On Automatic Families

Sanjay Jain, Yuh Shin Ong, Shi Pu and Frank Stephan

S. Jain, Y. S. Ong, S. Pu: School of Computing, National University of Singapore, Block COM1, 13 Computing Drive, Singapore 117417
Email: sanjay@comp.nus.edu.sg, yuhshin@gmail.com and pushi@nus.edu.sg
F. Stephan: Department of Mathematics, National University of Singapore Block S17, 10 Lower Kent Ridge Road, Singapore 119076
Email: fstephan@comp.nus.edu.sg

This paper summarises previous work on automatic families. It then investigates a natural size measure for members of an automatic family: the size of a member language in the family is defined as the length of its smallest index. This measure satisfies various properties similar to those of Kolmogorov complexity; in particular the size of a language depends only up to a constant on the underlying automatic family. This family of size measures is extended to a measure on all regular sets. This extension is given by the maximum number of states visited in some run of the minimal deterministic finite automaton recognising the language. Furthermore, a characterisation is given regarding when a class of languages is a subclass of an automatic family.

Sanjay Jain and Frank Stephan are supported in part by NUS grant numbers R252-000-308-112 and R146-000-114-112.

1. Introduction

Automatic families are uniformly regular families of sets which are mainly used in inductive inference (learning theory) as a way to represent hypothesis spaces with decidable first-order theory. For such spaces, one can decide whether they are explanatorily learnable in the limit and have effective procedures to generate such learners. Furthermore, they have also been used to study various other learnability notions. Automatic families are closely related to automatic graphs and automatic structures. All these notions are based on the notion of a finite automaton. The present work formalises the notion of a size of a language with respect to an automatic class. It also relates this size to various measures of the size of a language derived from its minimal deterministic finite automaton. Furthermore, an overview of related properties and applications of automatic families is given.

The basic notion of the field is that of a finite automaton. Informally, a finite automaton could be considered as an algorithm which reads a word from the left to the right and has a constant-sized memory; when it reaches the right end of the word, the algorithm says either "accept" or "reject". This informal description permits to write algorithms representing finite automata quite well; on the other hand, when one wants to prove properties of finite automata, it is better to write them as a finite set of states together with a state-transition table, a starting state and a set of final states. The automaton then reads the symbols of the word to be checked from the left to the right and each time changes its state based on a transition-table which is a set of triples of the form (old state, current symbol, new state). Ideally, for every old state and current symbol, there is a unique triple in the table; an automaton with this property is called a "deterministic finite automaton" (dfa) and every finite automaton can be converted into an equivalent automata of this form, which accepts the same language as the original automaton. Now, for studying automatic structures, the key generalisation is to permit automata to track various inputs simultaneously, provided that the reading of these inputs is synchronised. Informally, the algorithm consists of one loop. In each iteration of the loop the algorithm first reads, from each of those inputs which are not already exhausted, one symbol and then updates its internal memory accordingly; the memory is restricted to a constant size. Here, the algorithm knows when an input is exhausted and when not, so this piece of information can be taken into account. When every input is exhausted, the algorithm says "accept" or "reject". The fact that all inputs are read at the same speed is important, otherwise the model would become too powerful and certain undecidability problems of the theory of such models would arise. Mathematically, one can also map this back to the case of a single input which is formed as the convolution of the inputs. The convolution $conv(x, y)$ of two inputs x and y would consist of symbols which are pairs of the corresponding symbols from x and y; in the case that one of these input words is shorter than the other, it is brought up to the same length by appending the special symbol $\diamond$ sufficiently often; the symbol $\diamond$ is not contained in any of the input alphabets used for x and y. One can define *conv* on more than two arguments similarly. The next algorithm is an example which decides the lexicographic order of two strings x and y; the algorithm uses an underlying ordering $<$ on Σ:

(1) If the input x is exhausted then accept and terminate.
(2) If the input y is exhausted then reject and terminate.
(3) Read the current symbol a of the input x and the current symbol b of the input y.
(4) If $a < b$ then accept and terminate.
(5) If $b < a$ then reject and terminate.
(6) Go to (1).

Note that this algorithm also accepts if both words are equal; so it accepts if $x \leq_{lex} y$ and rejects if $y <_{lex} x$. Furthermore, the algorithm produces an early decision, when possible. So when comparing 25880 with 2593 the algorithm would already make the decision after having read 258 from 25880 and 259 from 2593. Such an early decision is permitted for easier formulations of the algorithms, although formally the algorithm has to scan the full input.

Besides automatic relations like the lexicographic ordering, one can also define automatic functions where the automaton recognises a function f as follows: it reads convolution of x and y and accepts iff x is in the domain of f and $y = f(x)$. An automatic family is a structure with the following properties: One has a domain D and an index domain I which are both regular sets plus an automatic relation on $I \times D$ defining a family $\{L_i : i \in I\}$ such that $x \in L_i$ iff the underlying relation contains (i, x). Such a family defines a class of subsets of D and two families have the same range iff they contain the same languages. In many cases it is also handy to have that the indexing is a one-one indexing, that is, $L_i \neq L_j$ for different $i, j \in I$. However, this property is not mandatory within the present work. An important result from the early days of automatic structures [8, 9, 17] is that whenever a relation or function is first-order definable using other automatic relations or functions then it is itself automatic. Furthermore, the first-order theory of automatic structures is decidable. These two results give this field some importance in model checking and similar applications and the two results are also frequently used when constructing learners in inductive inference. The next examples of automatic families illustrate the concept.

Example 1.1: Let $\Sigma = \{0, 1, 2\}$ and $D = \Sigma^*$.

The family of all $L_i = \Sigma^{|i|}$, where $i \in \{0\}^*$, is an automatic family. The automaton recognising the family accepts $conv(i, x)$ iff i and x have the same length.

The family of all $L_i = \{x \in D : x \text{ extends } i\}$ with $i \in \Sigma^*$ is an automatic family; an example member of it is $L_{001} = \{001, 0010, 0011, 0012, 00100, 00101, \ldots, 00122, 001000, 001001, \ldots\}$. The automaton recognising the family accepts $conv(i, x)$ iff every symbol of i appears at the same position in x.

