Learning a Subclass of Regular Patterns in Polynomial Time

John Case[1,⋆], Sanjay Jain[2,⋆⋆], Rüdiger Reischuk[3], Frank Stephan[4], and Thomas Zeugmann[3]

[1] Dept. of Computer and Information Sciences, University of Delaware, Newark, DE 19716-2586, USA
case@cis.udel.edu

[2] School of Computing, National University of Singapore, Singapore 117543
sanjay@comp.nus.edu.sg

[3] Institute for Theoretical Informatics, University at Lübeck, Wallstr. 40, 23560 Lübeck, Germany
{reischuk, thomas}@tcs.mu-luebeck.de

[4] Mathematisches Institut, Universität Heidelberg, Im Neuenheimer Feld 294, 69120 Heidelberg, Germany
fstephan@math.uni-heidelberg.de

Abstract. Presented is an algorithm (for learning a subclass of erasing regular pattern languages) which can be made to run with arbitrarily high probability of success on extended regular languages generated by patterns π of the form $x_0\alpha_1x_1...\alpha_mx_m$ for unknown m but known c, from number of examples polynomial in m (and exponential in c), where $x_0, \ldots, x_m$ are variables and where $\alpha_1, ..., \alpha_m$ are each strings of constants or terminals of length c. This assumes that the algorithm randomly draws samples with natural and plausible assumptions on the distribution.
The more general looking case of extended regular patterns which alternate between a variable and fixed length constant strings, beginning and ending with either a variable or a constant string is similarly handled.

1 Introduction

The *pattern languages* were formally introduced by Angluin [1]. A *pattern language* is (by definition) one generated by all the positive length substitution instances in a pattern, such as, *for example*,

$$abxycbbzxa$$

— where the variables (for substitutions) are x, y, z and the constants/terminals are a, b, c.

⋆ Supported in part by NSF grant number CCR-0208616 and USDA IFAFS grant number 01-04145.

⋆⋆ Supported in part by NUS grant number R252-000-127-112.

Since then, much work has been done on pattern languages and *extended pattern languages* which also allow empty substitutions as well as on various special cases of the above (cf., e.g., [1, 6, 7, 10, 12, 21, 20, 22, 23, 26, 19, 29] and the references therein). Furthermore, several authors have also studied finite unions of pattern languages (or extended pattern languages), unbounded unions thereof and also of important subclasses of (extended) pattern languages (see, for example, [11, 5, 27, 3, 32]).

Nix [18] as well as Shinohara and Arikawa [28, 29] outline interesting applications of pattern inference algorithms. For example, pattern language learning algorithms have been successfully applied toward some problems in molecular biology (see [25, 29]). Pattern languages and finite unions of pattern languages turn out to be subclasses of Smullyan's [30] Elementary Formal Systems (EFSs), and Arikawa, Shinohara and Yamamoto [2] show that the EFSs can also be treated as a logic programming language over strings. The investigations of the learnability of subclasses of EFSs are interesting because they yield corresponding results about the learnability of subclasses of logic programs. Hence, these results are also of relevance for Inductive Logic Programming (ILP) [17, 13, 4, 15]. Miyano *et al.* [16] intensively studied the polynomial-time learnability of EFSs.

In the following we explain the main philosophy behind our research as well as the ideas by which it emerged. As far as learning theory is concerned, pattern languages are a prominent example of non-regular languages that can be learned in the limit from positive data (cf. [1]). Gold [9] has introduced the corresponding learning model. Let L be any language; then a *text* for L is any infinite sequence of strings containing eventually all strings of L, and nothing else. The information given to the learner are successively growing initial segments of a text. Processing these segments, the learner has to output *hypotheses* about L. The hypotheses are chosen from a prespecified set called *hypothesis space*. The sequence of hypotheses has to *converge* to a correct description of the target language.

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 having this property generates a proper subset of the language generated by π. But no efficient algorithm is known for computing descriptive patterns. Thus, unless such an algorithm is found, it is even infeasible to compute a single hypothesis in practice by using this approach.

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 a learner for one-variable pattern languages with update time $O(\ell^4 \log \ell)$, where ℓ is the sum of the length of all examples seen so far. Note that this algorithm is also computing descriptive patterns even of maximum length.

Another important special case extensively studied are the *regular pattern languages* introduced by Shinohara [26]. These are generated by the *regular patterns*, i.e., patterns in which each variable that appears, appears only once. The learners designed by Shinohara [26] for regular pattern languages and extended regular pattern languages are also computing descriptive patterns for the data seen so far. These descriptive patterns are computable in time polynomial in the length of all examples seen so far.

