

PATTERN AVOIDING PERMUTATIONS ARE CONTEXT-SENSITIVE

MURRAY ELDER¹

ABSTRACT. We establish a bijection from the set of all permutations (of a given length) that avoid a pattern q and a context-sensitive language.

1. INTRODUCTION

Gessel conjectured that the sequence of q -avoiding permutations of each length is P-recursive [6]. Some doubt that Gessel's conjecture is true, believing that for some q the set of such permutations could be arbitrarily bad in some sense. The purpose of this article is to show that from the formal language perspective, the set is reasonable for any q . Specifically, we describe a context-sensitive language over a finite alphabet of symbols that bijectively encodes the set of permutations (of a given length) which avoid a fixed permutation q .

The encoding used is a variant of the “insertion encoding” introduced by Albert and Ruskuc [3]. In their work, Albert and Ruskuc are able to prove that for certain q the set of q -avoiding permutations is described by an unambiguous context-free language. In particular, they recover the fact that length three permutations are enumerated by the Catalan numbers, first observed by Knuth. Chomsky and Schützenberger [5] proved that unambiguous context-free languages have algebraic generating functions, and Bousquet-Mélou [4] proved that the set of 1234-avoiding permutations cannot have an algebraic generating function, thus we could not expect to describe 1234-avoiders by an unambiguous context-free language. The “next best thing” might be ambiguous context-free (unlikely) or indexed (maybe).

Implicit in the motivation of this result of this article is the hope that there might be some P-recursive analogue to Chomsky-Schützenberger's theorem that unambiguous context-free languages are algebraic, that is, that from the context-sensitive grammar one could derive a generating function for the number of q -avoiding permutations of each length that was of some desirable form. Currently we have no general procedures that can enumerate ambiguous context-free, indexed or context-sensitive languages.

Date: June 7, 2005.

2000 Mathematics Subject Classification. 05A05, 68Q45, 03D10.

Key words and phrases. Formal language, pattern-avoiding permutation, indexed language, context-sensitive language.

¹ Supported by EPSRC grant GR/S53503/01 .

The article is organised as follows. In Section 2 we define the encoding. In Section 3 we define context-sensitive and indexed grammars, and their machine counterparts. In Section 4 we prove the main theorem, that the encoding is a context-sensitive language.

2. THE ENCODING

We begin with a description of the “insertion encoding” [3]. The idea of this encoding is to build up a permutation by successively inserting the next highest entry in some open *slot*, starting from a single open slot, until all slots are filled. One can insert in one of four different ways: one can insert on the *left* of a slot, on the *right*, in the *middle* (creating two slots from one) or *filling* the slot.

As an example, the instructions “middle,right,left,fill,fill” or *mrlff* build the permutation 34215 as follows:

$$\begin{aligned}
 & \quad \quad \quad * \\
 \rightarrow & \quad * 1 * \\
 \rightarrow & \quad * 2 1 * \\
 \rightarrow & \quad 3 * 2 1 * \\
 \rightarrow & \quad 3 4 2 1 * \\
 \rightarrow & \quad 3 4 2 1 5
 \end{aligned}$$

In this example, not knowing any better, we performed each insertion on the left-most open slot. If we want to use any other slot, we precede the instruction by some number of letter *t* for *translate*, such that you shift one slot to the right for each *t* preceding each instruction.

So for the above example, the instructions *mrtltff* build the permutation 52134 as follows:

$$\begin{aligned}
 & \quad \quad \quad * \\
 \rightarrow & \quad * 1 * \\
 \rightarrow & \quad * 2 1 * \\
 \rightarrow & \quad * 2 1 3 * \\
 \rightarrow & \quad * 2 1 3 4 \\
 \rightarrow & \quad 5 2 1 3 4
 \end{aligned}$$

The original insertion encoding [3] used integer subscripts on the *l, r, m, f* letters to indicate the choice of slot. We call the present version the *it-encoding* which stands for “insert and translate”.

Definition 1 (*it-encoding*). Define a codeword be a string of letters *l, r, m, f, t*, subject to the condition that the number of *t* letters immediately preceding any of the other letters $x \in \{l, r, m, f\}$ is at most *k* where *k* is the difference between the number of *m* and the number of *f* letters that appear to the left of *x*, and the codeword ends with an *f*. Then such a codeword represents a permutation *p* obtained by a sequence of insertions defined by the following procedure.

Define the *next entry* to be the number that is to be inserted next into the permutation and denote it by *i*, define the *next slot* to be number of

the slot (numbered from left to right) in p into which the insertion will take place, and the *next letter* to be the next letter of the codeword.