The family $L_{conv(i,j)} = \{x : i <_{lex} x <_{lex} j\}$ of all $i, j \in \Sigma^*$ with $i <_{lex} j$ is also automatic; note that in some special cases like $j = i0$ the language $L_{conv(i,j)}$ is empty. The automaton recognising the family is mainly checking whether x is between the two boundaries i and j which are coded up in the index $conv(i, j)$.

The family of all L_{i3a3b} with $i3a3b \in I = \Sigma^* \cdot \{3\} \cdot \Sigma \cdot \{3\} \cdot \Sigma$ and $L_{i3a3b} = \{ixaybz : x, y, z \in \Sigma^*\}$ is automatic. For example, $L_{00123231}$ is given by the regular expression $0012 \cdot (0 + 1 + 2)^* \cdot 2 \cdot (0 + 1 + 2)^* \cdot 1 \cdot (0 + 1 + 2)^*$. The automaton recognising the automatic family has to memorise the values of a and b when reaching the corresponding position in the input $conv(i3a3b, u)$ as a and b can occur in u much later than in $i3a3b$. This is possible as a finite automaton can memorise a constant amount of information.

Example 1.2: Using a certain regular domain D, one can also code the natural numbers with addition, the relation $<$ and a predicate Fib recognising the Fibonacci numbers [28]. If now $\phi(x, i, j, k)$ is a first-order formula with four inputs defined using $+$, $<$, Fib and natural numbers then the family of all $L_{conv(i,j,k)} = \{x \in D : \phi(x, i, j, k) \text{ is true}\}$ is an automatic family where the index-domain is the set of all convolutions of three elements of D. An example for such a formula ϕ is $\phi(x, i, j, k) \Leftrightarrow \exists y \exists z [i < x + y < j \wedge x + x = z + z + z + 2 \wedge Fib(x + y + y + k)]$.

Note that the above example is more in the traditional style of automatic structures where all aspects of coding can be freely chosen in order to meet the specification. Automatic families usually are a bit more fixed; here one wants to find an automatic indexing of a given class of regular languages; that is, while the indexing is considered to be "chosen", the languages inside the class and the domain D are more considered as "given". The results in the next section will establish various facts on the indexings which show that the indexings are not completely free, but some aspects of them are determined by the class which has to be represented by the automatic family. See [4, 5, 8, 9, 16, 17, 25, 26, 28] for more information on automatic structures.

2. The size of languages inside a family

Having the concept of an automatic family, one can use its indexing in order to introduce a measure for the size of the languages inside the given family.

Definition 2.1: Given an automatic family $\mathcal{L}$ and a language $R \in \mathcal{L}$, let $d_{\mathcal{L}}(R) = \min\{|i| : i \in I \wedge L_i = R\}$ be the size of R.

The next result shows that the size depends only up to an additive constant on the chosen automatic family; so enlarging the underlying family or just changing its indexing has not much impact on the size of a languages inside the family.

Proposition 2.2: *Let $\mathcal{L} = \{L_i : i \in I\}$ and $\mathcal{H} = \{H_j : j \in J\}$ be two automatic families. Then there is a constant c such that $d_{\mathcal{L}}(R)$ and $d_{\mathcal{H}}(R)$ differ by at most c, for all $R \in \mathcal{L} \cap \mathcal{H}$.*

Proof: The basic idea is to look at the set

> $O = \{conv(i, j) : i \in I$ and $j \in J$ and $L_i = H_j$ and i, j are the length-lexicographically least indices of their respective sets$\}$.

Note that $(i = i' \vee j = j') \Rightarrow (i = i' \wedge j = j')$ whenever $conv(i, j)$, $conv(i', j') \in O$. The length-lexicographic order $<_{ll}$ is automatic. Hence O is first-order definable from automatic relations:

$$\begin{aligned} conv(i,j) \in O \Leftrightarrow \quad & i \in I \wedge j \in J \wedge \forall x \in D\,[x \in L_i \Leftrightarrow x \in H_j] \\ & \wedge \forall i' \in I\,[[\forall y \in D\,[y \in L_{i'} \Leftrightarrow y \in L_i]] \Rightarrow i \leq_{ll} i'] \\ & \wedge \forall j' \in J\,[[\forall y \in D\,[y \in H_{j'} \Leftrightarrow y \in H_j]] \Rightarrow j \leq_{ll} j']. \end{aligned}$$

From this fact it follows, by a result of Khoussainov and Nerode [17], that the set O is regular.

Now one uses the following version of the pumping lemma: If R is a regular language then there is a constant c such that for all $uvw \in R$ with $|v| > c$ it also holds that $utw \in R$ for some string t which consists of up to c symbols taken from v.

This version of the pumping lemma is now applied to O. Let c be the corresponding constant. Given any $conv(i, j) \in O$, one takes u to be the prefix of length $\min\{|i|, |j|\}$ of $conv(i, j)$, v to be the suffix of length

$\max\{|i|,|j|\} - \min\{|i|,|j|\}$ and w to be the empty string. Assume that $|v| > c$ and let t be formed by up to c characters from v as described in the pumping lemma above. Now $ut \in O$. There are two cases. If $|j| > |i| + c$ then there is a shorter $j' \in J$ with $conv(i, j') = ut$ and $conv(i, j') \in O$; otherwise $|i| > |j| + c$ and there is a shorter $i' \in I$ with $conv(i', j) = ut$ and $conv(i', j) \in O$. So in either case, there is besides $conv(i, j)$ another pair, namely $conv(i', j)$ or $conv(i, j')$, in O; this pair coincides with $conv(i, j)$ in one but not in both coordinates in contradiction to the choice of O. Hence it cannot happen that $|v| > c$. Thus, the length of i and j differ by at most the constant c. $\square$

This result is a bit parallel to the corresponding result in the field of Kolmogorov complexity [20] that the Kolmogorov complexity of an object depends only up to a constant on the underlying universal machine. The main difference is that here the measures $d_{\mathcal{L}}$ are only defined on a subfamily $\mathcal{L}$ of the regular languages and not on all of them; this invokes some problems and in the following it is investigated to which degree one can overcome these problems. Before doing this in the next sections, first a parallel to Kolmogorov complexity is pointed out: Boolean operations and images of sets under functions essentially have the complexity of the input sets.