But when applying these algorithms in practice, another problem comes into play, i.e., all the learners mentioned above are only known to converge in the limit to a correct hypothesis for the target language. But the *stage of convergence* is not decidable. Thus, a user never knows whether or not the learning process is already finished. Such an uncertainty may not be tolerable in practice.

Consequently, one has tried to learn the pattern languages within Valiant's [31] PAC model. Shapire [24] could show that the whole class of pattern languages is not learnable within the PAC model unless $\mathcal{P}/poly = \mathcal{NP}/poly$ for any hypothesis space that allows a polynomially decidable membership problem. Since membership is $\mathcal{NP}$-complete for the pattern languages, his result does not exclude the learnability of all pattern languages in an extended PAC model, i.e., a model in which one is allowed to use the set of all patterns as hypothesis space.

However, Kearns and Pitt [10] have established a PAC learning algorithm for the class of all k-variable pattern languages, i.e., languages generated by patterns in which at most k different variables occur. Positive examples are generated with respect to arbitrary product distributions while negative examples are allowed to be generated with respect to any distribution. Additionally, the length of substitution strings has been required to be polynomially related to the length of the target pattern. Finally, their algorithm uses as hypothesis space all unions of polynomially many patterns that have k or fewer variables[5]. The overall learning time of their PAC learning algorithm is polynomial in the length of the target pattern, the bound for the maximum length of substitution strings, $1/\varepsilon$, $1/\delta$, and $|\Sigma|$. The constant in the running time achieved depends *doubly exponential* on k, and thus, their algorithm becomes rapidly impractical when k increases.

As far as the class of extended regular pattern languages is concerned, Miyano *et al.* [16] showed the consistency problem to be $\mathcal{NP}$-complete. Thus, the class of all extended regular pattern languages is not polynomial-time PAC learnable unless $\mathcal{RP} = \mathcal{NP}$ for any learner that uses the regular patterns as hypothesis space.

This is even true for REGPAT_1, i.e., the set of all extended regular pattern languages where the length of constant strings is 1 (see below for a formal

[5] More precisely, the number of allowed unions is at most $poly(|\pi|, s, 1/\varepsilon, 1/\delta, |\pm|)$, where π is the target pattern, s the bound on the length on substitution strings, ε and δ are the usual error and confidence parameter, respectively, and $\pm$ is the alphabet of constants over which the patterns are defined.

definition). The latter result follows from [16] via an equivalence proof to the common subsequence languages studied in [14].

In the present paper we also study the special cases of learning the *extended regular pattern languages*. On the one hand, they already allow non-trivial applications. On the other hand, it is by no means easy to design an efficient learner for these classes of languages as noted above. Therefore, we aim to design an efficient learner for an interesting subclass of the extended regular pattern languages which we define next.

Let $\mathrm{Lang}(\pi)$ be the extended pattern language generated by pattern π. For $c > 0$, let REGPAT_c be the set of all $\mathrm{Lang}(\pi)$ such that π is a pattern of the form $x_0\alpha_1x_1\alpha_2x_2\ldots\alpha_mx_m$, where each α_i is a string of terminals of length c and $x_0, x_1, x_2, \ldots, x_m$ are distinct variables.

We consider polynomial time learning of REGPAT_c for various data presentations and for natural and plausible probability distributions on the input data. As noted above, even REGPAT_1 is not polynomial-time PAC learnable unless $\mathcal{RP} = \mathcal{NP}$. Thus, one has to restrict the class of all probability distributions. Then, the conceptional idea is as follows.

We explain it here for the case mainly studied in this paper, learning from text (in our above notation). One looks again at the whole learning process as learning in the limit. So, the data presented to the learner are growing initial segments of a text. But now, we do not allow any text. Instead every text is drawn according to some fixed probability distribution. Next, one determines the *expected* number of examples needed by the learner until convergence. Let E denote this expectation. Assuming prior knowledge about the underlying probability distribution, E can be expressed in terms the learner may use conceptionally to calculate E. Using Markov's inequality, one easily sees that the probability to exceed this expectation by a factor of t is bounded by $1/t$. Thus, we introduce, as in the PAC model, a confidence parameter δ. Given δ, one needs roughly $(1/\delta) \cdot E$ many examples to converge with probability at least $1 - \delta$. Knowing this, there is of course no need to compute any intermediate hypotheses. Instead, now the learner firstly draws as many examples as needed and then it computes just one hypothesis from it. This hypothesis is output, and by construction we know it to be *correct* with probability at least $1 - \delta$. Thus, we arrive at a learning model which we call *probabilistically exact learning* (cf. Definition 5 below). Clearly, in order to have an efficient learner one also has to ensure that this hypothesis can be computed in time polynomial in the length of all strings seen. For arriving at an overall polynomial-time learner, it must be also ensured that E is polynomially bounded in a suitable parameter. We use the number of variables occurring in the regular target pattern, c (the length of substitution strings) and a term describing knowledge about the probability distribution as such a parameter.

