A Polynomial Time Learner for a Subclass of Regular Patterns

John Case*

Department of Computer and Information Sciences

University of Delaware

Newark, DE 19716-2586, USA

case@cis.udel.edu

Sanjay Jain†

School of Computing

National University of Singapore

Singapore 117543

sanjay@comp.nus.edu.sg

Rüdiger Reischuk

Institute for Theoretical Informatics

University at Lübeck

Ratzeburger Allee 160

23538 Lübeck, Germany

reischuk@tcs.uni-luebeck.de

Frank Stephan‡

National ICT Australia Ltd

Sydney Research Laboratory at Kensington

The University of New South Wales

Sydney NSW 2052, Australia

fstephan@cse.unsw.edu.au

Thomas Zeugmann

Institute for Theoretical Informatics

University at Lübeck

Ratzeburger Allee 160

23538 Lübeck, Germany

thomas@tcs.uni-luebeck.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 $x_0,\dots,x_m$ are variables and where $\alpha_1,...,\alpha_m$ are each strings of terminals of length c. This assumes that the algorithm randomly draws samples with natural and plausible assumptions on the distribution. With the aim of finding a better algorithm, we also explore computer simulations of a heuristic.

*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.

‡National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence Program.

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, $01xy211zx0$ — where the variables (for substitutions) are letters and the terminals are digits.

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, see, for example, [1, 5, 6, 7, 10, 12, 19, 21, 23, 24, 25, 28, 31] 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, [4, 11, 29, 32, 35].

Nix [18] as well as Shinohara and Arikawa [30, 31] outline interesting applications of pattern inference algorithms. For example, pattern language learning algorithms have been successfully applied toward some problems in molecular biology, see [27, 31]. Pattern languages and finite unions of pattern languages turn out to be subclasses of Smullyan's [33] 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) [3, 13, 15, 17]. 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. Angluin [1]). Gold [8] 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 members 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 based on computing descriptive patterns even of maximum length.

Giving up the idea to find descriptive patterns of maximum length but still computing descriptive patterns, Erlebach *et al.* [7] arrived at a one variable pattern language learner having update time $O(\ell^2 \log \ell)$, where ℓ is as above. Moreover, they also studied the *expected* total learning time of a variant of their learner and showed an $O(|\pi|^2 \log |\pi|)$ upper bound for it, where π is the target pattern. Subsequently, Reischuk and Zeugmann [21] designed a one-variable pattern language learner achieving the optimal expected total learning time of $O(|\pi|)$ for every target pattern π and for almost all meaningful distributions. Note that the latter algorithm does *not* compute descriptive patterns.

Another important special case extensively studied are the *regular pattern languages* introduced by Shinohara [28]. These are generated by the *regular patterns*, i.e., patterns in which each variable that appears, appears only once. The learners designed by Shinohara [28] 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 [34] PAC model. Shapire [26] could show that the whole class of pattern languages is not learnable within the PAC model unless $\mathcal{P}/poly = \mathcal{N}P/poly$ for any hypothesis space that allows a polynomially decidable membership problem. Since membership is $\mathcal{N}P$-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 – more precisely, the number of allowed unions is at most $poly(|\pi|, s, 1/\varepsilon, 1/\delta, |\Sigma|)$, 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 Σ is the alphabet of terminals over which the patterns are defined. 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 $\mathcal{N}P$-complete. Thus, the class of all extended regular pattern languages is not polynomial-time PAC learnable unless $\mathcal{R}P = \mathcal{N}P$ 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 terminal strings is 1, see below for a formal 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 $x_0\alpha_1x_1\alpha_2x_2 \ldots \alpha_m x_m$, 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 (cf. [21, 22, 23]).

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 arbitrary 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 conceptually 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 E/δ 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*[1], cf. Definition 7 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, the maximal length of a terminal string in the pattern and a term describing knowledge about the probability distribution as such a parameter.

We shall provide a learner which succeeds with high probability in polynomial time on every text which is drawn to any admissible probability distribution *prob*. An admissible distribution *prob* has to satisfy, besides some normality conditions, also

[1] This model has also been called *stochastic finite learning* (cf. [21, 22, 23]).

the condition

$$prob(\sigma) \geq \frac{|\Sigma|^{-|\sigma|}}{pol(|\sigma|)} \text{ for } \sigma \in L \text{ where } \sum_{n=0}^{\infty} \frac{1}{pol(n)} \leq 1 \; .$$

Here, pol is a $\{2, 3, \ldots\}$-valued increasing polynomial and the pause-symbol $\#$ has the probability $1 - \sum_{\sigma \in L} prob(\sigma)$. This condition guarantees that long examples are still sufficiently frequent. The more precise requirements for $prob$ and its texts are given in Definition 5.

Furthermore, probabilistically exact learnability from text for non-empty languages implies probabilistically exact learnability from pause-free texts and informants where pause-free texts are those generated by probability distributions satisfying

$$\sum_{\sigma \in L} prob(\sigma) = 1.$$

Our algorithm is presented in detail in Section 3 below and runs with all three models of data presentation in a uniform way. 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 $x_0 \alpha_1 x_1 ... \alpha_m x_m$ for unknown m but known c, from number of examples polynomial in m (and exponential in c), where $\alpha_1, ..., \alpha_m \in \Sigma^c$. Here Σ^c denotes the set of all strings over Σ of length c.

Note that having our patterns defined as starting and ending with variables is not crucial. One can just handle patterns starting or ending with terminals easily by 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 terminal 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.

While the main goal of the paper is to establish some polynomial bound for the learning algorithm, Section 4 is dedicated to giving more explicit bounds for the basic case that $\Sigma = \{0, 1\}$ and $c = 1$. The probability distribution satisfies

$$prob(\sigma) = \frac{|\Sigma|^{-|\sigma|}}{(|\sigma| + 1)(|\sigma| + 2)}$$

for $\sigma \in L$ and $prob(\#)$ is chosen accordingly. Although the bounds for this basic case are much better than in the general case, it seems that the current implementation of the algorithm is even more efficient than the improved theoretical bounds suggest. Experiments have also been run for alphabets of size $3, 4, 5$ and the pattern languages REGPAT_2 and REGPAT_3.