Remark 2.3: Let an automatic family $\mathcal{L} = \{L_i : i \in I\}$ and an automatic predicate Φ mapping n inputs $L_{i_1}, L_{i_2}, \ldots, L_{i_n}$ to a new set $\Phi(L_{i_1}, L_{i_2}, \ldots, L_{i_n})$ be given. Then there is a new automatic family $\mathcal{H}$ such that for every $i_1, i_2, \ldots, i_n$ it holds that $\Phi(L_{i_1}, L_{i_2}, \ldots, L_{i_n})$ is contained in $\mathcal{H}$ and $d_{\mathcal{H}}(\Phi(L_{i_1}, L_{i_2}, \ldots, L_{i_n})) \leq c + \max\{d_{\mathcal{L}}(L_{i_1}), d_{\mathcal{L}}(L_{i_2}), \ldots, d_{\mathcal{L}}(L_{i_n})\}$, where c is a constant only depending on $\mathcal{L}$, $\mathcal{H}$ and Φ. Examples of such operators Φ are the Boolean operations like union, intersection and complementations as well as forming the range under an automatic function f: $\Phi(L) = \{f(x) : x \in L\}$.

3. Universal Complexity Measures

In the following, let A_R be the smallest deterministic finite automaton accepting R; A_R has to be complete, that is, for every state p and every symbol $a \in \Sigma$ there is exactly one state q such that A_R goes from p to q on input a. Let $d_{dfa}(R)$ denote the number of states of A_R and $d_{run}(R)$ denote the maximum n such that, for some input word x, A_R on x goes through n different states. Note that $d_{run}(R) \leq d_{dfa}(R)$. The next result establishes an inequality for the converse direction. This inequality witnesses that, for each n, there are only finitely many R with $d_{run}(R) = n$. Thus d_{run} satisfies some minimum requirement for measuring the size of a language adequately.

Proposition 3.1: *Let $R \subseteq \Sigma^*$ be a regular language. If Σ has at least two members then*

$$d_{dfa}(R) \leq \frac{|\Sigma|^{d_{run}(R)} - 1}{|\Sigma| - 1};$$

otherwise $d_{dfa}(R) = d_{run}(R)$.

Proof: Recall that A_R is the minimal deterministic finite automaton recognising R. One can look at the finite tree T of all runs of A_R in which no state is visited twice. The height of this tree T is at most $d_{run}(R) - 1$. Furthermore, every state of A_R occurs in this tree T as it can be reached by a repetition-free run. In the case that Σ has exactly one element, T has $d_{run}(R)$ members and $d_{dfa}(R) = d_{run}(R)$. In the case that Σ has at least two members, the formula

$$d_{dfa}(R) \leq |T| \leq |\Sigma|^0 + |\Sigma|^1 + \ldots + |\Sigma|^{d_{run}(R)-1} = \frac{|\Sigma|^{d_{run}(R)} - 1}{|\Sigma| - 1}$$

provides an upper bound on the number of members of T and thus on the value $d_{dfa}(R)$. □

Remark 3.2: The exponential bound in the case of the alphabet Σ having at least two symbols looks large, but the gap cannot be made much smaller.

The proof of this fact and later results use the notion of the derivative: $L[x] = \{y : xy \in L\}$ is called the *derivative of L at x*. Note that $d_{dfa}(L)$ coincides with the number of distinct derivatives of L.

The exponential bound is now witnessed by the example family $L_0, L_1, L_2, \ldots$ where $L_n = \{xx : x \in \Sigma^n\}$. Then $d_{run}(L_n) = 2n + 2$ while $d_{dfa}(L_n) \geq |\Sigma|^n$, as for every $x \in \Sigma^n$ the derivative $L_n[x] = \{x\}$ is encoding x.

The following connections hold between the size based on automatic families and these two measures.

Theorem 3.3: *For every automatic family $\mathcal{L}$ there is a constant c such that $d_{dfa}(R) \leq d_{\mathcal{L}}(R) \cdot c + 1$ for all $R \in \mathcal{L}$.*

Proof: Let Σ be the alphabet satisfying $R \subseteq \Sigma^*$ for all $R \in \mathcal{L}$ and let A be the automaton recognising the automatic class. That is, there is a regular set I of indices such that $\mathcal{L} = \{L_i : i \in I\}$ and a deterministic finite automaton A which accepts $conv(i, x)$ iff $i \in I$ and $x \in L_i$. Let c be the number of states of A. Let n be the length of the index i of some L_i. Now one constructs an automaton $Comb(A, i)$ which recognises the language L_i and which has at most $n \cdot c + 1$ states. $Comb(A, i)$ is constructed as follows.

- The alphabet of $Comb(A, i)$ is Σ.
- For $m < n$ let $X_m = \Sigma^m$ and for $m = n$ let $X_m = \bigcup_{k \geq n} \Sigma^k$.
- The set of states of $Comb(A, i)$ is the union of sets $Q_0, Q_1, \ldots, Q_n$, where Q_m consists of all pairs (q, m) with q being a state in A such that for some $x \in X_m$, A is in state q after reading the first $|x|$ symbols of $conv(i, x)$.
- For $m < n$, let there be a transition from (q, m) to $(p, m+1)$ on symbol a if there is a word $x \in X_{m+1}$ of length $m + 1$ such that A is in state q after reading first m symbols of $conv(i, x)$, A is in state p after reading first $m + 1$ symbols of $conv(i, x)$ and the last symbol of x is a. Furthermore, let there be a transition from (q, n) to (p, n) on symbol a if there is a word $x \in X_n$ such that A after reading $conv(i, x)$ is in state q and after reading $conv(i, xa)$ is in state p. There are no other transitions.
- Note that Q_0 contains only $(s, 0)$ where s is the starting symbol of A; $(s, 0)$ is then the starting state of the automaton $Comb(A, i)$.
- The accepting states of $Comb(A, i)$ are all states of the form (p, m) such that there is an $x \in X_m \cap L_i$ and $Comb(A, i)$, on input x, goes from state $(s, 0)$ to state (p, m).