For REGPAT_c, we have results for three different models of data presentation. The data are drawn according to the distribution *prob* described below.

The three models are as follows. Thanks to space limitations we present herein the details and verification of our algorithm for the first model only. The journal version of this paper will present more details. Σ is the terminal alphabet. For natural numbers $c > 0$, Σ^c is Σ^* restricted to strings of length c.

(1) For drawing of examples according to *prob* for learning a pattern language generated by π: one draws terminal string σ according to distribution *prob* over Σ^* until $\sigma \in \text{Lang}(\pi)$ is obtained. Then σ is returned to the learner.

(2) One draws σ according to *prob* and gives $(\sigma, \chi_{\text{Lang}(\pi)}(\sigma))$ to the learner.

(3) As in (2), but one gives σ to the learner in the case that $\sigma \in \text{Lang}(\pi)$, and gives a pause-symbol to the learner otherwise.

For this paper, the natural and plausible assumptions on *prob* are the following.

(i) $prob(\Sigma^c) \geq prob(\Sigma^{c+1})$ for all c;

(ii) $prob(\sigma) = \frac{prob(\Sigma^c)}{|\Sigma^c|}$, where $\sigma \in \Sigma^c$.

(iii) there is an increasing polynomial *pol* such that $prob(\Sigma^c) \geq \frac{1}{pol(c)}$ for all c.

Our algorithm is presented in detail in Section 3 below. The complexity bounds are described more exactly there, but, basically, the algorithm can be made to run with arbitrarily high probability of success on extended regular languages generated by patterns π of the form $x_0\alpha_1x_1...\alpha_mx_m$ for unknown m but known c, from number of examples polynomial in m (and exponential in c), where $\alpha_1, ..., \alpha_m \in \Sigma^c$.

N.B. Having our patterns defined as starting and ending with variables is not crucial (since one can handle patterns starting or ending with constants easily by just looking at the data and seeing if they have a common suffix or prefix). Our results more generally hold for patterns alternating variables and fixed length constant strings, where the variables are not repeated. Our statements above and in Section 3 below involving variables at the front and end is more for ease of presentation of proof.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote the set of natural numbers, and let $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. For any set S, we write $|S|$ to denote the cardinality of S.

Let Σ be any non-empty finite set of constants such that $|\Sigma| \geq 2$ and let V be a countably infinite set of variables such that $\Sigma \cap V = \emptyset$. By Σ^* we denote the free monoid over Σ. The set of all finite non-null strings of symbols from Σ is denoted by Σ^+, i.e., $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$, where λ denotes the empty string. As above, Σ^c denotes the set of strings over Σ with length c. We let $a, b, \ldots$

range over constant symbols. $x, y, z, x_1, x_2, \ldots$ range over variables. Following Angluin [1], we define patterns and pattern languages as follows.

Definition 1. A *term* is an element of $(\Sigma \cup V)^*$. A *ground term* (or a *word*, or a *string*) is an element of Σ^*. A *pattern* is a non-empty term.

A *substitution* is a homomorphism from terms to terms that maps each symbol $a \in \Sigma$ to itself. The image of a term π under a substitution θ is denoted $\pi\theta$. We next define the language generated by a pattern.

Definition 2. The *language generated by a pattern* π is defined as $\mathrm{Lang}(\pi) = \{\pi\theta \in \Sigma^* \mid \theta \text{ is a substitution}\}$.

We set $\mathrm{PAT} = \{\mathrm{Lang}(\pi) \mid \pi \text{ is a pattern}\}$.

Note that we are considering extended (or erasing) pattern languages, i.e., a variable may be substituted with the empty string λ. Though allowing empty substitutions may seem a minor generalization, it is not. Learning erasing pattern languages is more difficult for the case considered within this paper than learning non-erasing ones. For the general case of arbitrary pattern languages, already Angluin [1] showed the non-erasing pattern languages to be learnable from positive data. However, the erasing pattern languages are *not* learnable from positive data if $|\Sigma| = 2$ (cf. Reidenbach [19]).

Definition 3 (Shinohara[26]). A pattern π is said to be *regular* if it is of the form $x_0\alpha_1x_1\alpha_2x_2\ldots\alpha_mx_m$, where $\alpha_i \in \Sigma^+$ and x_i is the i-th variable.

We set $\mathrm{REGPAT} = \{\mathrm{Lang}(\pi) \mid \pi \text{ is a regular pattern}\}$.