Set $p = *$, $i = 1$ and the next slot to be 1.

Repeat forever:

Read the next letter of the codeword.

- If the next letter is t , increment the next slot by 1.
- If the next letter is l then replace the next slot by $(i)*$. Increment i by 1 and reset the next slot back to 1.
- If the next letter is r then replace the next slot by $*(i)$. Increment i by 1 and reset the next slot back to 1.
- If the next letter is m then replace the next slot by $*(i)*$ (thus increasing the number of slots in p by 1). Increment i by 1 and reset the next slot back to 1.
- If the next letter is f then replace the next slot by (i) (thus decreasing the number of slots in p by 1). Increment i by 1 and reset the next slot back to 1.

Note that the length of the permutation created is equal to the number of non- t letters in the code-word. Thus to enumerate permutations in this encoding of a given length, we would make use of a two-variable generating function, with one variable for non- t letters and the other for the ts . Then we could set the second variable to 1.

3. CONTEXT-SENSITIVE LANGUAGES

Definition 2 (Linear-bounded automaton). A *linear-bounded automaton* is a nondeterministic Turing machine such that for every input $w \in \Sigma^*$, there is some accepting computation in which the tape contains at most $|w| + 1$ symbols.

Definition 3 (Context-sensitive). A *context-sensitive grammar* consists of three finite sets N, T, P where N is a set of *non-terminals* which are denoted by upper case letters, and includes a distinguished letter S called the *start symbol*, T is a set of *terminals* which are denoted by lower case letters, and P is a set of *productions* of the form $\alpha \rightarrow \beta$ where $\alpha \in (N \cup T)^* N (N \cup T)^*$ and $\beta \in (N \cup T)^*$. The *language* of a context-sensitive grammar is the set of all words in T^* that can be obtained by applying some sequence of productions starting from S until all letters are terminals.

For readers familiar with context-free languages, a context-sensitive grammar is like a context-free grammar where some productions or rules can only be applied in the correct “context”. Just as context-free languages have a machine-theoretic counterpart (pushdown automata), so do context-sensitive languages.

Lemma 4. *A language is generated by a context-sensitive grammar if and only if it is accepted by a linear-bounded automaton.*

Examples of context-sensitive languages are $\{a^n b^n c^n | n \geq 0\}$, $\{(ab^n)^n | n \geq 0\}$ and $\{1^p | p \text{ is prime}\}$ (which are not context-free). Context-sensitive languages are closed under union, concatenation, star, intersection and complement [12].

Indexed languages were introduced by Aho [1] in '68, and lie between the classes of context-free and context-sensitive languages.

Definition 5 (Indexed grammar). An *indexed grammar* consists of four finite sets N, T, F, P where N is a set of *non-terminals* with S the *start symbol*, T is a set of *terminals*, F is a set of *flags* or *index productions* of the form $f : A \rightarrow \alpha$ where $A \in N$ and $\alpha \in (N \cup T)^*$, and P is a set of *productions* of the form $A \rightarrow \beta$ where $A \in N$ and $\beta \in (NF^* \cup T)^*$. The *language* of an indexed grammar is the set of all words in T^* that can be obtained by applying some sequence of productions starting from S until all letters are terminals.

Aho proved that every indexed grammar can be effectively converted to a grammar of the following form [1].

Definition 6 (Normal form). An indexed grammar is in *normal form* if each index production is of the form $A \rightarrow B$ and each production is one of $A \rightarrow BC$, $A \rightarrow a$ or $A \rightarrow Bf$ where $A, B \in N$, $a \in T$ and $f \in F$.

Once again, this class of languages has a machine counterpart. Aho first described these machines in [2], but the (equivalent) definition we use here comes from Gilman and Shapiro [9].

Definition 7 (One-way nondeterministic nested stack automaton). Consider a machine consisting of three sets Σ, Q, Γ where Σ is the *input alphabet*, Q is the set of *states*, which includes a distinguished *start state* q_0 and some *accept states*, and Γ is the *stack alphabet*; a finite *input tape* (on which the input string is written from left to right); a *finite state control* (which at any instance is in one of the states of Q); and a *nested stack* which is a rooted-tree such that the root node is at the base of the tree and cannot be deleted, the leaves are numbered in the order in which they were added, and edges can be added and deleted according to four *stack operations*; a pointer which points to a node of the tree, called the *current node*. It follows from the stack operations allowed that the pointer always points to a node that lies on a path from the root to the most recently added leaf.