Note that $|Q_0| = 1$ and, in general, $|Q_m| \leq c$. Hence $Comb(A, i)$ has at most $n \cdot c + 1$ states. Furthermore, one can show that $Comb(A, i)$, on input x, goes from state $(s, 0)$ to state (q, m) iff $x \in X_m$ and A, after reading first $|x|$ symbols of $conv(i, x)$, goes from starting state s to state q. Assume now that x, y are such that $Comb(A, i)$ is in the same state (q, m) after processing x, y. Clearly $x, y \in X_m$. If $m < n$, then after reading the first m symbols of $conv(i, x)$ and $conv(i, y)$, respectively, A is in the same state. Therefore A accepts $conv(i, x)$ iff A accepts $conv(i, y)$; hence $x \in L_i$ iff $y \in L_i$. Furthermore, if $m = n$, then after reading $conv(i, x)$ and $conv(i, y)$, A is in the same state q. Again $x \in L_i$ iff $y \in L_i$. It follows that $H_{q,m} = \{x \in \Sigma^* : Comb(A, i)$, on input x, goes from state $(s, 0)$ to state $(q, m)\}$ is either a subset of L_i or disjoint to L_i. So, by the definition of $Comb(A, i)$, the automaton $Comb(A, i)$ accepts the members of $H_{q,m}$ iff $H_{q,m} \subseteq L_i$ and rejects the members of $H_{q,m}$ iff $H_{q,m} \cap L_i = \emptyset$. It follows that $Comb(A, i)$

recognises the language L_i. The minimal automaton of L_i has at most as many states as $Comb(A, i)$. So $d_{dfa}(L_i) \leq |i| \cdot c + 1$ and therefore the theorem follows. □

Remark 3.4: The multiplicative constant c in the above theorem is indeed needed: If $\mathcal{L}$ is the class of all finite languages consisting of up to c strings, then each automaton accepting the language $L_n = \{0^d 1^n 0^d : d < c\}$ needs at least $c \cdot n$ states in order to memorise the number of 0s and then to count the number of 1s before comparing the number of 0s after the block of 1s with the memorised value. Also, one can easily verify that — up to an additive constant — $d_{\mathcal{L}}(L_n) = n$.

The next result is the main contribution of this paper. It shows that d_{run} is a measure which meets the expectation in at least one point: given any automatic class $\mathcal{L}$, the measures $d_{\mathcal{L}}$ and d_{run} coincide on $\mathcal{L}$ up to a constant.

Theorem 3.5: *For every automatic family $\mathcal{L}$ there is a constant c' such that, for all $R \in \mathcal{L}$, the values $d_{run}(R)$ and $d_{\mathcal{L}}(R)$ differ from each other by at most c'.*

Proof: Let $\mathcal{L}$ be the given automatic family and let c be the number of states of the minimal automaton A recognising the family. Let $R \in \mathcal{L}$.

Now it is shown that $d_{run}(R) \leq d_{\mathcal{L}}(R) + c$. For every index $i \in I$, one constructs the automaton $Comb(A, i)$ as described in Theorem 3.3. In any run, this automaton passes through at most $|i| + c$ states; namely for each $m < |i|$ through at most one state in Q_m and through at most c states in $Q_{|i|}$. On input x, the minimal automaton A_R goes through at most as many states as $Comb(A, i)$. Thus the bound obtained is also a bound for $d_{run}(R)$.

Now it is shown that $d_{\mathcal{L}}(R)$ is bounded by $d_{run}(R)$ plus a constant independent of R. For $i \in I$, let $L_i[x] = \{y : xy \in L_i\}$ be a derivative of the language L_i. Now, if $|x| \geq |i|$ then one can produce the following automaton B accepting the derivative $L_i[x]$:

- The set of states of B equals the set of states of A;
- The starting state of B is the state of A after having read the first $|x|$ symbols of $conv(i, x)$;
- The state transition of B from p to q on a symbol a occurs iff A goes on the one-symbol word $conv(\diamond, a)$ from p to q;
- The accepting states of B and A are the same.

Hence the language $L_i[x]$ is recognised by an automaton containing only c states. Let B_q be the automaton B with the starting state q. Now let n be the least positive natural number such that, for every $x \in \{0,1\}^n$, $L_i[x]$ is recognised by some automaton B_q. Note that the choice of q only depends on the state in which A is after having read first $|x|$ symbols of $conv(i, x)$. Without loss of generality, some of the B_q accept the empty string and some do not. Now let $t : \{1, 2, \ldots, c\} \to \{1, 2, \ldots, c\}$ code a finite function satisfying the following conditions:

- if there is $x \in \Sigma^n$ and A is in state b after processing the first n symbols of $conv(i, x)$ then $B_{t(b)}$ recognises $L_i[x]$;
- if there is $y \in \Sigma^*$ with $|y| < n$ such that A is in state b after processing the first n symbols of $conv(i, y)$ then $B_{t(b)}$ accepts the empty string iff $y \in L_i$.

Note that if there are $x, y \in \Sigma^*$ with $|x| = n \wedge |y| < n$ and A being in the same state after processing the first n symbols of $conv(i, x)$ and $conv(i, y)$, respectively, then $x \in L_i$ iff $y \in L_i$ iff the empty string is in $L_i[x]$. So above conditions do not contradict each other and the mapping t exists.

Using n and t, one can code L_i by an index j which is the convolution of t and the first n symbols of i. Note that there is an automatic function f with $f(i) = j$ as above; the reason is that n and t can be defined from i and the indexing of $\mathcal{L}$ using first-order formulas. Let $J = \{f(i) : i \in I\}$ and $H_{f(i)} = L_i$; the set J is regular and the family $\{H_j : j \in J\}$ is automatic.

Now let $i \in I$ and $j = f(i)$. It follows, using the pumping lemma, that every word in L_i of length $d_{run}(L_i)$ is of the form uvw such that $L_i[uv^{|i|}w] = L_i[uvw]$. Hence $L_i[uvw]$ is recognised by one of the automata B_q and so $|j| \leq d_{run}(L_i)$. Thus it holds, for all $R \in \mathcal{L}$, that $d_{run}(R) \geq d_{\mathcal{H}}(R)$. As $d_{\mathcal{L}}(R)$ and $d_{\mathcal{H}}(R)$ differ by at most a constant (Proposition 2.2), it follows that $d_{\mathcal{L}}(R) \leq d_{run}(R) + c''$, for some constant c''.