Definition 4. Suppose $c \in \mathbb{N}^+$. We define

(a) $\mathrm{reg}_c^m = \{\pi \mid \pi = x_0\alpha_1x_1\alpha_2x_2\ldots\alpha_mx_m, \text{ where each } \alpha_i \in \Sigma^c\}$.

(b) $\mathrm{reg}_c = \bigcup_m \mathrm{reg}_c$.

(c) $\mathrm{REGPAT}_c = \{\mathrm{Lang}(\pi) \mid \pi \in \mathrm{reg}_c\}$.

Next, we define the learning model considered in this paper. As already explained in the Introduction, our model differs to a certain extent from the PAC model introduced by Valiant [31] which is distribution independent. In our model, a bit of background knowledge concerning the class of allowed probability distributions is allowed. So, we have a stronger assumption, but also a stronger requirement, i.e., instead of learning an approximation for the target concept, our learner is required to learn it exactly. Moreover, the class of erasing regular pattern languages is known *not* to be PAC learnable (cf. [16] and the discussion within the Introduction).

Definition 5. A learner M is said to *probabilistically exactly learn* a class $\mathcal{L}$ of pattern languages according to probability distribution *prob*, if for all δ, $0 < \delta < 1$, for some polynomial q, when learning a $\text{Lang}(\pi) \in \mathcal{L}$, with probability at least $1 - \delta$, M draws at most $q(|\pi|, \frac{1}{\delta})$ examples according to the probability distribution *prob*, and then outputs a pattern π', such that $\text{Lang}(\pi) = \text{Lang}(\pi')$.

As far as drawing of examples according to *prob* for learning a pattern language generated by π is concerned, we assume the following model (the first model discussed in the Introduction): one draws σ according to distribution *prob* over Σ^*, until $\sigma \in \text{Lang}(\pi)$ is obtained. Then σ is returned to the learner. (Note: *prob* is thus defined over Σ^*.)

The other two models we mentioned in the Introduction are:

(2) There is a basic distribution *prob* and one draws σ according to *prob* and gives $(\sigma, \chi_{\text{Lang}(\pi)}(\sigma))$ to the learner.

(3) As in (2), but one gives σ to the learner in the case that $\sigma \in \text{Lang}(\pi)$, and gives a pause-symbol to the learner otherwise.

We note that our proof works for models (2) and (3) above too.

For this paper, the assumptions on *prob* are (as in the Introduction) the following.

(i) $prob(\Sigma^c) \geq prob(\Sigma^{c+1})$ for all $c \in \mathbb{N}$;

(ii) $prob(\sigma) = \frac{prob(\Sigma^c)}{|\Sigma^c|}$, where $\sigma \in \Sigma^c$.

(iii) there is an increasing polynomial *pol* with $prob(\Sigma^c) \geq \frac{1}{pol(c)}$ and $pol(c) \neq 0$ for all $c \in \mathbb{N}$.

3 Main Result

In this section we will show that REGPAT_c is probabilistically exactly learnable according to probability distributions *prob* satisfying the constraints described above.

Lemma 1. *(based on Chernoff Bounds) Suppose $X, Y \subseteq \Sigma^*$, δ, ϵ are properly between 0 and 1/2, and $prob(X) \geq prob(Y) + \epsilon$. Let e be the base of natural logarithm. Then, if one draws at least*

$$\frac{2}{\epsilon^2} * \frac{-\log(\delta)}{\log e}$$

many examples from Σ^* according to the probability distribution prob, then with probability at least $1 - \delta$, more elements of X than of Y show up. The number $\frac{2}{\epsilon^2 * \delta}$ is an upper bound for this number.

More generally, the following holds.

Lemma 2. *One can define a function* $r(\epsilon, \delta, k)$ *which is polynomial in* $k, \frac{1}{\epsilon}, \frac{1}{\delta}$ *such that for all sets* $X, Z, Y_1, Y_2, \ldots, Y_k \subseteq \Sigma^*$ *, the following holds.*

If $prob(X) - prob(Y_i) \geq \epsilon$ *, for* $i = 1, 2, \ldots, k$ *, and* $prob(Z) \geq \epsilon$ *, and one draws* $\geq r(\epsilon, \delta, k)$ *many examples from* Σ^* *according to the distribution* $prob$ *, then with probability at least* $1 - \delta$

(a) *there is at least one example from* Z *.*

(b) *there are strictly more examples in* X *than in any of the sets* $Y_1, ..., Y_k$ *.*