There are four stack operations:

- The pointer can move from the current node up to a child (if it is not a leaf)
- The pointer can move from the current node down to its parent (if it is not the root)
- A new edge can be added to the current node, directed towards a new node, which becomes the new current node.
- If the current node is a leaf (and not the root), the current node can be deleted, so that the new current node is its parent.

The transition function of the machine takes either an input letter or ϵ , a state of the finite state control, and a current node, and outputs a new state and a stack operation. If the input letter is from the tape, then the next input letter is the next letter to the right. (The machine is “one-way” which means that the input is read from left to right with no backtracking).

A string in Σ^* is *accepted* if there is a sequence of transitions from the start state to an accept state of the finite machine, such that the root is never deleted.

Theorem 8 (Aho). *A language is indexed if and only if it is accepted by some one-way nondeterministic nested stack automaton.*

The following lemma is due to Gilman [8].

Lemma 9 (Shrinking Lemma). *If L is indexed and m is a positive integer, then there is a constant $k = k(L, m) > 0$ such that each word $w \in L$ with $|w| \geq k$ can be written as $w = w_1 \dots w_r$ with $m < r \leq k$, $w_i \neq \epsilon$, and every choice of m of the factors is included in a proper sub-product that lies in L . That is, the chosen m factors occur in a product of t of the factors $w_{i_1} \dots w_{i_t} \in L$ (in order, that is, $1 \leq i_1 \leq \dots \leq i_t \leq r$) with $m \leq t < r$.*

Examples of indexed languages are $\{a^n b^n c^n | n \geq 0\}$, $\{ab^{i_1} ab^{i_2} \dots ab^{i_n} a | i_1 \leq \dots \leq i_n\}$ (this language has intermediate growth by Grigorchuk and Machi [10]). A non-example is $\{(ab^n)^n | n \geq 0\}$ by Lemma 9. Indexed languages are closed under union, concatenation and star, but not closed under intersection or complement [1].

4. RESULTS

In this section we describe a nested stack automaton for the set of all codewords and a linear-bounded automaton for all codewords that avoid a permutation q .

Define E to be the set of all legal codewords. We will first show that E is accepted by a one-way nondeterministic nested stack automaton. We also observe that E is not context-free.

Next, let $E(q)$ be the set of codewords representing permutations that *avoid* the pattern q . We show that $E(q)$ is accepted by a linear-bounded automaton. We ask if there is a nested stack automaton that accepts it.

Proposition 10. *E is indexed.*

Proof. We will describe a one-way nondeterministic nested stack automaton that accepts the set of all strings of l, r, m, f, t that are codewords. That is, we must check that the number of ts immediately preceding any of the other letters is no more than the difference between the number of ms and fs preceding it.

The nested stack in this case is simply a single stack, however the pointer to the current node can be pointing to any node from the root up of the stack. Such a machine is sometimes called a *stack automaton* in the literature (see

for example [7]). There are three main states, q_0, q_A, q_F , start, accept and fail respectively.

First we describe the transition function for the start state q_0 . Let x be any node of the stack, and let x_r be any node except the root.

- $\delta(q_0, l, x)$: move the pointer to the top of the stack (that is, go into a state q_{up} and perform single ϵ transitions moving the cursor up one step) and continue.
- $\delta(q_0, r, x)$: move the pointer to the top of the stack and continue.
- $\delta(q_0, m, x)$: move the pointer to the top of the stack, add a new edge to a new leaf, and continue (so the pointer points to the new leaf).
- $\delta(q_0, f, x_r)$: move the pointer to the top of the stack, delete the top node, (so the pointer reverts back one step) and continue.
- $\delta(q_0, f, \text{root})$: if all the input has been read, move to q_A , or if not, move to q_F , and stop.
- $\delta(q_0, t, x_r)$: move the pointer down the stack one step.
- $\delta(q_0, t, \text{root})$: move to q_F , and stop.

The machine reads input letters from left to right, and the height of the stack indicates the current sum of number of ms minus fs . When a t is read, the machine checks that this sum is not zero (the pointer points above the root). For each t read in succession, the pointer moves down the stack, so if the number of ts preceding another letter exceeds $(m - f)$ the input is rejected, otherwise, if the end of the input is reached, and the last letter is an f with an empty stack, accept. If an f is read on an empty stack before all the input has been read, then no slots remain so the string is not a codeword. \square

It follows that E is context-sensitive. It may be possible to modify this argument to show that $E(q)$ is indexed. The best we can do at present is to show that $E(q)$ is context-sensitive.