It follows from above analysis that $d_{run}(R)$ differs from $d_{\mathcal{L}}(R)$ only by a constant independent of R. □

Corollary 3.6: *For every automatic family $\mathcal{L}$ there is a constant c such that $d_{\mathcal{L}}(R) \leq d_{dfa}(R) + c$ for all $R \in \mathcal{L}$.*

The following result shows that — when restricted to an automatic family — the size of Boolean operations and images among the members of the family do not have a much larger size than the corresponding components. This situation is similar to the situation with respect to Kolmogorov complexity.

Theorem 3.7: *For every automatic family $\mathcal{L}$ over domain D and every automatic function f over domain D, there is a constant c such that, for all $L, H \in \mathcal{L}$, it holds that*

- $d_{run}(L \cup H) \leq \max\{d_{run}(L), d_{run}(H)\} + c$,
- $d_{run}(L \cap H) \leq \max\{d_{run}(L), d_{run}(H)\} + c$,
- $d_{run}(D - L) \leq d_{run}(L) + c$ *and*
- $d_{run}(\{f(x) : x \in L\}) \leq d_{run}(L) + c$.

However the first, second and fourth condition of this list do not hold without the restriction to an automatic family.

Proof: The main part of this result follows from the Proposition 2.2, Remark 2.3 and Theorem 3.5. So the rest of the proof is to show that these connections do not hold in general, except for the third condition which is just obtained by interchanging acceptance and rejection inside D.

Let $n \in \{1, 2, 3, \ldots\}$. For the first two conditions one considers the languages $(\Sigma^n)^*$ and $(\Sigma^{n+1})^*$. While $d_{run}((\Sigma^n)^*) = n$ and $d_{run}((\Sigma^{n+1})^*) = n + 1$, it holds that $d_{run}((\Sigma^n)^* \cup (\Sigma^{n+1})^*) = n(n + 1)$ and $d_{run}((\Sigma^n)^* \cap (\Sigma^{n+1})^*) = n(n + 1)$. For the fourth condition, assume $\Sigma = \{0, 1, 2\}$ and consider the automatic function f which interchanges 1 and 2 at every second occurrence of one of these digits. So $f(001001001001) = 001002001002$ and $f(1212121221212121) = 1111111122222222$. Let $L = (0^n1)^*$. Then $\{f(x) : x \in L\} = (0^n10^n2)^* + (0^n10^n2)^* \cdot 0^n1$. While $d_{run}(L) = n + 2$ it holds that $d_{run}(\{f(x) : x \in L\}) = 2n + 3$. □

Remark 3.8: Besides d_{run}, one could also look at the following measure: $d_{rf}(R)$ is the largest number n such that there is an input $x \in \Sigma^*$ on which the minimal automaton A_R for R goes through exactly n states without repeating any of them.

Note that $d_{rf}(R) \leq d_{run}(R) \leq d_{dfa}(R)$ for every regular language R and the proof of Proposition 3.1 gives directly that

$$d_{dfa}(R) \leq \frac{|\Sigma|^{d_{rf}(R)} - 1}{|\Sigma| - 1}$$

for the case that Σ has at least two elements. Furthermore, Theorem 3.5 holds also with d_{rf} in place of d_{run}.

Although there are many parallel results for these two measures, d_{run} and d_{rf} are not identical. Consider the language $R = \{0^n, 1^n\}^*$ with $n \geq 2$. The minimal automaton A_R has the states reached by 0^m with $m < n$, 1^m with $m < n$ plus one rejecting state which is never left; as the initial state is double counted in this list, there are in total $2n$ states and $d_{dfa}(R) = 2n$. On the word $0^n 1^n 01$ all states are visited and therefore $d_{run}(R) = 2n$. However $d_{rf}(R) = n + 1$ and this maximum number is taken on the input $0^{n-1}1$.

4. Characterising Automatic Families

The central question of this section is: When is a class $\mathcal{L}$ of regular languages a subclass of some automatic family. The answer is that every language in the class must be representable by an automaton of a specific form.

Theorem 4.1: *A class $\mathcal{L}$ is a subclass of an automatic family iff there is a constant c such that every $R \in \mathcal{L}$ is accepted by a deterministic finite automaton whose states can be partitioned into sets $Q_0, Q_1, Q_2, \ldots, Q_n$ satisfying the following conditions: each set Q_m has up to c states; Q_0 consists exactly of the starting state; if there is a transition from a state p to a state q, then there are r, r' with $p \in Q_r$, $q \in Q_{r'}$ and $r' = \min\{r + 1, n\}$.*

Proof: Given an automatic class $\mathcal{L}$ with an automaton A recognising the class, one can construct, for each $i \in I$, the automaton $Comb(A, i)$ to recognise L_i as in Theorem 3.3; it is easy to see that the automaton is of the above form.

For the converse direction, consider the class $\mathcal{H}$ of all languages which are accepted by an automaton of the above form with a given fixed constant c. The indices j of $\mathcal{H}$ consist of symbols coding $Q_0, Q_1, \ldots, Q_n$, respectively. For each Q_m it is coded which of the states of Q_m (numbered as $1, 2, \ldots, c$) are accepting and what transitions are there from Q_m to $Q_{\min\{m+1,n\}}$ based on various inputs from Σ. Without loss of generality state 1 from Q_0 is the starting state and that does not need to be coded. Now $J = \Gamma^*$ where

$$\Gamma = \{\text{reject}, \text{accept}\}^c \times \{1, 2, \ldots, c\}^{|\{1,2,\ldots,c\} \times \Sigma|},$$

that is, where each symbol in Γ codes the acceptance of the c states in Q_m plus the transition table to $Q_{\min\{m+1,n\}}$. Without loss of generality, the empty string just codes the empty language. This convention permits to avoid a domain check for the index j.

As a finite automaton is the same as an algorithm working from the left to the right through the word with constant memory, the algorithm is now given more explicitly than it would be in the case of an automaton. It runs in stages and it memorises information which can be stored in constantly many bits; note that the number of these bits depends on the value of c but not of the value of n. For easier readability, this information is memorised in variables which are initialised in the first two steps. The algorithm reads in steps (2) and (4) the symbols unless the end of the corresponding inputs (j and x) is reached in which case the last value is not overwritten. If j is empty, then the algorithm rejects all x.