Proposition 1. *For every regular pattern* π *and all* $m \in \mathbb{N}$ *,* $\text{Lang}(\pi) \cap \Sigma^{m+1} \geq |\Sigma| * (\text{Lang}(\pi) \cap \Sigma^m)$ *.*

Proof. Since any regular pattern π has a variable at the end, the proposition follows. ∎

Proposition 2. *For any fixed constant* $c \in \mathbb{N}^+$ *and any alphabet* Σ *, there is a polynomial* f *such that for every* $\pi \in \text{reg}_c^m$ *, at least half of the strings of length* $f(m)$ *are generated by* π *.*

Proof. Suppose that $\pi = x_0\alpha_1x_1\alpha_2x_2 \ldots \alpha_mx_m$, and $\alpha_1, \alpha_2, ..., \alpha_m \in \Sigma^c$.

Clearly, there is a length $d \geq c$ such that for every $\tau \in \Sigma^c$, at least half of the strings in Σ^d contain τ as a substring, that is, are in the set $\bigcup_{k=0}^{d-c} \Sigma^k \tau \Sigma^{d-k-c}$.

Now let $f(m) = d * m^2$. We show that given π as above, at least half of the strings of length $f(m)$ are generated by π .

In order to see this, draw a string $\sigma \in \Sigma^{d*m^2}$ according to a fair $|\Sigma|$ -sided coin such that all symbols are equally likely. Divide σ into m equal parts of length $d * m$. The i -th part contains α_i with probability at least $1 - 2^{-m}$ as a substring, and thus the whole string is generated by π with probability at least $1 - m * 2^{-m}$. Note that $1 - m * 2^{-m} \geq 1/2$ for all m , and thus $f(m)$ meets the specification. ∎

We now present our algorithm for learning REGPAT_c . The algorithm has prior knowledge about the function r from Lemma 2 and the function f from Proposition 2. It takes as input c , δ and knowledge about the probability distribution by getting pol .

Learner (c, δ, pol)

(1) Read examples until an n is found such that the shortest example is strictly shorter than $c * n$ and the total number of examples (including repetitions) is at least

$$n * r\left(\frac{1}{2 * |\Sigma|^c * f(n) * pol(f(n))}, \frac{n}{\delta}, |\Sigma|^c\right) .$$

Let A be the set of all examples and A_j $(j \in \{1, 2, \ldots, n\})$, be the examples whose index is j modulo n; so the $(k * n + j)$-th example from A goes to A_j where k is an integer and $j \in \{1, 2, ..., n\}$.
Let $i = 1$, $\pi_0 = x_0$, $X_0 = \{\lambda\}$ and go to Step (2).

(2) For $\beta \in \Sigma^c$, let $Y_{i,\beta} = X_{i-1}\beta\Sigma^*$.
If $A \cap X_{i-1} \neq \emptyset$, then let $m = i - 1$ and go to Step (3).
Choose α_i as the $\beta \in \Sigma^c$, such that $|A_i \cap Y_{i,\beta}| > |A_i \cap Y_{i,\beta'}|$, for $\beta' \in \Sigma^c - \{\beta\}$ (if there is no such β, then abort the algorithm).
Let X_i be the set of all strings σ such that σ is in $\Sigma^*\alpha_1\Sigma^*\alpha_2\Sigma^* \ldots \Sigma^*\alpha_i$, but no proper prefix τ of σ is in $\Sigma^*\alpha_1\Sigma^*\alpha_2\Sigma^* \ldots \Sigma^*\alpha_i$.
Let $\pi_i = \pi_{i-1}\alpha_i x_i$, let $i = i + 1$ and go to Step (2).

(3) Output the pattern $\pi_m = x_0\alpha_1 x_1\alpha_2 x_2 \ldots \alpha_m x_m$ and halt.
End

Note that since the shortest example is strictly shorter than $c * n$ it holds that $n \geq 1$. Furthermore, if $\pi = x_0$, then the probability that a string drawn is λ is at least $1/pol(0)$. A lower bound for this is $1/(2 * |\Sigma|^c * f(n) * pol(f(n))$, whatever n is, due to the fact that pol is monotonically increasing. Thus λ appears with probability $1 - \delta/n$ in the set A_n and thus in the set A. So the algorithm is correct for the case that $\pi = x_0$.

It remains to consider the case where π is of the form $x_0\alpha_1 x_1\alpha_2 x_2 \ldots a_m x_m$ for some $m \geq 1$ where all α_i are in Σ^c.