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. Furthermore, for all real numbers s, t with $s < t$ we use (s, t) to denote the open intervall generated by s and t, i.e., $(s, t) = \{r \mid r \text{ is real and } s < r < t\}$.

Let Σ be any finite alphabet such that $|\Sigma| \geq 2$ and let V be a countably infinite set of variables such that $\Sigma \cap V = \emptyset$. Following a tradition in formal language theory, the elements of Σ are called *terminals*. By Σ^* we denote the free monoid over Σ, and we set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$, where λ is the empty string. As above, Σ^c denotes the set of strings over Σ with length c. We let $a, b, \dots$ range over terminal symbols from Σ and $\alpha, \sigma, \tau, \eta$ over terminal strings from Σ^*. $x, y, z, x_1, x_2, \dots$ 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 consider 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 studied 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, Reidenbach [19] proved that even the terminal-free erasing pattern languages are *not* learnable from positive data if $|\Sigma| = 2$. On the other hand, the terminal-free erasing pattern languages are learnable from positive data if $|\Sigma| \geq 3$ (cf. Reidenbach [20]).

Definition 3 [Shinohara[28]]. *A pattern* π *is said to be* ***regular*** *if it is of the form* $x_0\alpha_1x_1\alpha_2x_2\dots\alpha_mx_m$, *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) $\mathrm{reg}_c^m = \{\pi \mid \pi = x_0\alpha_1x_1\alpha_2x_2\dots\alpha_mx_m, \text{ where each } \alpha_i \in \Sigma^c\}$.

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

(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 [34] 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. *Let* $\#$, $\# \notin \Sigma$, *denote a pause-symbol and* $D \subseteq \Sigma^* \cup \{\#\}$. *Given a polynomial* pol, *a probability distribution* $prob$ *on* D *is called* ***pol*** ***-regular*** *if*

(a) $prob(\sigma) * |\Sigma|^{|\sigma|} \geq 1/pol(|\sigma|)$ *for all* $\sigma \in D \setminus \{\#\}$ *and*

(b) $prob(\sigma) * |\Sigma|^{|\sigma|} \leq prob(\tau) * |\Sigma|^{|\tau|}$ *for all* $\sigma, \tau \in D \setminus \{\#\}$ *with* $|\sigma| \geq |\tau|$.

Note that a distribution *prob* on D generates only elements from D. The second item in the definition of the *pol*-regular probability distribution enforces that in D strings of the same length have the same probability.

Next, we define the different sources of information for the learners considered in this paper.

Definition 6. *Let $L \subseteq \Sigma^*$ be a language.*

(a) *A **probabilistic text for L with parameter** pol is an infinite sequence drawn with respect to any pol-regular distribution prob on the domain $D = L \cup \{\#\}$.*

(b) *If $L \neq \emptyset$ then a **probabilistic pause-free text for L with parameter pol** is a probabilistic text for L with parameter pol with respect to any pol-regular distribution prob on the domain $D = L$.*

(c) *A **probabilistic informant for L with parameter pol** is an infinite sequence of pairs $(\sigma, L(\sigma))$ where σ is drawn according to a pol-regular distribution prob on $D = \Sigma^*$.*

In the following, we shall frequently omit the words "probabilistic" and "with parameter *pol*" when referring to these types of text, pause-free text and informant, since it is clear from the context what is meant.

Definition 7. *A learner M is said to **probabilistically exactly learn** a class $\mathcal{L}$ of pattern languages if for every increasing $\{2,3,\ldots\}$-valued polynomial pol with*

$$\sum_{n=0}^{\infty} \frac{1}{pol(n)} \leq 1$$

there is a polynomial q such that for all δ and every language $L \in \mathcal{L}$, with probability at least $1-\delta$, M halts and outputs a pattern generating L after reading at most $q(|\pi|, \frac{1}{\delta})$ examples from a probabilistic text for L with parameter pol. That is, for all δ and every pattern π generating a language $L \in \mathcal{L}$ and for every pol-regular distribution prob on $L \cup \{\#\}$, with probability $1-\delta$, M draws at most $q(|\pi|, \frac{1}{\delta})$ examples according to prob and then outputs a pattern π such that $L = \mathrm{Lang}(\pi)$.

It should be noted that learning from pause-free text can be much easier than learning from text. For example, the class of all singletons $\{\sigma\}$ with $\sigma \in \Sigma^*$ is learnable from pause-free text: the learner conjectures $\{\sigma\}$ for the first example σ in the text and is correct. But it is not learnable from informant since for each length n, the strings $\sigma \in \Sigma^n$ satisfy that $prob(\sigma) \leq |\Sigma|^{-n}$ and the learner sees with high probability exponentially many negative examples of low information content before $(\sigma, 1)$ comes up. Furthermore, every informant can be translated into a text as follows: one replaces $(\sigma, 1)$ by σ and $(\sigma, 0)$ by $\#$. Thus the class of all singletons is not probabilistically exactly learnable from text. So permitting pauses satisfies two goals: (a) there is a text for the empty set; (b) it is enforced that learnability from text implies learnability from informant. The latter also holds in standard inductive inference.