(1) Variables: status $\in$ {reject,accept}; state $\in \{1, 2, \ldots, c\}$; $a \in \Sigma$ (current input symbol to be processed); b (table of current Q_m).
Let state = 1 and b be a code such that 1 is a rejecting state and all transitions are from 1 to 1.
(2) If there is some symbol of the code of j to be read
then read b
else let b unchanged.
(3) If the current value of state according to b is an accepting state
then let status = accept
else let status = reject.
(4) If there is some symbol of the input word to be read
then read a
else accept/reject according to status and terminate.
(5) Decode from b the new value of state in dependence of a and of the current value of state.
(6) Go to (2).

The verification is left to the reader, as it is straightforward but lengthy. Note that $\mathcal{L}$ is automatic iff the set $\{j \in J : H_j \in \mathcal{L}\}$ is regular. □

The characterisation from Theorem 4.1 could be put into a more general form. Recall that $R[x]$ is the set $\{y \in \Sigma^* : xy \in R\}$.

Theorem 4.2: *A class $\mathcal{L}$ is a subclass of an automatic family iff there is a constant c such that $R \in \mathcal{L}$ iff there is an n such that:*

- *For every m there are at most c different derivatives $R[x]$ with $x \in \Sigma^m$;*
- *There are at most c different derivatives $R[x]$ with $x \in \Sigma^* \wedge |x| \geq n$.*

This result has an interesting corollary for the case of the unary alphabet.

Corollary 4.3: *If $\Sigma = \{0\}$ then $\mathcal{L}$ is contained in an automatic family iff there is a finite class $\mathcal{F}$ of regular languages such that every language in $\mathcal{L}$ is equal to $H \cup (0^m \cdot L)$ for some m, some $L \in \mathcal{F}$ and some $H \subseteq \{0^\ell : \ell \leq m\}$.*

In learning theory, an important and well-studied family is that of the pattern languages [2, 18, 27]. Here a pattern is a string consisting of constants and variables, where the language generated by the pattern is the set of all those words which are obtained by replacing variables by strings. If the strings replacing the variables are permitted to be empty, then these languages are called "erasing pattern languages"; if these strings are not permitted to be empty, then these languages are called "non-erasing pattern languages". The learnability of pattern languages has been extensively studied and while there is an explanatory learner for the class of non-erasing pattern languages [2], such a learner does not exist in the case of the erasing pattern languages [24]. In particular, the class of "regular pattern languages" is quite important and one might ask what the automatic counterpart of it is. Here Shinohara [27] defined that a pattern is called a *regular pattern* iff it contains every variable at most once. So if $\Sigma = \{0, 1, 2\}$ and the pattern is $i = 01121x121y112z$ then the language generated by this pattern is given by the regular expression $01121 \cdot (0+1+2)^* \cdot 121 \cdot (0+1+2)^* \cdot 112 \cdot (0+1+2)^*$; in the non-erasing case, one would have to replace "$(0+1+2)^*$" by "$(0+1+2) \cdot (0+1+2)^*$" as every variable represents at least one letter.

Theorem 4.4: *Let $\mathcal{L}$ be a class of erasing pattern languages, each generated by a regular pattern. The class $\mathcal{L}$ is contained in an automatic family iff there is a constant c such that, in every pattern of a language in the family, there are at most c constants after the occurrence of the first variable.*

Proof: In the case that Σ has only one element, say 0, variables and constants commute and every pattern language in the class is generated by a pattern where the constants come first and the variables come last. One can without loss of generality then assume that there is at most one variable. Hence over the unary alphabet, the class of all erasing regular pattern languages is contained in the automatic family of the languages generated by one of the patterns in $\{x, 0x, 00x, 000x, \ldots\} \cup \{\epsilon, 0, 00, 000, \ldots\}$. So assume the case that Σ has at least two elements.

Given the constant c, it is first shown that there is an automatic family containing all the erasing regular pattern languages with up to c constants after the first occurrence of a variable. Note that there are only finitely many regular patterns which start with a variable and contain up to c constants; the reason is that a double variable xy has the same effect as a single variable in the case that variables can take the empty word and are not repeated. Now let Γ be the set of all these patterns and assume that Γ is coded such that it is disjoint to Σ. One chooses as an indexing the set $\Sigma^*\Gamma$. If L_g is the language generated by $g \in \Gamma$ then extend this definition to $L_{ig} = i \cdot L_g$ for $i \in \Sigma^*$. The family of all L_{ig} is automatic as the finite automaton recognising $\{conv(ig, x) : x \in L_{ig}\}$ accepts $conv(ig, x)$ iff $x = iy$ for some y and $y \in L_g$; note that the latter can be checked as there are only finitely many languages of the form L_g with $g \in \Gamma$ — thus, one can combine the corresponding finite automata to get an automaton for checking whether $y \in L_g$.

For the converse direction, assume that an automatic family $\{L_i : i \in I\}$ of erasing regular pattern languages is given. As shown in the proof of Theorem 3.5, there is a constant c such that, for every $i \in I$, there are at most c different derivatives $L_i[x]$ with $|x| \geq |i|$. Now assume that $i \in I$ is the index of the language generated by a pattern of the form $ux_0a_1x_1 \ldots a_nx_n$ where $u \in \Sigma^*$, $a_1, \ldots, a_n \in \Sigma$, x_0 is a variable and $x_1, \ldots, x_n$ are each either a variable or the empty string. Furthermore, let $b \in \Sigma - \{a_1\}$. Now choose k larger than the length of i. Then there are n different derivatives $L_i[ub^k a_1 \ldots a_m]$ correspondingly with the shortest word $a_{m+1} \ldots a_n$, where $m \in \{1, 2, \ldots, n\}$. Thus, as $|ub^k| \geq |i|$ the inequality $n \leq c$ holds and therefore the pattern has at most c constants after the occurrence of the variable x_0. $\square$

By using essentially the same proof, one can obtain the counterpart of this result for non-erasing pattern languages. As here the variables have at least the length 1, the patterns x and yz do not generate the same language; hence one has to bound the number of variables as well.

Corollary 4.5: *Let $\mathcal{L}$ be a class of non-erasing pattern languages, each generated by a regular pattern. The class $\mathcal{L}$ is contained in an automatic family iff there is a constant c such that every language in $\mathcal{L}$ can be generated by a pattern which has at most c variables and constants after the first block of variables. That is, if the pattern contains a constant a after some variable x, then there are at most c variables and constants after the above xa.*

A direct application of this result is the following: The family of languages generated by the patterns $\{x, 0x, 00x, 000x, 0000x, 00000x, \ldots\}$ is automatic while the family of languages generated by the patterns $\{x_00, x_00x_1, x_00x_1x_2, x_00x_1x_2x_3, \ldots\}$ is, in the case that $|\Sigma| \geq 2$, not automatic.