Claim. Suppose any pattern $\pi = x_0\alpha_1 x_1\alpha_2 x_2 ... \alpha_m x_m \in \mathrm{reg}_c^m$. Furthermore, let $\pi_{i-1} = x_0\alpha_1 x_1 ... \alpha_{i-1} x_{i-1}$. Let the sets $Y_{i,\beta}, X_i$ be as defined in the algorithm and let $C(i, \beta, h)$ be the cardinality of $Y_{i,\beta} \cap \mathrm{Lang}(\pi) \cap \Sigma^h$.

Then, for all $h > 0$ and all $\beta \in \Sigma^c \setminus \{\alpha_i\}$, we have $C(i, \beta, h) \leq |\Sigma| * C(i, \alpha_i, h - 1) \leq C(i, \alpha_i, h)$.

Proof. Let $\sigma \in Y_{i,\beta} \cap \mathrm{Lang}(\pi)$. Note that σ has a unique prefix $\sigma_i \in X_i$. Furthermore, there exist $s \in \Sigma$, $\eta, \tau \in \Sigma^*$ such that

(i) $\sigma = \sigma_i\beta s\eta\tau$ and

(ii) $\beta s\eta$ is the shortest possible string such that $\beta s\eta \in \Sigma^*\alpha_i$.

The existence of s is due to the fact that $\beta \neq \alpha_i$ and β, α_i have both the length c. So the position of α_i in σ must be at least one symbol behind the one of β. If the difference is more than a symbol, η is used to take these additional symbols.

Now consider the mapping t from $\mathrm{Lang}(\pi) \cap Y_{i,\beta}$ to $\mathrm{Lang}(\pi) \cap Y_{i,\alpha_i}$ which replaces βs in the above representation of σ by α_i – thus $t(\sigma) = \sigma_i\alpha_i\eta\tau$. The mapping t is $|\Sigma|$-to-1 since it replaces the constant β by α_i and erases s (the information is lost about which element from Σ the value s is).

Clearly, σ_i but no proper prefix of σ_i is in X_i. So $\sigma_i\alpha_i$ is in $X_i\alpha_i$. The position of $\alpha_{i+1}, \ldots, \alpha_m$ in σ are in the part covered by τ, since $\sigma_i\beta s\eta$

is the shortest prefix of σ generated by $\pi_i\alpha_i$. Since π_i generates σ_i and $x_i\alpha_{i+1}x_{i+1}...\alpha_m x_m$ generates $\eta\tau$, it follows that π generates $t(\sigma)$. Hence, $t(\sigma) \in \text{Lang}(\pi)$. Furthermore, $t(\sigma) \in \Sigma^{h-1}$ since the mapping t omits one element. Also, clearly $t(\sigma) \in X_i\alpha_i\Sigma^* = Y_{i,\alpha_i}$. Thus, for $\beta \neq \alpha_i$, $\beta \in \Sigma^c$, it holds that $C(i,\beta,h) \leq |\Sigma| * C(i,\alpha_i,h-1)$. By combining with Proposition 1, $C(i,\alpha_i,h) \geq |\Sigma| * C(i,\alpha_i,h-1) \geq C(i,\beta,h)$. ∎

Claim. If $m > i$ then there is a length $h \leq f(m)$ such that

$$C(i,\alpha_i,h) \geq C(i,\beta,h) + \frac{|\Sigma|^h}{2 * |\Sigma|^c * f(m)}$$

for all $\beta \in \Sigma^c \setminus \{\alpha_i\}$. In particular,

$$prob(Y_{i,\beta} \cap \text{Lang}(\pi)) + \frac{1}{2 * |\Sigma|^c * pol(f(m)) * f(m)} \leq prob(Y_{i,\alpha_i} \cap \text{Lang}(\pi)).$$

Proof. Let $D(i,\beta,h) = \frac{C(i,\beta,h)}{|\Sigma|^h}$, for all h and $\beta \in \Sigma^c$. Proposition 1 and Claim 3 give that

$$D(i,\beta,h) \leq D(i,\alpha_i,h-1) \leq D(i,\alpha_i,h).$$

Since every string in $\text{Lang}(\pi)$ is in some set $Y_{i,\beta}$, it holds that $D(i,\alpha_i,f(m)) \geq \frac{1}{2*|\Sigma|^c}$. Furthermore, $D(i,\alpha_i,h) = 0$ for all $h < c$ since $m > 0$ and π does not generate the empty string. Thus there is an $h \in \{1,2,...,f(m)\}$ with

$$D(i,\alpha_i,h) - D(i,\alpha_i,h-1) \geq \frac{1}{2 * |\Sigma|^c * f(m)}\ .$$

For this h, it holds that

$$D(i,\alpha_i,h) \geq D(i,\beta,h) + \frac{1}{2 * |\Sigma|^c * f(m)}\ .$$

The second part of the claim follows, by noting that

$$prob(\Sigma^h) \geq \frac{1}{pol(h)} \geq \frac{1}{pol(f(m))}\ .$$

∎

We now show that the learner presented above indeed probabilistically exactly learns $\text{Lang}(\pi)$, for $\pi \in \text{reg}_c$.