Lange and Wiehagen [12, 36] presented an algorithm which learns all non-erasing pattern languages by just analyzing all strings of shortest length generated by the target pattern. From these strings, the pattern can be reconstructed. Similarly, in the case of erasing pattern languages in REGPAT_c, the shortest string is just the concatenation of all terminals. Knowing c, one can reconstruct the whole pattern from this string. But this algorithm does not succeed for learning REGPAT_c, even if one is learning from pause-free texts. The distributions $prob_L$ given as

$$prob_L(\sigma) = \frac{|\Sigma|^{-|\sigma|}/pol(|\sigma|)}{1 - \sum_{\tau\in L} |\Sigma|^{-|\tau|}/pol(|\tau|)} \text{ for all } \sigma \in L$$

witness this fact where L ranges over REGPAT_c. Let $m * c$ be the number of terminals in the pattern generating L. By Proposition 1 there is a polynomial f such that at least half of the words of length $f(m)$ are in L. So the denominator is at least $0.5/pol(f(m))$. On the other hand, the numerator for the shortest word in L is exactly $|\Sigma|^{-m}/pol(m)$. So the probability of this word is at most $|\Sigma|^{-m} * pol(f(m))/pol(m)$ and goes down exponentially in m.

3 Main Result

In this section we show that REGPAT_c is probabilistically exactly learnable. First we need some well-known facts which hold for arbitrary distributions, later we only consider pol-regular distributions. The following lemma is based on Chernoff Bounds (cf., e.g., [9]). Here, we use e to denote the base of the natural logarithm.

Lemma 1. *Let $X, Y \subseteq \Sigma^*$, let $\delta, \varepsilon \in (0, 1/2)$, and let $prob(X) \geq prob(Y) + \varepsilon$. If one draws at least*

$$\frac{2}{\varepsilon^2} * \frac{-\log\delta}{\log e}$$

many examples from Σ^* according to the probability distribution $prob$, then with probability at least $1 - \delta$, elements of X show up more frequently than elements of Y.

Note that the number $\frac{2}{\varepsilon^2 * \delta}$ is an upper bound for $\frac{2}{\varepsilon^2} * \frac{-\log\delta}{\log e}$. More generally, the following holds.

Lemma 2. *One can define a function $r: (0, 1/2) \times (0, 1/2) \times \mathbb{N} \longrightarrow \mathbb{N}$ such that $r(\varepsilon, \delta, k)$ is polynomial in $k, \frac{1}{\varepsilon}, \frac{1}{\delta}$ and for all sets $X, Z, Y_1, Y_2, \ldots, Y_k \subseteq \Sigma^*$, the following holds.*

If $prob(X) - prob(Y_i) \geq \varepsilon$, for $i = 1, 2, \ldots, k$, $prob(Z) \geq \varepsilon$ and one draws at least $r(\varepsilon, \delta, k)$ 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 $Y_1, ..., Y_k$.*

Since any regular pattern π has a variable at the end, the following lemma holds.

Lemma 3. *For every regular pattern π and all $m \in \mathbb{N}$, $|\mathrm{Lang}(\pi) \cap \Sigma^{m+1}| \geq |\Sigma| * |\mathrm{Lang}(\pi) \cap \Sigma^m|$.*

Proposition 1. *For every fixed constant $c \in \mathbb{N}^+$ and every finite alphabet Σ, $|\Sigma| \geq 2$, there is a polynomial f such that for every $\pi \in \mathrm{reg}_c^m$, 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. 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. ∎

Next we present our algorithm ***LRP***[2] for learning REGPAT_c. The algorithm has prior knowledge about the function r from Lemma 2 and the function f from Proposition 1. It takes as input c, δ and knowledge about the probability distribution by getting pol.

Algorithm *LRP*: The learner has parameters (Σ, c, δ, pol) and works as follows. The variables $A, A_0, A_1, \ldots$ range over multisets.

(1) Read examples until an n is found such that the shortest non-pause example is strictly shorter than $c * n$ and the total number of examples (including repetitions and pause-symbols) 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 multiset of all positive examples (including pause-symbols) and A_j ($j \in \{1, 2, \ldots, n\}$) be the multiset of examples in A 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 some string in X_{i-1} is also in the multiset A, then let $m = i - 1$ and go to Step (3).

Choose α_i as the lexicographically first $\beta \in \Sigma^c$, such that the strings from $Y_{i,\beta}$ occur in A_i at least as often as the strings from $Y_{i,\beta'}$ for any $\beta' \in \Sigma^c \setminus \{\beta\}$.

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 $\pi_i = \pi_{i-1}\alpha_ix_i$, let $i = i + 1$ and go to Step (2).

[2] LRP stands for Learner for Regular Patterns.

(3) Output the pattern $\pi_m = x_0\alpha_1x_1\alpha_2x_2\ldots\alpha_mx_m$ 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 $x_0\alpha_1x_1\alpha_2x_2\ldots a_mx_m$ for some $m \geq 1$ where all α_i are in Σ^c.

Claim 1 *Suppose any pattern* $\pi = x_0\alpha_1x_1\alpha_2x_2...\alpha_mx_m \in \mathrm{reg}_c^m$. *Furthermore, let* $\pi_{i-1} = x_0\alpha_1x_1...\alpha_{i-1}x_{i-1}$. *Let the sets* $Y_{i,\beta}, X_i$ *be defined as in the algorithm and let* $C(i,\beta,h)$ *be the cardinality of* $Y_{i,\beta} \cap \mathrm{Lang}(\pi) \cap \Sigma^h$. *Then we have* $C(i,\beta,h) \leq |\Sigma| * C(i,\alpha_i,h-1) \leq C(i,\alpha_i,h)$ *for all* $h > 0$ *and all* $\beta \in \Sigma^c \setminus \{\alpha_i\}$.

