m$$

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').

For excluding this case, let us define

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

and let us make

Assumption 2. $p_0 < 1$.

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. Otherwise, for some σ it would hold $\mathtt{Range}(Z) \subseteq L(\sigma x)$ or $\mathtt{Range}(Z) \subseteq L(x\sigma)$. 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 $p_0 \leq 1/s$ and $p_a = p_e = 1/s$.

In general, define

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

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

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

Define

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

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

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

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

∎

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 $y = y[1]y[2]\ldots y[\ell] \in \Sigma^+$ 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 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 also includes the special case $v_0 = \varepsilon$. In this case, since c and d are not uniquely determined in general, 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 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 y does not have other k'-symmetries for $k' \in \{1,\ \ldots,\ \max\{c,d\}\ mls(y)\}$.*

(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 $k = mls(y)$. This implies the bound on $sym(y)$. ∎

Assertion (4) of the latter lemma directly implies the simple bound

$$sym(y) \;\leq\; \frac{|y|}{2}\ ,$$

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, let us consider the expected number of symmetries. For the purpose of motivation of our Assumption 3, we first take a closer look at the length uniform case.

LEMMA 3. *In the length uniform case we have*

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

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 of the alphabet, it is independent of the length of the generated strings. 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

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

The estimation for $E[sym(Z)]$ above are 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

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

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 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

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

For efficiency reasons, symmetries and factorizations should be computed fast which is expressed by the following result.

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.

The second, and in the worst-case more efficient solution 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 and $r = \ell - \kappa$ the length of w. Then

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

which implies that w can be written as

$$w \;=\; w[1 \ldots \kappa]^{\lfloor r/\kappa \rfloor}\; w'\ ,$$

where w' of length $r \bmod \kappa$ is a prefix of w. Now define

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

Note that in the first and third case w_0 is the maximal symmetry of y, whereas in the second case $y = w[1 \ldots \kappa]^{\ell/\kappa}$ and the maximal symmetry has the form $w[1 \ldots \kappa]^{\lfloor \ell/2\kappa \rfloor}$.

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 [CR94], 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. Again, 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

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

Again, for illustration, in the length-uniform situation for p_{sym} one can easily derive the upper bound $p_{\text{sym}} \;\le\; \sum_{k\ge 1} s^{-k} \;\le\; (s-1)^{-1}$, which gives a significantly good estimation smaller than 1 except for $s=2$. In this case, by a more careful analysis one can show the bound $p_{\text{sym}} \le 3/4$.
In general, it will be required that the distribution is not restricted to substitutions with symmetries – with positive probability also nonsymmetric substitutions should occur.

***Assumption* 4.** $p_{\text{sym}} < 1$.

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,

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

Define

$$q(\mathcal{Z}) \;:=\; \min\{g \mid \neg Q_g[\mathcal{Z}]\} \;.$$

Similarly to Lemma 1, one can show

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

5. Basic Subroutines: Factorizations and Compatibility

For a subset A of Σ^* let $\texttt{PRE}(A)$ and $\texttt{SUF}(A)$ denote the maximal common prefix and suffix of all strings in A, respectively. Furthermore, 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.

We will store the current information about the prefix and suffix in the two variables PRE and SUF. All the remaining pattern learning is done with respect to the current value of these variables. If the algorithm sees a new sample string X such that $\mathtt{PRE}(\{X, \mathrm{PRE}\}) \neq \mathrm{PRE}$ or $\mathtt{SUF}(\{X, \mathrm{SUF}\}) \neq \mathrm{SUF}$ 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 $(\mathrm{PRE}, \mathrm{SUF})$–***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 positive integers c_1, d_1.

If such a representation is not possible for a given pair $(\mathrm{PRE}, \mathrm{SUF})$ then Y is said to have no $(\mathrm{PRE}, \mathrm{SUF})$–factorization.