Note that every regular pattern generates a regular language [27]. Reidenbach [23] discussed the converse direction and showed that certain non-erasing patterns generate regular languages although the patterns themselves are not regular; he furthermore noted that in the non-erasing case, a language is either generated only by regular patterns or only by nonregular patterns. Jain, Ong and Stephan [13] show that the converse direction also fails for erasing pattern languages and alphabet sizes up to 3.

Remark 4.6: Reidenbach [23] provided examples of nonerasing pattern languages which are regular but are not generated by a regular pattern. An example is given by the pattern $xyxz$ which generates the regular language $\cup_{a,b,c \in \Sigma} ab\Sigma^* ac\Sigma^*$ and which, for alphabets of size 2 or more, cannot be generated by a regular pattern.

Jain, Ong and Stephan [13] considered the corresponding question for erasing pattern languages. If Σ has at least four symbols, then every erasing pattern language which is regular is generated by a pattern which does not have repetitions of the variables. However for alphabets of size 1, 2 and 3 there are also erasing pattern languages which are regular sets but need repetitions of variables to be generated. For $\Sigma = \{0,1\}$ the pattern $x_1 x_2 x_3 1 x_2 x_4 x_4 x_5 1 x_6 x_5 x_7$ generates the language $(0+1)^* \cdot 1 \cdot (0+1)^* \cdot 1 \cdot (0+1)^* - 10 \cdot (00)^* \cdot 1$, which is not generated by any regular pattern.

5. Applications of Automatic Families in Learning Theory

Automatic families are a special case of indexed families [1, 19] which are a widely studied subject in inductive inference. Here an indexed family $\{L_i : i \in I\}$ is uniformly recursive, that is, the domain D, the set of admissible indices I and the mapping $i, x \mapsto L_i(x)$ are all recognisable by Turing machines. So automatic families are just the restriction obtained when replacing Turing machines by finite automata with possibly several inputs. Gold [7] formalised the notion of learning and introduced the following notion of *explanatory learning*: A family $\{L_i : i \in I\}$ is explanatorily learnable iff there is a recursive learner which reads more and more data about the language R to be learnt from a text T and outputs a sequence $e_0, e_1, e_2, \ldots$ of indices which syntactically converges to one index $i \in I$ with $R = L_i$. Here a text is an infinite sequence of words containing every member of the set to be learnt but not any nonmember; the words in the text can be in arbitrary and adversary order and repetitions are permitted. The interested reader might find more background information on learning theory in the standard textbooks of inductive inference [14, 22]. Angluin [1] gave a criterion on the learnability of an indexed family which — in the case of automatic families — can be simplified to the following one [11].

Theorem 5.1: *An automatic family $\mathcal{L} = \{L_i : i \in I\}$ is explanatorily* *learnable iff there exists a constant c such that for all $i, j \in I$ the equality $L_i = L_j$ holds whenever $\{x \in L_i : |x| \leq d_{\mathcal{L}}(L_i) + c\} \subseteq L_j \subseteq L_i$.*

In other words, one can build a learner which — when learning R — always conjectures the least i such that all data in L_i shorter than $|i| + c$ have already been observed but no data outside L_i.

Based on this observation, Jain, Luo and Stephan [11] formulated the notion of an automatic learner which is less powerful than a recursive learner. An automatic learner has a long term memory which stores all relevant information about the data observed; this long term memory is a string like any input word, though it might be over a larger alphabet. The learner is then given by an update function

> F: (old long term memory, current data item) $\mapsto$ (new long term memory, new hypothesis).

Starting from an initial value for its long term memory, the learner reads in each round a current datum and updates its memory and the hypothesis according to F. The update function F has to be automatic. Jain, Luo and Stephan [11] showed that not every learnable automatic class can also be learnt by an automatic learner. Indeed even quite simple classes like the one given by $L_i = \Sigma^* - \{i\}$ (where $D = I = \Sigma^*$) is learnable only if the alphabet consists of one symbol; in the case of an alphabet with at least two symbols this class is no longer learnable by an automatic learner, as the automatic learner cannot memorise enough information about the data observed. Special cases considered were those where the long term memory cannot be longer than the longest datum observed so far, where the long term memory is the last hypothesis conjectured (iterative learning) and where the long term memory consists of up to c data items observed previously (bounded example memory); the ability for an automatic learner to learn depends heavily on the nature of such restrictions imposed. Ong [21] investigated the learnability of automatic families of pattern languages and related classes.

Jain, Martin and Stephan [12] considered the setting of robust learning [6, 15, 29] and asked when every translation of an automatic family $\mathcal{L}$ is learnable, where a translation is given by a first-order definable operator which preserves inclusions among all languages as well as non-inclusions among languages from the class. An example is $\Phi(L) = \{x : \exists y \in L\, [y \neq x]\}$ and the underlying family $\mathcal{L}$ contains $\emptyset$, every singleton $\{x\}$ and the full set D. It is easy to see that $\mathcal{L}$ is learnable. However the given translation of $\mathcal{L}$ is not learnable, as the learner would have to converge to $\Phi(D)$ after having seen finitely many data items and then the input language could be $\Phi(\{y\}) = D - \{y\}$ for some y which has not yet been observed in the input. General characterisations were given for the question of the following type: "For which classes are all translations learnable under a given criterion?" Besides the standard criteria from inductive inference, the paper also looked at query learning. Here a learner can ask a teacher questions, in a given query language; for example the learner can ask whether the language L_i is a superset of the language R to be learnt (if the query language allows superset queries). The learner asks finitely many queries and has then to conjecture the correct index of the input language. Angluin [3] started the investigation of the learnability of regular languages from queries. She showed that the class of all regular languages can be learnt using membership and equivalence queries in polynomial time where, when learning R, the time bound depends on $d_{dfa}(R)$ and the largest counter example observed. The following result of Jain, Martin and Stephan [12] links explanatory learning, query learning and robustness.