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 $|\beta| = |\alpha_i| = c$. 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 $t\colon \mathrm{Lang}(\pi) \cap Y_{i,\beta} \longrightarrow \mathrm{Lang}(\pi) \cap Y_{i,\alpha_i}$ replacing βs in the above representation (I) 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 terminal string 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 $x_i\alpha_{i+1}x_{i+1}...\alpha_mx_m$ generates $\eta\tau$, it follows that π generates $t(\sigma)$. Hence, $t(\sigma) \in \mathrm{Lang}(\pi)$. Furthermore, $t(\sigma) \in \Sigma^{h-1}$ since the mapping t omits one element. Also, clearly $t(\sigma) \in X_i\alpha_i\Sigma^* = Y_{i,\alpha_i}$. Therefore, $C(i,\beta,h) \leq |\Sigma| * C(i,\alpha_i,h-1)$ for $\beta \in \Sigma^c \setminus \{\alpha_i\}$. By combining with Lemma 3, $C(i,\alpha_i,h) \geq |\Sigma| * C(i,\alpha_i,h-1) \geq C(i,\beta,h)$. ∎

Claim 2 *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, for every* pol *-regular distribution with domain* $\mathrm{Lang}(\pi)$ *or domain* $\mathrm{Lang}(\pi) \cup \{\#\}$,

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

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

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

Since every string in $\mathrm{Lang}(\pi)$ is in some set $Y_{i,\beta}$, we conclude 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 taking into account that $h \leq f(m)$ implies $pol(h) \leq pol(f(m))$, since pol is monotonically increasing. Thus,

$$prob(\sigma) \geq \frac{|\Sigma|^{-h}}{pol(h)} \geq \frac{|\Sigma|^{-h}}{pol(f(m))}$$

for all $\sigma \in \Sigma^h \cap \mathrm{Lang}(\pi)$. ∎

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

Theorem 1. *Let $c \in \mathbb{N}^+$ and let Σ be any finite alphabet with $|\Sigma| \geq 2$. Algorithm **LRP** probabilistically exactly learns the class of all $\mathrm{Lang}(\pi)$ with $\pi \in \mathrm{reg}_c$ from text, from pause-free text and from informant.*

Proof. Since an informant can be translated into a text, the result is only shown for learning from text. The proof also covers the case of learning from pause-free text, it is almost identical for both versions. Let $prob$ be a pol-regular distribution on $\mathrm{Lang}(\pi)$ or $\mathrm{Lang}(\pi) \cup \{\#\}$.

A loop invariant (in Step (2)) 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 $\mathrm{Lang}(\pi)$ 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

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

by Claim 2. 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 an $h \in \{1,2,\ldots,f(m)\}$ 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). Therefore, the algorithm terminates after going through Step (2) m times with the correct output with probability at least $1-\delta$.

To get a polynomial time bound for the learner, 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 algorithm ***LRP***. Thus, with probability at least $1-\delta'-\delta$ algorithm ***LRP*** is successful in time and example-number polynomial in $m, 1/\delta, 1/\delta'$. Hence, for any given δ'', by choosing $\delta' = \delta = \delta''/2$, one can get the desired polynomial time algorithm. ∎

If one permits the pattern to start or end with some terminal parts and these parts are not too long, then one can learn also this derived class by reading polynomially more data-items and skipping off the common prefixes and suffixes of all data. Consequently, we have the following result.

Theorem 2. *Let $c \in \mathbb{N}^+$ and let Σ be any finite alphabet with $|\Sigma| \geq 2$. Then, the class $\{\alpha \mathrm{Lang}(\pi)\beta \mid \alpha,\beta \in \Sigma^*,\ |\alpha| \leq d,\ |\beta| \leq d,\ \pi \in \mathrm{reg}_c\}$ is probabilistically exactly learnable from text where the polynomial bound for the number of examples has the parameters pol and c, d.*

4 Experimental Results

Looking at the results proved so far, we see that the theoretical bounds are very large. Therefore, we are interested in finding possible improvements.

First, we fix the polynomial *pol*. Linear functions cannot be used, since the sum of their reciprocals diverges, thus *pol* is taken to be quadratic. More precisely, *pol* is taken such that, for all $n \in \mathbb{N}$,

$$pol(n) = (n+1)*(n+2) \text{ and } \frac{1}{n+1} = \sum_{m=n}^{\infty} \frac{1}{pol(m)} .$$

Second, we study the special case that $\Sigma = \{0,1\}$ and REGPAT_1 is being learned. For this particular setting, we improve the theoretical bounds, see Theorem 3 below.

Then, we provide some evidence, that even these bounds are not optimal, since experiments run for small alphabets Σ and small values of m, c give much better results. These experiments use Algorithm ***HLRP***[3] which does not compute the number of examples in advance but monitors its intermediate results and halts if a hypothesis looks sufficiently reasonable with respect to the data seen so far. More details will be given below.

Theorem 3. *For* $\Sigma = \{0,1\}$ *and considering only* $(n+1)(n+2)$ *-regular distributions, Algorithm* ***LRP*** *can be adapted such that it probabilistically exactly learns* REGPAT_1 *and needs for any* $L \in \mathrm{REGPAT}_1^m$ *and confidence* $1-\delta$ *at most*

$$8*(4m-1)^2*(4m+1)^2*(4m+2)^2*(2m+2)*\frac{\log(2m+2)-\log(\delta)}{\log(e)}$$

positive examples including pauses.

Proof. We can improve the bounds on the number of examples needed by using Lemma 1 instead of Lemma 2. We need enough examples to guarantee with probability $1-(\delta/(m+2))$:

(1) A data-item of length $2m$ or less is drawn in Step (1);

(2) For $i = 1, 2, \ldots, m$, the right $\alpha_i \in \{0,1\}^*$ is selected in Step (2);

(3) For $i = m+1$ a data-item in $A \cap X_{i-1}$ is found.

In the first case, one has to draw an example of length $2m$ which is generated by the pattern. Since at least half of the strings of this length are generated by the pattern, the probability that a randomly drawn datum has this property is at least $\frac{1}{2(2m+1)(2m+2)}$.

For the second case, one counts the number of strings of length $2m$ which are in one of the following three regular expressions.