Y' may have other symmetries $u_2, u_3, \ldots$ which give rise to factorizations

$$Y \;=\; \mathrm{PRE}\; u_i^{c_i}\; v_i\; u_i^{d_i}\; \mathrm{SUF}$$

for positive integers c_i, d_i. 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* $(\mathrm{PRE}, \mathrm{SUF})$–*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* $(\mathrm{PRE}, \mathrm{SUF})$–*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* $\pi = w_0\, x\, v\, w_m$ *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* $(\mathrm{PRE}, \mathrm{SUF})$–*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* $(\mathrm{PRE}, \mathrm{SUF})$–*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 $(\mathrm{PRE}, \mathrm{SUF})$ if from their minimal $(\mathrm{PRE}, \mathrm{SUF})$–factorizations a single pattern $\boldsymbol{\psi} = \boldsymbol{\psi}(\boldsymbol{Y}, \tilde{\boldsymbol{Y}})$ 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.

This compatibility notion is extended to pairs consisting of a string Y and a pattern π. Y is directly compatible to π with respect to $(\mathrm{PRE}, \mathrm{SUF})$ if for the minimal symmetry u_1 of the $(\mathrm{PRE}, \mathrm{SUF})$–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* $(\mathrm{PRE}, \mathrm{SUF})$ *–factorization is based equals* u. *Thus,* Y *is directly compatible to* π.

Proof. It is easy to see that for a nonsymmetric 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.*
Furthermore, 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 that $(\text{PRE}, \text{SUF}) = (w_0, w_m)$ has 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.

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 $\tilde{u}$ (which will not be minimal if $\tilde{u}$ 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_{mid} to $\tilde{Y}_{\text{mid}}$ after the first $\alpha_1 - 1$ occurrences of u in Y_{mid} have been read and matched against occurrences of $\tilde{u}$ in $\tilde{Y}_{\text{mid}}$ 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 $\tilde{Y}_{\text{mid}}$ and u is not a prefix of $\tilde{u}$. The same holds for the other occurrences of u in Y.

Given the corresponding factorizations, the checking whether Y_{mid} and $\tilde{Y}_{\text{mid}}$ 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

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

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. 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 briefly describe the global strategy of the learning procedure. In the trivial case where the pattern is just a constant $\pi = w$ all sample strings will be equal 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* $\boldsymbol{\psi_0} := \mathrm{PRE}_g\ x\ \mathrm{SUF}_g$.

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, otherwise 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 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 sample X_g;

if $X_g = \psi$ **then** output ψ,

else
PRE := $\mathtt{PRE}(\{\mathrm{PRE}, X_g\})$; SUF := $\mathtt{SUF}(\{\mathrm{SUF}, X_g\})$;

if PRE $\neq$ PRE$'$ or SUF $\neq$ SUF$'$ **then** (a new phase begins)
compute the minimal (PRE, SUF)–factorization of Y ;
$\psi_0 \; := \;$ PRE x SUF ;
$\psi := \psi_0$ **endif**;

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

case 1: Y does not have a (PRE, SUF)–factorization
then output ψ_0 :
case 2: X_g does not have a (PRE, SUF)–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 shorter 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 output ψ_0 .

7. Proof of Correctness

Since the sample 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 ($p < 1$) 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 samples 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 a sample 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 sample 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 sample generated by a nonsymmetric substitution.*

8. Complexity Analysis

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 sample sequence $\mathcal{X}$, and recall Z and X defined as random variables for the substitutions, resp. samples.

LEMMA 13. *For each round g it holds*

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

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

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

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

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

∎

Now we can also bound the total learning time.

LEMMA 14. *The total expected 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,

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

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

$$\begin{aligned} Time_{total}(\mathcal{X}) &= \sum_{g=1}^{G} Time_g(\mathcal{X}) = \sum_{t \ge 2} \Pr[G = t] \cdot \sum_{g=1}^{t} Time_g(\mathcal{X}) \qquad \text{and} \\ E[Time_{total}(\mathcal{X})] &= E[\sum_{t \ge 2} \Pr[G = t] \cdot \sum_{g=1}^{t} Time_g(\mathcal{X})] \end{aligned}$$

$$\begin{aligned}
&\leq O\left(E[\sum_{t\geq 2}\Pr[G=t] \;\cdot\; t\cdot E[|X|\cdot(1+E[sym(Z)])\right) \\
&\leq O\left(E[|X|\cdot(1+E[sym(Z)])\cdot\sum_{t}\Pr[G=t]\cdot t\right) \\
&= O\Big(E[|X|\cdot(1+E[sym(Z)])\cdot E[G]\Big) \\
&\leq O\left(n\cdot E[|Z|]\cdot(1+E[sym(Z)])\cdot\Big(\frac{1}{1-p}+\frac{1}{1-p_{\mathrm{sym}}}\Big)\right) \\
&= O(n)\ .
\end{aligned}$$

▮

Summarizing we can state

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.*

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.