Theorem 5.2: *The following conditions are equivalent for an automatic family $\mathcal{L} = \{L_i : i \in I\}$.*
(a) *For every $i \in I$ there is a bound b such that, for all $j \in I$ with $L_j \subset L_i$, there is a $k \in I$ with $k \leq_{ll} b$, $L_j \subseteq L_k$ and $L_i \not\subseteq L_k$.*
(b) *Every translation of $\mathcal{L}$ is explanatorily learnable.*
(c) *Every translation of $\mathcal{L}$ can be learnt using superset queries and membership queries.*
(d) *$\mathcal{L}$ can be learnt using superset queries.*

Note that in condition (d) it does not matter whether one learns only $\mathcal{L}$ using superset queries or every translation of $\mathcal{L}$ using superset queries. The reason for this is that translations do not change the inclusion structure of the automatic family. This result shows that there are also connections to robust notions of query learning.

Jain, Luo, Semukhin and Stephan [10] investigated the question on what can be said on the learnability of uncountable families. For this they use ω-automatic families, where the indices of the sets are ω-words and where a nondeterministic Büchi-automaton checks whether a finite word x belongs to the language defined by an ω-word. Also here the Büchi automaton is fed with the convolution of the input word and the ω-word serving as the index of the language. As there are uncountably many indices, it is no longer possible for the learner to come up with the correct index after finite time; therefore the model is adjusted to a verification game. So the learner reads in parallel a text consisting of all the words in the language and an ω-index; the output is a sequence of Büchi automata which converges to one fixed automaton. Then this automaton has to accept the given ω-index iff it is an index for the language observed. Also in this setting, learnability is equivalent to Angluin's tell-tale condition [1]. However it is necessary for this result that the learner has the right to choose the indexing; otherwise the criterion is more restrictive. Also further other criteria are transferred to this model and it is shown that one can abstain from adjusting the indexing if one requires vacillatory learning, where the learner in the limit oscillates between finitely many Büchi automata, and each of them accepts the given ω-index iff it is an index for the language to be learnt.

References

1. Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control* 45:117–135, 1980.
2. Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
3. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.
4. Achim Blumensath. *Automatic structures.* Diploma thesis, RWTH Aachen, 1999.
5. Achim Blumensath and Erich Grädel. Automatic structures. *15th Annual IEEE Symposium on Logic in Computer Science* (LICS), pages 51–62, IEEE Computer Society, 2000.
6. Mark Fulk. Robust separations in inductive inference. *Proceedings of the 31st Annual Symposium on Foundations of Computer Science* (FOCS), pages 405–410, St. Louis, Missouri, 1990.
7. E. Mark Gold. Language identification in the limit. *Information and Control* 10:447–474, 1967.
8. Bernard R. Hodgson. *Théories décidables par automate fini.* Ph.D. thesis, University of Montréal, 1976.
9. Bernard R. Hodgson. Décidabilité par automate fini. *Annales des sciences mathématiques du Québec*, 7(1):39–57, 1983.
10. Sanjay Jain, Qinglong Luo, Pavel Semukhin and Frank Stephan. Uncountable automatic classes and learning. *Algorithmic Learning Theory*, Twentieth International Conference, ALT 2009, Porto, Portugal, October 3-5, 2009. Proceedings. Springer LNAI 5809:293–307, 2009. Technical Report TRB1/09, School of Computing, National University of Singapore, 2009.
11. Sanjay Jain, Qinglong Luo and Frank Stephan. Learnability of automatic classes. *Language and Automata Theory and Applications*, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings. Springer LNAI 6031:321-332, 2010.

12. Sanjay Jain, Eric Martin and Frank Stephan. *Robust learning of automatic classes of languages.* Manuscript, 2010.
13. Sanjay Jain, Yuh Shin Ong and Frank Stephan. Regular patterns and regular languages. *Information Processing Letters*, 110:1114–1119, 2010.
14. Sanjay Jain, Daniel N. Osherson, James S. Royer and Arun Sharma. *Systems That Learn.* MIT Press, 2nd Edition, 1999.
15. Sanjay Jain and Frank Stephan. *A tour of robust learning. Computability and Models. Perspectives East and West.* Edited by S. Barry Cooper and Sergei S. Goncharov. Kluwer Academic / Plenum Publishers, University Series in Mathematics, pages 215–247, 2003.
16. Bakhadyr Khoussainov and Mia Minnes. Three lectures on automatic structures. *Proceedings of Logic Colloquium 2007. Lecture Notes in Logic*, 35:132–176, 2010.
17. Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. *Logical and Computational Complexity*, (International Workshop LCC 1994). Springer LNCS 960:367–392, 1995.
18. Steffen Lange and Rolf Wiehagen. Polynomial time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.
19. Steffen Lange and Thomas Zeugmann. Language learning in dependence on the space of hypotheses. *Proceedings of the Sixth Annual Conference on Computational Learning Theory* (COLT), pages 127–136. ACM Press, 1993.
20. Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Third Edition, Springer, 2008.
21. Yuh Shin Ong. *Automatic Structures and Learning.* Final Year Project, School of Computing, National University of Singapore, 2010.
22. Daniel Osherson, Michael Stob and Scott Weinstein. *Systems That Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists.* Bradford — The MIT Press, Cambridge, Massachusetts, 1986.
23. Daniel Reidenbach. *The Ambiguity of Morphisms in Free Monoids and its Impact on Algorithmic Properties of Pattern Languages.* Logos Verlag, Berlin, 2006.
24. Daniel Reidenbach. A non-learnable class of E-pattern languages. *Theoretical Computer Science*, 350:91–102, 2006.
25. Sasha Rubin. *Automatic Structures.* Ph.D. Thesis, University of Auckland, 2004.
26. Sasha Rubin. Automata presenting structures: a survey of the finite string case. *The Bulletin of Symbolic Logic*, 14:169–209, 2008.
27. Takeshi Shinohara. Polynomial time inference of extended regular pattern languages. *RIMS Symposia on Software Science and Engineering, Kyoto, Japan, Proceedings.* Springer LNCS 147:115–127, 1982.
28. Wai Yean Tan. *Automatic Structures.* Honours Year Thesis, Department of Mathematics, National University of Singapore, 2008.
29. Thomas Zeugmann. On Bārzdiņš' Conjecture. *Proceedings of the International Workshop on Analogical and Inductive Inference* (AII'86), Springer LNCS 265:220–227, 1986.