- $(1-a_0)^*a_0 \ldots (1-a_{i-1})^*a_{i-1}0\{0,1\}^*a_i(1-a_i)^*a_{i+1}(1-a_{i+1})^* \ldots a_m(1-a_m)^*$;
- $(1-a_0)^*a_0 \ldots (1-a_{i-1})^*a_{i-1}1\{0,1\}^*a_i(1-a_i)^*a_{i+1}(1-a_{i+1})^* \ldots a_m(1-a_m)^*$;
- $(1-a_0)^*a_0 \ldots (1-a_{i-1})^*a_{i-1}a_i(1-a_i)^*a_{i+1}(1-a_{i+1})^* \ldots a_m(1-a_m)^*$.

[3]HLRP stands for Heuristic Learner for Regular Patterns.

The number of strings of length $2m$ generated by the first and generated by the second expression is the same. But the one generated by the third expression is exactly as large as the number of strings of length $2m-1$ which consist of m zeros and $m-1$ ones. This number has the lower bound $\frac{1}{2m-1}2^{2m-1}$. This permits to conclude that the value a_i is favored over the value $1-a_i$ at least with probability

$$\varepsilon = \frac{1}{2(2m-1)(2m+1)(2m+2)} \, .$$

The last event for leaving the loop has the same probability ε, one just obtains it by $i = m+1$ in the above regular expressions where of course then the part $a_i(1-a_i)^* \ldots a_m(1-a_m)^*$ has to be omitted from the expression. Furthermore, ε is a lower bound for the first probability.

Using Lemma 1, it suffices to draw $(m+2)$ times $2 * \varepsilon^{-2} * \frac{\log(m+2)-\log\delta}{\log e}$ examples.

Note that the algorithm uses the parameter n as a bound for m, i.e., $n = 2m$.

So with probability at least $1-\delta$, the algorithm uses at most the following quantity of data:

$$8 * (2n-1)^2 * (2n+1)^2 * (2n+2)^2 * (n+2) * \frac{\log(n+2) - \log\delta}{\log e} \, . \quad \blacksquare$$

The entry "upper bound" in Figure 1 has the specific values obtained for $m = 2, 3, \ldots, 20$ and $\delta = 1/3$.

We have tried to find further improvements for these bounds. These improvements have no longer been verified theoretically, but only be looked up experimentally. For that, we applied the following experimental setting.

Experimental Setting 1 *The heuristic used is Algorithm **HLRP** which ignores pause-symbols. This algorithm reads data and tries in parallel to find the pattern. It stops reading data when the process has given a pattern with sufficiently high estimated confidence.*

Learning L, the experiments were run for the following two distributions which are the extreme $(n+1)(n+2)$-regular distributions with respect to having as many pauses as possible and no pauses at all where $\sigma \in L$:

$$\begin{aligned} prob_{L,\#}(\sigma) &= \frac{|\Sigma|^{-|\sigma|}}{(|\sigma|+1) * (|\sigma+2|)}; \\ prob_{L,\#}(\#) &= 1 - \sum_{\tau \in L} prob_{L,\#}(\tau); \\ prob_L(\sigma) &= \frac{prob_{L,\#}(\sigma)}{1 - prob_{L,\#}(\#)}. \end{aligned}$$

The probabilities can be obtained from $prob_{\Sigma^*,\#}$ as follows: In the case of $prob_{L,\#}$, one draws an example string σ and then provides σ to the learner in the case that $\sigma \in L$ and provides $\#$ to the learner otherwise. In the case of $prob_L$, one draws examples according to $prob_{\Sigma^*,\#}$ until an example in L is found which one then provides to the learner.

*Since Algorithm **HLRP** ignores pause-symbols, the experiments were run with examples drawn according to $prob_{L,\#}$. While the algorithm runs, one can count the number a of non-pause examples drawn and the number b of pauses drawn. The example complexities with respect to $prob_L$ and $prob_{L,\#}$ are a and $a+b$, respectively. So the experiments cover both cases at once where only the way to count the number of drawn examples are different for $prob_L$ and $prob_{L,\#}$.*

The used heuristic is Algorithm ***HLRP***. The two main modifications to the theoretical model are the following: The algorithm alternately draws a data item and tries to learn from the data seen so far until it thinks that the result of the learning-trial gives a sufficiently reasonable hypothesis. Furthermore, only sufficiently short data-items are considered.

Algorithm *HLRP* (for learning regular patterns):

`Repeat`

Initialize *bound* by a default constant, g a function determined below and let A be the empty multiset.

`Repeat`

Draw example σ;

Put σ into A if

- σ is not the pause symbol;
- $|\sigma| \leq bound$;
- $|\sigma| \leq 1.4 * |\tau|$ for all τ already in A;

`Until Either` there are m and $\rho = x_0 a_1 x_1 a_2 x_2 \dots a_m x_m$ such that

- ρ generates all data in A;
- for all $i \leq m$, the number of data-items in A generated by the regular expression

  $$(\Sigma \setminus \{a_0\})^* a_0 \dots (\Sigma \setminus \{a_{i-1}\})^* a_{i-1} a_i \Sigma^*$$

  minus the number of the data-items in A generated by

  $$(\Sigma \setminus \{a_0\})^* a_0 \dots (\Sigma \setminus \{a_{i-1}\})^* a_{i-1} b \Sigma^*$$

  is at least $g(m)$ for any $b \in \Sigma \setminus \{a_i\}$;
- the number of data-items in A generated by the regular expression

  $$(\Sigma \setminus \{a_0\})^* a_0 \dots (\Sigma \setminus \{a_m\})^* a_m$$

  is at least $g(m)$.

`Or` default bounds on memory usage and number of examples are exceeded.

`Until` The previous loop ended in the "either-case" or has been run for ten times.

`If` no such pattern ρ has been found `Then Halt` with error message
`Else` consider this unique ρ.