9. Conclusions

We have shown that one-variable pattern languages are learnable for basically all meaningful distributions in optimal linear time on the average. The algorithm obtained is quite simple and is based on symmetries that occur in such languages. Thus, our appraoch to minimize the expected total learning time turned out to be quite satisfactory. 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. Therefore, knowing the probability of failure decreases exponentially a potential user of our algorithm can easily gain an exponentially growing confidence into the correctness of the actual hypothesis received.

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 sample data. Let us assume that there is some (small) probability ϵ that

error model 1: in a sample string $X[1] \dots X[l]$ 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 samples. The details and modification of the algorithm will be given in another paper.

References

[An80] D. Angluin. *Finding Patterns Common to a Set of Strings,* Journal of Computer and System Sciences 21(1):46–62, 1980.

[CR94] M. Crochmore, W. Rytter. *Text Algorithms,* Oxford University Press, 1994.

[ERS96] T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger and T. Zeugmann. *Efficient learning of one-variable pattern languages from positive data,* Technical Report DOI-TR-128, Kyushu University, Department of Informatics, Fukuoka, Japan, December 1996.

[ERS97] 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 International Workshop on Algorithmic Learning Theory - ALT'97," (M. Li and A. Maruoka, Eds.), Lecture Notes in Artificial Intelligence 1316, pages 260–276, Berlin 1997, Springer-Verlag.

[Go67] E. M. Gold. *Language identification in the limit,* Inf. & Control 10:447–474, 1967.

[HU69] J. E. Hopcroft and J. D. Ullman. *Formal Languages and their Relation to Automata,* Addison-Wesley Publishing Company, Reading, Massachusetts, 1969.

[KP89] M. Kearns and L. Pitt. *A polynomial-time algorithm for learning k-variable pattern languages from examples,* Proc. of the 2nd Ann. Workshop on Computational Learning Theory, COLT'89 (Santa Cruz, CA), pages 57–71, San Mateo, CA, 1989. Morgan Kaufmann.

[LW91] S. Lange and R. Wiehagen. *Polynomial-time inference of arbitrary pattern languages,* New Generation Computing 8:361–370, 1991.

[LZ96] S. Lange and T. Zeugmann. *Set-driven and rearrangement-independent learning of recursive languages,* Mathematical Systems Theory 29(6):599–634, 1996.

[Pi89] L. Pitt. *Inductive inference, DFAs and computational complexity,* In K.P. Jantke (Ed.), Proc. Analogical and Inductive Inference - AII'89, LNAI 397, pp. 18–44, Berlin, 1989, Springer-Verlag.

[Sal94a] A. Salomaa. *Patterns (The Formal Language Theory Column),* EATCS Bulletin 54:46 – 62, 1994.

[Sal94b] A. Salomaa. *Return to patterns (The Formal Language Theory Column),* EATCS Bulletin 55:144 – 157, 1994.

[Sch90] R. E. Schapire. *Pattern languages are not learnable,* in M. Fulk and J. Case (Eds.), Proc. of the 3rd Ann. Workshop on Computational Learning Theory, COLT'90 (Rochester, NY), pages 122–129, San Mateo, CA, 1990. Morgan Kaufmann.

[ShA95] T. Shinohara and S. Arikawa. *Pattern inference,* in "Algorithmic Learning for Knowledge-Based Systems," (K. P. Jantke and S. Lange, Eds.) Lecture Notes in Artificial Intelligence 961, pages 259 – 291, Berlin, 1995. Springer-Verlag.

[WZ94] R. Wiehagen and T. Zeugmann. *Ignoring data may be the only way to learn efficiently,* Journal of Experimental and Theoretical Artificial Intelligence 6(1):131–144, 1994.

[Ze95] T. Zeugmann. *Lange and Wiehagen's pattern language learning algorithm: An average-case analysis with respect to its total learning time,* Technical Report, Kyushu University, Department of Informatics, Fukuoka, Japan, 1995, to appear in Annals of Mathematics and Artificial Intelligence.