A loop (Step (2)) invariant is that with probability at least $1 - \frac{\delta*(i-1)}{n}$, the pattern π_{i-1} is a prefix of the desired pattern π. This certainly holds before entering Step (2) for the first time.

Case 1. $i \in \{1,2,...,m\}$.

By assumption, $i \leq m$ and π_{i-1} is with probability $1 - \frac{\delta*(i-1)}{n}$ a prefix of π, that is, $\alpha_1, ..., \alpha_{i-1}$ are selected correctly.

Since α_i exists and every string generated by π is in $X_i\Sigma^*\alpha_i\Sigma^*$, no element of Lang($\pi$) and thus no element of A is in X_{i-1} and the algorithm does not stop too early.

If $\beta = \alpha_i$ and $\beta' \neq \alpha_i$, then

$$prob(Y_{i,\beta} \cap \mathrm{Lang}(\pi))$$

$$\geq\ prob(Y_{i,\beta'} \cap \mathrm{Lang}(\pi)) + \frac{1}{2 * |\Sigma|^c * f(m) * pol(f(m))}\ ,$$

by Claim 3. By Lemma 2, α_i is identified correctly with probability at least $1 - \delta/n$ from the data in A_i. It follows that the body of the loop in Step (2) is executed correctly with probability at least $1 - \delta/n$ and the loop-invariant is preserved.

Case 2. $i = m + 1$.

By Step (1) of the algorithm, the shortest example is strictly shorter than $c * n$ and at least $c * m$ by construction. Thus, we already know $m < n$.

With probability $1 - \frac{\delta*(n-1)}{n}$ the previous loops in Step (2) have gone through successfully and $\pi_m = \pi$. Consider the mapping t which omits from every string the last symbol. Now $\sigma \in X_m$ iff $\sigma \in \mathrm{Lang}(\pi)$ and $t(\sigma) \notin \mathrm{Lang}(\pi)$. Let $D(\pi, h)$ be the weighted number of strings generated by π of length h, that is, $D(\pi, h) = \frac{|\Sigma^h \cap \mathrm{Lang}(\pi)|}{|\Sigma|^h}$. Since $D(\pi, f(m)) \geq \frac{1}{2}$ and $D(\pi, 0) = 0$, there is a $h \in \{1, 2, \ldots, f(m)\}$ such that

$$D(\pi, h) - D(\pi, h-1) \geq \frac{1}{2 * f(m)} \geq \frac{1}{2 * |\Sigma|^c * f(n)}.$$

Note that $h \leq f(n)$ since f is increasing. It follows that

$$prob(X_m) \geq \frac{1}{2 * |\Sigma|^c * (f(n) * pol(f(n))}$$

and thus with probability at least $1 - \frac{\delta}{n}$ a string from X_m is in A_m, and in particular in A (by Lemma 2). Thus the algorithm terminates after going through the step (2) m times with the correct output with probability at least $1 - \delta$.

To get polynomial time bound for the learner, we note the following. It is easy to show that there is a polynomial $q(m, \frac{1}{\delta'})$ which with sufficiently high probability ($1 - \delta'$, for any fixed δ') bounds the parameter n of the learning algorithm. Thus, with probability at least $1 - \delta' - \delta$ the whole algorithm is successful in time and example-number polynomial in $m, 1/\delta, 1/\delta'$. Thus, for

any given δ'', by choosing $\delta' = \delta = \delta''/2$, one can get the desired polynomial time algorithm. ∎

We are hoping in the future (not as part of the present paper) to run our algorithm on molecular biology data to see if it can quickly provide useful answers.