`If` $|\Sigma| \geq 3$ `Then For` $k = 1$ to $m-1$ Remove the variable x_k from the pattern whenever it can be removed without becoming inconsistent with the data stored in A. (ITF)

`Output` the resulting pattern ρ and `Halt`.
`End`

Note that in the second condition on ρ, the regular expressions for $i = 0$ are $a_0\Sigma^*$ and $b\Sigma^*$; for $i = 1$ the corresponding expressions are $(\Sigma \setminus \{a_0\})^* a_0 a_1 \Sigma^*$ and $(\Sigma \setminus \{a_0\})^* a_0 b \Sigma^*$.

The `For-Loop` labeled (ITF) removes variables not needed to generate the data. It is included for the later considered case where the above algorithm is adapted to learn regular patterns outside REGPAT_1. Surprisingly, for all experiments done with the alphabet $\{0,1,2\}$, there were no errors caused by removing superfluous variables in an otherwise correct pattern. Thus this routine was actually always activated when $|\Sigma| \geq 3$, even when learning languages from REGPAT_1 where this routine is superfluous. The routine is not used for $\Sigma = \{0,1\}$ since there the patterns are not unique: $\mathrm{Lang}(x_0 0 x_1 1 x_2) = \mathrm{Lang}(x_0 01 x_2)$.

Algorithm ***HLRP*** has been implemented such that at every learning trial it is permitted to store up to 10^6 digits belonging to 50000 examples and draw up to 10^7 examples. The algorithm tries up to 10 times to learn a pattern without violating the resource-bounds and then gives up. The parameter *bound* is initialized to 2000 at each learning trial which is 0.2 % of the available memory to store the data-items in A and $g(m)$ being the constant 10. The existence of the pattern ρ can be checked effectively by constructing it inductively as in Algorithm ***LRP***.

m	correct	without pause	with pause	% used	upper bound
10	1000	4064.909	81633.997	4.42	$3 * 10^{12}$
20	999	4184.943	167777.073	5.42	$447 * 10^{12}$
30	993	4909.675	294712.912	7.37	$7999 * 10^{12}$
40	970	6439.009	515298.445	9.05	$62110 * 10^{12}$
50	967	8895.355	889807.785	9.79	$304901 * 10^{12}$
60	946	11980.308	1438119.953	10.53	$1119325 * 10^{12}$
70	933	15777.699	2209051.668	11.44	$3361797 * 10^{12}$
80	927	20792.314	3325566.018	12.03	$8716312 * 10^{12}$
90	917	27871.036	5017601.075	12.63	$20197744 * 10^{12}$
100	897	48742.807	9749833.913	13.24	$42832613 * 10^{12}$

Figure 1: Learning REGPAT_1^m where $\Sigma = \{0,1\}$.

Figure 1 gives a table on the outcome of running Algorithm ***HLRP*** in order to learn for each $m \in \{10,20,30,40,50,60,70,80,90,100\}$ exactly 1000 patterns having m terminals. These patterns have been chosen according to the uniform distribution over REGPAT_1^m. The algorithm itself does not know m. The first column gives the number m of terminals of the pattern $x_0a_1x_1a_2x_2\ldots a_mx_m$ to be learned. For each m, 1000 patterns were randomly drawn with the parameters $a_1, a_2, \ldots, a_m$ being chosen according to the uniform distribution. The second column gives how many of these pattern have been learned correctly and the third and fourth column state the average number of examples needed. The third column counts the examples drawn excluding the pause-symbols and the fourth column counts the examples drawn including the pause-symbols. That is, the third column refers to the distribution $prob_L$ and the fourth column refers to the distribution $prob_{L,\#}$ discussed in the Experimental Setting 1. The fifth column states the percentage of the examples from the third column which were actually used by the algorithm and not skipped. The other examples were too long and not processed by the learner. The upper bound is the theoretically sufficient number of examples with respect to confidence $1-\delta = 2/3$. Only trials for patterns in $\mathrm{REGPAT}_1^{80}, \mathrm{REGPAT}_1^{90}, \mathrm{REGPAT}_1^{100}$ were aborted and rerun, but each of these patterns was learned within the first 10 trials. So the incorrect learning processes are due to incorrect hypotheses and not due to violating memory or example bounds in each of the 10 trials.

Algorithm ***HLRP*** does not use all data different from the pause-symbol but only the sufficiently short ones. This can be justified as follows: The underlying probability distribution produces with probability $\frac{1}{n+1}$ a string which has at least the length n and the longer the string, the more likely it is in the language to be learned. Thus if the length is very high, the data reveals almost nothing about the language and might confuse the learner more by being random. Skipping long examples has been justified experimentally: Algorithm ***HLRP*** was correct on all 1000 random patterns when learning REGPAT_1^{10} for the alphabet $\{0,1\}$. The algorithm was run for the same class with the factor in the third condition for the selection of data, namely that $|\sigma| \leq 1.4 * |\tau|$ for all previously seen data τ, being relaxed to 1.6, 1.8 and 2.0. The number of correct patterns was 998, 998 and 994 respectively. Larger factors give even worse results. Relaxing the third condition and permitting more long strings reduces the confidence of the algorithm.

The parameter *bound* itself should not be there if sufficient resources are available since it makes it definitely impossible to learn patterns having more than *bound* terminals. But if one has a memory limit of length ℓ, then, in average, every ℓ-th learning experiment has a first example of length ℓ or more. Therefore one cannot avoid having such a bound in an environment with a memory limitation. The choice $bound = \ell/500$ looks a bit arbitrary, but this choice is not severe for the learning problems investigated. For learning much longer patterns, one would of course have to revise this decision.

Choosing the parameter $g(m) = 10$ is debatable. If one wants to learn large patterns with sufficient high confidence, then g has to be a growing function. Otherwise, as indicated by the experimental data, the confidence is going down for large m. More precisely, the larger the parameter m, the less likely the learner succeeds to learn any given pattern from REGPAT_c^m. Repairing this by having larger values for $g(m)$ has the obvious disadvantage that Algorithm ***HLRP*** draws more examples in the learning process. That is, learning is slowed down by increasing $g(m)$.