Proposition 11. *E is not context-free.*

Proof. By the Pumping lemma for context-free languages [11], if E is context-free, then there is a constant p so that any word in E of length at least p can be subdivided into five subwords $uvxz$ so that $|v|, |y| > 0, |vxy| \leq p$ and for all $i \in \mathbb{N}, uv^i xy^i z \in E$.

Consider the word $m^p t^p f^{p+1}$ which represents the permutation $(p+2)p(p+3)(p-1) \dots (2p)2(2p+1)1(p+1)$. If the subword v contains an m then since $|vxy| \leq p, y$ cannot contain any fs . Thus $uv^i xy^i z$ would fail the condition that the number of ms must be one less than the number of fs in a codeword for large i . Thus v has no ms .

If y has an f , then v cannot contain any ms since $|vxy| \leq p$, so again $uv^i xy^i z$ would fail the condition that the number of ms must be one less than the number of fs in a codeword for large i .

Thus y has no fs . Then vxy have only t letters, so $uv^i xy^i z$ has more ts before the first f than the number of slots, so is not a codeword. \square

If the permutation used in the previous lemma avoids q , then $E(q)$ cannot be context-free. For $|q| \leq 3$ versions of the insertion encoding are context-free, but a reasonable conjecture might be that for $|q| \geq 4$ no encoding using a finite alphabet for the set of q -avoiders is context-free. Two good test-cases are $q = 1234$ and 1324 ; the first is known not to have an algebraic generating function ([4]) and for the second, no expression for the generating function is known.

Next we prove that $E(q)$ is context-sensitive. The first step is to show that a linear-bounded automaton can check that a string of l, r, m, f, t is a codeword, which is true since indexed languages are context-sensitive, and E is indexed. We construct the linear-bounded automaton directly here to exhibit a bound on the time complexity.

Lemma 12. *Let $w \in \{l, r, m, f, t\}^*$ written on $|w|$ squares of tape and let $n \in \mathbb{N}$. Then one can determine whether w is a legal code-word (that is, a word in E) for some permutation of length n and return the original string w on the tape, using only the $|w|$ squares in use and at most $O(|w|^2)$ steps.*

Proof. To do this, we need to make sure that the right number of t s appear before each non- t letter. Scan the tape (from left to right) until you find an m . Move right until either you find another m , or an f . (if you find neither, reject). When you find an m, f pair (with only r, l, t letters in between), overwrite the m and f with m^* and f^* .

Now scan from m^* to f^* and mark a single t (that is, replace it by t^*) in front of every r, l and the final f^* , if there are t s in front of them.

Continue until you cannot find any more unmarked m, f pairs. If there remains an m , reject. If there remains an f which is not at the end of the tape, reject. (you should still have the $f^\#$ at the end of the tape). If there remain any unmarked t letters, reject.

If not, then the work has the correct number of t s in front of each non- t , and the input ends with an f , so represents a correct permutation. \square

If $x \in \{l, r, m, f\}$ occurs in a code-word w then the entry of the encoded permutation p corresponding to w inserted by x is denoted by p_x .

Lemma 13. *If $w = w_1 x w_2 y w_3$ is a codeword with $x, y \in \{l, r, m, f\}$ then $p_x < p_y$.*

Proof. Since entries are inserted in increasing order, and x occurs before y in the codeword, then the entry inserted by x is smaller than the entry that is inserted at a later time by y . \square

We will make use of the following lemma.

Lemma 14. *Let w be an it-encoded word written on $|w|$ squares of tape and let $x, y \in \{l, r, m, f\}$ be such that $w = w_1 x w_2 y w_3$. Then one can determine whether p_x occurs before or after p_y in the permutation corresponding to w , (that is, determine whether the sub-permutation consisting of p_x and p_y is*

order isomorphic to 12 or 21), and return the original string w on the tape, using only the $|w|$ squares the input is written on and at most $O(|w|^2)$ steps.

Proof. By the previous lemma, since x is to the left of y , $p_x < p_y$. So x inserts the “1” and y inserts the “2”.

The slot in which x is inserted is determined by “counting” the preceding ts , and we can do this by marking each preceding t by a $*$. Suppose there are i marked ts , which means x inserts into the $(i + 1)$ -th slot. If x is an l or f then there are i slots to the left of x . If x is an r or m then there are $(i + 1)$ slots to the left of x , so star x as well.

So the number of starred squares will represent the number of open slots to the left of x . This number can be changed if an m or f inserts in a slot to the left of x .