References

1. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
2. S. Arikawa, T. Shinohara, and A. Yamamoto. Learning elementary formal systems. *Theoretical Computer Science*, 95:97–113, 1992.
3. T. Shinohara and H. Arimura. Inductive inference of unbounded unions of pattern languages from positive data. *Theoretical Computer Science*, 241:191–209, 2000.
4. I. Bratko and S. Muggleton. Applications of inductive logic programming. *Communications of the ACM*, 1995.
5. A. Brāzma, E. Ukkonen, and J. Vilo. Discovering unbounded unions of regular pattern languages from positive examples. In *Proceedings of the 7th International Symposium on Algorithms and Computation (ISAAC'96)*, volume 1178 of *Lecture Notes in Computer Science*, pages 95–104, Springer, 1996.
6. J. Case, S. Jain, S. Kaufmann, A. Sharma, and F. Stephan. Predictive learning models for concept drift. *Theoretical Computer Science*, 268:323–349, 2001. Special Issue for *ALT'98*.
7. J. Case, S. Jain, S. Lange, and T. Zeugmann. Incremental concept learning for bounded data mining. *Information and Computation*, 152(1):74–110, 1999.
8. T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger, and T. Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261(1):119-156, 2001.
9. E.M. Gold. Language identification in the limit. *Information & Control*, 10:447–474, 1967.
10. M. Kearns and L. Pitt. A polynomial-time algorithm for learning k-variable pattern languages from examples. In R. Rivest, D. Haussler and M. K. Warmuth (Eds.), *Proceedings of the Second Annual ACM Workshop on Computational Learning Theory*, pages 57–71, Morgan Kaufmann Publishers Inc., 1989.
11. P. Kilpeläinen, H. Mannila, and E. Ukkonen. MDL learning of unions of simple pattern languages from positive examples. In Paul Vitányi, editor, *Second European Conference on Computational Learning Theory*, volume 904 of *Lecture Notes in Artificial Intelligence*, pages 252–260. Springer, 1995.
12. S. Lange and R. Wiehagen. Polynomial time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.
13. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
14. S. Matsumoto and A. Shinohara. Learnability of subsequence languages. *Information Modeling and Knowledge Bases VIII*, pages 335–344, IOS Press, 1997.
15. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
16. S. Miyano, A. Shinohara and T. Shinohara. Polynomial-time learning of elementary formal systems. *New Generation Computing*, 18:217–242, 2000.

17. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:669–679, 1994.
18. R. Nix. Editing by examples. Technical Report 280, Department of Computer Science, Yale University, New Haven, CT, USA, 1983.
19. D. Reidenbach. A Negative Result on Inductive Inference of Extended Pattern Languages. In N. Cesa-Bianchi and M. Numao, editors, *Algorithmic Learning Theory, 13th International Conference, ALT 2002, Lübeck, Germany, November 2002, Proceedings*, pages 308–320. Springer, 2002.
20. R. Reischuk and T. Zeugmann. Learning one-variable pattern languages in linear average time. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 198–208. ACM Press, 1998.
21. P. Rossmanith and T. Zeugmann. Stochastic Finite Learning of the Pattern Languages. *Machine Learning* 44(1/2):67-91, 2001. Special Issue on Automata Induction, Grammar Inference, and Language Acquisition
22. A. Salomaa. Patterns (The Formal Language Theory Column). *EATCS Bulletin*, 54:46–62, 1994.
23. A. Salomaa. Return to patterns (The Formal Language Theory Column). *EATCS Bulletin*, 55:144–157, 1994.
24. R. Schapire, Pattern languages are not learnable. In M.A. Fulk and J. Case, editors, *Proceedings, 3rd Annual ACM Workshop on Computational Learning Theory*, pages 122–129, Morgan Kaufmann Publishers, Inc., 1990.
25. S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, and S. Arikawa. Knowledge acquisition from amino acid sequences by machine learning system BONSAI. *Trans. Information Processing Society of Japan*, 35:2009–2018, 1994.
26. T. Shinohara. Polynomial time inference of extended regular pattern languages. In *RIMS Symposia on Software Science and Engineering, Kyoto, Japan*, volume 147 of *Lecture Notes in Computer Science*, pages 115–127. Springer-Verlag, 1982.
27. T. Shinohara. Inferring unions of two pattern languages. *Bulletin of Informatics and Cybernetics*, 20:83–88., 1983.
28. T. Shinohara and S. Arikawa. Learning data entry systems: An application of inductive inference of pattern languages. Research Report 102, Research Institute of Fundamental Information Science, Kyushu University, 1983.
29. T. Shinohara and S. Arikawa. Pattern inference. In Klaus P. Jantke and Steffen Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *Lecture Notes in Artificial Intelligence*, pages 259–291. Springer, 1995.
30. R. Smullyan. *Theory of Formal Systems, Annals of Mathematical Studies, No. 47.* Princeton, NJ, 1961.
31. L.G. Valiant. A theory of the learnable. *Communications of the ACM* 27:1134–1142, 1984.
32. K. Wright. Identification of unions of languages drawn from an identifiable class. In R. Rivest, D. Haussler, and M.K. Warmuth, editors, *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 328–333. Morgan Kaufmann Publishers, Inc., 1989.
33. T. Zeugmann. Lange and Wiehagen's pattern language learning algorithm: An average-case analysis with respect to its total learning time. *Annals of Mathematics and Artificial Intelligence*, 23(1-2):117-145, 1998.