Figure 2 shows the result for learning patterns having 50 terminals in dependence of the alphabet size. The larger the alphabet, the more restrictive is the length bound

$\|\Sigma\|$	correct	repetitions	without pause	with pause	% used
2	967	0	8895.355	889807.785	9.79
3	980	0	22604.453	3366611.336	4.60
4	993	8	44111.067	8735098.330	2.78
5	995	82	80512.278	19891099.233	1.90

Figure 2: Learning REGPAT_1^{50} in dependence of the alphabet size.

$\|\Sigma\|$	c	$m*c$	correct	rptts.	without pause	with pause	% used
2	1	12	1000	0	5130.285	123529.904	3.58
2	2	12	920	9	23256.913	738004.683	1.08
2	3	12	890	17	37152.843	1542162.670	0.71
3	1	12	1000	15	98571.932	3460717.043	0.36
3	2	12	957	159	268867.961	14615063.504	0.11
3	3	12	922	1034	611210.432	58396686.666	0.06
4	1	12	1000	294	436565.410	20142491.835	0.12
4	2	12	711	3677	1905036.705	171744393.587	0.04
4	3	12	23	8922	1924946.265	396696831.533	0.03

Figure 3: Learning REGPAT_1^{12}, REGPAT_2^{6} and REGPAT_3^{4}.

on the choice of data considered. So, on one hand the number of incorrect hypotheses went down from 33 to 5. On the other hand, the amount of data needed goes up. Furthermore, for alphabets of size 4 and 5, some repetitions on the inner repeat loop take place and the learning algorithm is rerun on the same pattern. For the alphabet $\{0,1,2,3,4\}$, it was four times needed to repeat the learning process twice.

An attempt was made to adapt the algorithm for $c = 2,3$ and to run it for larger alphabet sizes. Since the provenly correct algorithm needs a large quantity of data exceeding any practical bound, the heuristic approach already implemented in Algorithm ***HLRP*** was chosen: One uses the learner for REGPAT_1 in order to figure out the terminals in the pattern. For the alphabet $\{0,1\}$, one then interprets this sequence of terminals as being the one for REGPAT_c where $c = 2,3$. For the alphabets $\{0,1,2\}$ and $\{0,1,2,3\}$, the last three lines of Algorithm ***HLRP*** were run and taken into account at the check for correctness.

Figure 3 gives a summary of these results obtained from the following experiments: For each of the alphabets $\{0,1\}, \{0,1,2\}, \{0,1,2,3\}$ and each $c \in \{1,2,3\}$, 1000 patterns containing 12 terminals were tried to learn where the terminals $a_1, a_2, \ldots, a_{12}$ were chosen independently from the alphabet under the uniform distribution. Since the distributions of the correct data for patterns from REGPAT_1 and REGPAT_c with $c > 1$ are not the same, the performance of the algorithm goes down and the patterns to be learned in the data for Figure 3 had always 12 terminal symbols. It turned out that the data for the experiments in Figure 3 are much more sensitive to the alphabet size than those for the experiments in Figure 2. Algorithm ***HLRP*** had its best performance for the alphabet $\{0,1,2\}$. For the alphabet $\{0,1\}$, the main problem was errors due to figuring out the sequence of terminals incorrectly.

For the alphabet $\{0,1,2,3\}$, the main problem are too excessive usage of memory and time, so that many trials were aborted or rerun. For only 23 out of the 1000 languages from REGPAT_3^4, Algorithm ***HLRP*** terminated in any of the ten trials. But in that case, it always gave the correct pattern. Note that for REGPAT_3^4, the average number of examples drawn heavily depends on the choice of parameters: every trial reads at most $4 * 10^7$ examples (with pauses) and the overall number of examples drawn is at most $4 * 10^8$ since there are at most 10 trials per pattern. The measured average complexity is $3.96696831533 * 10^8$, that is, it is only slightly below the maximal upper bound imposed by running-time constraints. Therefore, the main message of these entries in the table is just the following one: Learning REGPAT_2^6 and REGPAT_3^4 for alphabet with four or more symbols is clearly beyond what the current implementation is capable to handle; maybe that on a faster computer with less severe running time constraints, Algorithm ***HLRP*** has a similar performance for the alphabet $\{0,1,2,3\}$ as the current implementation has for the alphabet $\{0,1,2\}$. The next experiments deal with the case where not only the terminals but also the exact form of the pattern is selected randomly. Based on the reasons just outlined, the alphabet $\{0,1,2\}$ was chosen for these experiments.

Experimental Setting 2 *Figure 4 shows experiments to learn patterns over the alphabet $\{0,1,2\}$ where the pattern π is chosen by iteratively executing always the first of the following cases which applies.*

- *When generating the first symbol of π, this is taken to be a variable.*
- *If the currently generated part of π contains less than m terminals and ends with a variable, then the next symbol is a terminal where each digit has the probability $1/3$.*
- *If the currently generated part of π contains less than m terminals and ends with one, two or three terminals following the last variable, then one adds with probability $1/2$ a further variable and with probability $1/6$ the digit a for $a = 1,2,3$.*
- *If the currently generated part of π contains less than m terminals and its last four symbols are all terminals then one adds a variable.*
- *If the currently generated part contains m terminals, then one adds a variable at the end and terminates the procedure since the pattern is complete.*

The first entry in the table of Figure 4 is the parameter m of the above procedure. For $m = 5,6,\ldots,15$, this protocol was used to generate 1000 patterns and run the learning algorithm on these pattern-languages.