Move right towards y . For each m passed, if we determine that m inserts into a slot before the position of x , we need to up our count by $+1$, (that is, mark m by a $*$ to signify a shift of slots). To check this, compare the starred entries with the ts immediately before this m . At the machine level, this can be achieved by scanning to the first t to the left of this m and marking it by \dagger , then scan left to the first starred square, which you remark with $**$, then move right to the t to the left of t^\dagger , then left and so on until either you run out of starred squares or run out of ts preceding the m . If there are more starred entries, then this m indeed occurs before x , so the position of x needs to be adjusted (so just star m). If m has more, do nothing. Unmark the t^\dagger s and remark the $**$ squares with $*$.

If you encounter an f before you get to y , again if it is determined that this f fills a slot before the position of x then we remove one of the stars from the rightmost starred square. We can determine this by comparing starred squares with ts preceding this f by the same procedure used in the previous paragraph.

If at any stage there are no starred entries left, then x now lies to the left of any open slots, so p_x, p_y give a 12 sub-permutation.

So by the time you get to y , the position of x is preserved. Now compare the number of ts preceding y with the number of starred entries. If there are j ts preceding y , and i starred squares, then y inserts into the $(j + 1)$ -th slot, so if $i \geq j + 1$ then p_x, p_y give a 21 sub-permutation, and if $i < j + 1$ then p_x, p_y give a 12 sub-permutation. Note that the type of y (l, r, m, f) doesn't affect this step.

Each time we pass an m or f and when we reach y , the machine scans left and right (marking with $**$ and \dagger) at most $|w|$ times, and there are no more than $|w|$ m, f s between x and y , so this procedure takes $O(|w|^2)$ steps. \square

Theorem 15. *$E(q)$ is accepted by a linear-bounded automaton in polynomial time.*

Proof. Let $|q| = k$. We will describe the linear-bounded automaton that can decide whether to accept or reject a string of l, r, m, f, t letters written on the tape, using the tape alphabet $l, r, m, f, t, l^*, r^*, m^*, f^*, t^*$.

First, verify that the input is a legal code-word by enacting the procedure given in Lemma 12. Note that the last letter must be an f . This takes at most $O(|w|^2)$ steps.

Next, we systematically run through all ordered k -tuples of squares, such that each square selected has a non- t letter on it. Let $\bar{x} = (x_1, x_2 \dots, x_k)$ be such a tuple.

We want to test if \bar{x} is order isomorphic to q . We do this by performing pairwise comparisons using the procedure of Lemma 13. This involves k comparisons, but since q is a fixed permutation, this is a constant.

Each comparison takes $O(|w|^2)$ time and we must repeat this $\binom{|w|}{k} \in O(|w|^k)$ times, so the algorithm described here runs in polynomial time.

If a sub-permutation that is order isomorphic to q is found, reject, else accept. \square

REFERENCES

- [1] Alfred V. Aho. Indexed grammars—an extension of context-free grammars. *J. Assoc. Comput. Mach.*, 15:647–671, 1968.
- [2] Alfred V. Aho. Nested stack automata. *J. Assoc. Comput. Mach.*, 16:383–406, 1969.
- [3] M. Albert and N. Ruskuc. The insertion encoding. Preprint 2003.
- [4] M. Bousquet-Mélou. Unpublished. See Bona, *Combinatorics of permutations*, page 210.
- [5] N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In *Computer programming and formal systems*, pages 118–161. North-Holland, Amsterdam, 1963.
- [6] Ira M. Gessel. Symmetric functions and P-recursiveness. *J. Combin. Theory Ser. A*, 53(2):257–285, 1990.
- [7] R. Gilman. Formal languages and infinite groups.
- [8] Robert H. Gilman. A shrinking lemma for indexed languages. *Theoret. Comput. Sci.*, 163(1-2):277–281, 1996.
- [9] Robert H. Gilman and Michael Shapiro. On groups whose word problem is solved by a nested stack automaton, 1998.
- [10] R. I. Grigorchuk and A. Machì. An example of an indexed language of intermediate growth. *Theoret. Comput. Sci.*, 215(1-2):325–327, 1999.
- [11] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Co., Reading, Mass., 1979. Addison-Wesley Series in Computer Science.
- [12] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.

SCHOOL OF COMPUTER SCIENCE, UNIVERSITY OF ST ANDREWS, NORTH HAUGH,
ST ANDREWS, FIFE, KY16 9SS, SCOTLAND

E-mail address: murray@dcs.st-and.ac.uk