The experiments shown in Figure 4 gave that for $m \leq 13$ most errors were due to conjecturing false terminals. Furthermore, for $m = 12,13,14,15$, the number of learning processes which failed in all 10 trials to give a hypothesis, was $1,12,97,109$, respectively, compared to $16,33,66,96$ errors due to incorrectly conjectured patterns.

Note that the probability to draw the smallest string generated by the pattern is $3^{-15} * 16^{-1} * 17^{-1} = 1/3902902704$ for $m = 15$. So the expected value for the number

m	correct	repetitions	without pause	with pause	% used
5	1000	0	13666.581	303467.013	2.95
6	1000	0	30471.300	854851.633	1.30
7	1000	0	69797.807	2504307.913	0.59
8	999	1	78186.534	3526912.510	0.47
9	997	13	134352.582	7190948.532	0.25
10	997	71	177112.452	11066283.854	0.19
11	992	244	259912.841	19128675.970	0.13
12	983	351	284064.711	24450019.647	0.11
13	955	568	365624.652	35745664.161	0.10
14	898	974	511091.000	54322939.087	0.09
15	795	1850	779662.091	94019439.240	0.08

Figure 4: Learning regular patterns with $\Sigma = \{0, 1, 2\}$ following Setting 2.

of example to be drawn until one has seen this string is approximately 3902902704 which is by a factor of approximately 41.51 above the average number 94019439.240 of examples drawn by the algorithm. This shows that the current algorithm is better than the trivial heuristic to draw sufficiently many examples such that the shortest example has shown up with sufficiently high probability and then to run the part of Algorithm ***HLRP*** which figures out the positions of the variables in the pattern.

5 Conclusion

In Theorem 1, it is shown that the class of c-regular pattern languages is probabilistically exactly learnable from any text drawn from a probability distribution *prob* satisfying

$$prob(\sigma) = \frac{|\Sigma|^{-|\sigma|}}{pol(|\sigma|)} \text{ where } \sum_{n=0}^{\infty} \frac{1}{pol(n)} = 1,$$

pol is a $\{2, 3, \ldots\}$-valued increasing polynomial and $prob(\#)$ is chosen accordingly. Actually, Algorithm ***LRP*** also works for distributions which are sufficiently near to the just mentioned ones and succeeds on texts as defined in Definition 5. Proposition 2 extends the learning algorithm to the case where patterns starting or ending with a constantly bounded number of terminals is permitted. Algorithm ***HLRP*** is a heuristic based on Algorithm ***LRP***, which has been implemented to learn the class REGPAT_1. Its parameters have been determined experimentally. This permits to learn the classes REGPAT_1^m faster for small m but these improved bounds are not theoretically generalized for all m. Furthermore, only small alphabets Σ have been considered, all experiments were run for alphabets of sizes $2, 3, 4, 5$ only. An interesting observation is that by skipping long examples, Algorithm ***HLRP*** has an experimentally improved performance compared to the version where all non-pause examples are taken into account. The intuitive reason is that long examples reflect less about the pattern to be learned and have more random components which might lead the learner to wrong conclusions. While Algorithm ***HLRP*** gave very good results for REGPAT_1, its performance for REGPAT_c with $c > 1$ was not so convincing. The difficult part is to figure out the order of the terminals in the string of the pattern while, for alphabet size $3, 4, 5$, it was easy to figure out between which terminals is a variable and between which not. On one hand, for alphabets having a finite size of 3 or more, future work should more focus on improving the method determining the sequence of terminals in a regular pattern than on determining the position of the variables. On the other hand, this second part needs attention for the special case of the alphabet $\{0, 1\}$ since there the solution is not unique.

Acknowledgment

The authors would like to thank Phil Long for fruitful discussions.

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] I. Bratko and S. Muggleton. Applications of inductive logic programming. *Communications of the ACM*, 1995.

[4] 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 1996*), volume 1178 of *Lecture Notes in Computer Science*, pages 95–104, Springer, 1996.

[5] 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 1998*.

[6] J. Case, S. Jain, S. Lange and T. Zeugmann. Incremental concept learning for bounded data mining. *Information and Computation*, 152(1):74–110, 1999.

[7] T. Erlebach, P. Rossmanith, H. Stadtherr, A. Steger and T. Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261(1):119-156, 2001.

[8] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[9] T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33(10):305–308, 1990.

[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*, volume 2533 of Lecture Notes in Artificial Intelligence, pages 308–320. Springer, 2002.

[20] D. Reidenbach. A discontinuity in pattern inference. In V. Diekert and M. Habib, editors, *STACS 2004, 21st Annual Symposium on Theoretical Aspects of Computer Science, Montpellier, France, March 25-27, 2004, Proceedings*. volume 2996 of *Lecture Notes in Computer Science*, pages 129140. Springer, 2004.

[21] 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.

[22] R. Reischuk and T. Zeugmann. An average-case optimal one-variable pattern language learner. *Journal of Computer and System Sciences* 60(2):302-335, 2000.

[23] 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

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

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

[26] 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.

[27] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara and S. Arikawa. Knowledge acquisition from amino acid sequences by machine learning system BONSAI. *Transactions of Information Processing Society of Japan*, 35:2009–2018, 1994.

[28] 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.

[29] T. Shinohara. Inferring unions of two pattern languages. *Bulletin of Informatics and Cybernetics*, 20:83–88, 1983.

[30] 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.

[31] T. Shinohara and S. Arikawa. Pattern inference. In K. P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *Lecture Notes in Artificial Intelligence*, pages 259–291. Springer, 1995.

[32] T. Shinohara and H. Arimura. Inductive inference of unbounded unions of pattern languages from positive data. *Theoretical Computer Science*, 241:191–209, 2000.

[33] R. Smullyan. *Theory of Formal Systems, Annals of Mathematical Studies, No. 47.* Princeton, NJ, 1961.

[34] L.G. Valiant. A theory of the learnable. *Communications of the ACM* 27:1134–1142, 1984.

[35] 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.

[36] 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.