Department of Computer Science,
University of Otago

Te Whare Wānanga o Otāgo

Technical Report OUCS-2002-04

Regular closed sets of permutations

Authors:

M. H. Albert, M. D. Atkinson,
Computer Science, University of Otago

N. Ruskuc
School of Mathematics and Statistics, University of St Andrews

---

REGULAR CLOSED SETS OF PERMUTATIONS

M.H. Albert, M.D. Atkinson
Department of Computer Science
University of Otago, New Zealand
and
N. Ruškuc
School of Mathematics and Statistics
University of St Andrews, UK

June 26, 2002

Abstract

Machines whose main purpose is to permute and sort data are studied. The sets of permutations that can arise are analysed by means of finite automata and avoided pattern techniques. Conditions are given for these sets being enumerated by rational generating functions.

Keywords Regular sets, permutations, involvement

1 Introduction

From the earliest days of Computer Science abstract machines have been used to model computations and categorise them according to the different resources they require. In this paper we consider a new type of machine that is suited to modelling computations whose sole or main effect is to permute data. Unlike most classical machines these new machines have an

infinite input alphabet whose symbols form the data that is to be permuted. Despite this we shall show how the theory of finite automata can be deployed in their analysis.

A *permuting machine* is a non-deterministic machine with the following properties:

1. it transforms an input stream of distinct tokens into an output stream that is a permutation of the input stream,

2. it is oblivious to the values of the input stream tokens,

3. it has a hereditary property: if an input stream σ can be transformed into an output stream τ and σ' is a subsequence of σ whose symbols transform into the subsequence τ' of τ, then it is possible for σ' (if presented as an input to the machine in its own right) to be transformed into an output stream τ'.

Examples

1. A *riffle shuffler* divides the input stream into two segments and then interleaves them in any way to form the output stream.

2. A *stack* receives members of the input stream and outputs them under a last-in-first-out discipline.

3. A *transportation network* [1] is any finite directed graph with a node to represent the input stream and a node to represent the output stream. The other nodes can each hold one of the input objects and the objects are moved around the graph until they emerge at the output node.

The oblivious property of permuting machines allows us to name the input tokens $1, 2, \ldots, n$ (in that order) in which case the output will be some permutation of $1, 2, \ldots, n$. In this way we can consider a permuting machine to be a generator of permutations (usually, because of the non-determinism, generating many of each length). There is another point of view which is sometimes more useful where we consider the input stream to be some permutation of $1, 2, \ldots, n$ and ask whether the machine is capable of sorting the tokens (so that they appear in the output stream in the order $1, 2, \ldots, n$).

2

These two viewpoints are equivalent since a machine can generate a partic-
ular permutation σ if and only if it can sort the permutation σ^{-1}.

However, it is the hereditary property which allows non-trivial properties of
permuting machines to be found because of a connection with the combi-
natorial theory of involvement and closed sets of permutations. Formally, a
permutation π is said to be *involved* in another permutation σ (denoted as
$\pi \preceq \sigma$) if π is order isomorphic to a subsequence of σ. For example 231 is
involved in 31542 because of the subsequence 352 (or the subsequence 342).
We also say that σ *avoids* π if π is not involved in σ.

Permutation involvement has been an active area of combinatorics for over
10 years although it surfaced long before that in data structuring questions
on stacks, queues and deques (see [7, 9, 10]). Involvement is a partial order
on the set of all permutations and is conveniently studied by means of order
ideals called closed sets. A *closed* set $\mathcal{X}$ of permutations is one with the
property that $\sigma \in \mathcal{X}$ and $\pi \preceq \sigma$ imply $\pi \in \mathcal{X}$. The connection between
permuting machines and closed sets is via the following result which follows
from the definitions.

Proposition 1 *The set of permutations that a permuting machine can gen-
erate, and the set that it can sort, are both closed.*

In classical automata theory machines are associated with the languages they
recognise. The above proposition suggests that the appropriate associated
language of a permuting machine is the closed set of permutations that it
can generate. We will study various permuting machines and their associated
closed sets, and will show the utility of the permuting machine paradigm as
a tool for advancing the theory of permutation involvement. Before giving
further details of our results we recall some key concepts about permutation
involvement.

A closed set $\mathcal{X}$ is, by definition, closed "downwards". But that is equivalent
to its complement $\mathcal{X}^C$ being closed "upwards" ($\sigma \in \mathcal{X}^C$ and $\sigma \preceq \tau$ imply
$\tau \in \mathcal{X}^C$). Obviously, $\mathcal{X}^C$ is determined by its set of minimal permutations
which we denote by $B(\mathcal{X})$ and call the *basis* of $\mathcal{X}$. Clearly

$$\mathcal{X} = \{\sigma \mid \sigma \notin \mathcal{X}^C\} = \{\sigma \mid \pi \npreceq \sigma \text{ for all } \pi \in B(\mathcal{X})\}$$

is determined by its basis. By definition, $B(\mathcal{X})$ is an antichain in the in-
volvement order and conversely every antichain has the form $B(\mathcal{X})$ for some

3

closed set $\mathcal{X}$. The bases of the closed sets of permutations generated by the machines in examples 1 and 2 above are $\{321, 2413, 2143\}$ and $\{312\}$ respectively. The closed sets that arise in practice are generally infinite so it is clearly significant to know when a finite description is available by means of the basis. Indeed, many combinatorial enumeration investigations begin from some particular finite basis and study properties of the closed set that it defines ([4, 11]). We let $\mathcal{A}(B)$ denote the closed set whose basis is the antichain B; in other words

$$\mathcal{A}(B) = \{\sigma \mid \beta \not\preceq \sigma \text{ for all } \beta \in B\}$$

Given a closed set $\mathcal{X}$ (or a permuting machine that defines it) we would like to be able to solve

- The decision problem: given a permutation σ decide whether $\sigma \in \mathcal{X}$ (in linear time if possible),

- The enumeration problem: determine, for each length n, the number of permutations in $\mathcal{X}$,

- The basis problem: find the basis of $\mathcal{X}$, or at least determine whether the basis is finite or infinite.

In this paper we shall show how to exploit the classical theory of finite automata to make progress on these problems. To do this we have to overcome the difficulty that this theory deals with strings over a finite alphabet, whereas the strings of $\mathcal{X}$ are written in the infinite alphabet $1, 2, \ldots$. Therefore we shall look for encodings of the permutations in $\mathcal{X}$ as strings over a finite alphabet (normally $[k] = \{1, 2, \ldots, k\}$) and hope to prove that the language of such encodings is regular (or to find conditions under which this is so). Once we have proved the regularity of such a language we can appeal to two well-known facts: that regular languages have linear time recognisers, and that the generating function (the formal power series whose coefficients give the number of sequences of each length) is a rational function.

Of course this approach cannot be expected to succeed in all cases if for no other reason than that closed sets do not always have rational generating functions. Nevertheless, in Sections 2 and 3, we shall give two wide classes of closed sets (and permuting machines) which show that the approach can have

4

significant successes. In particular we produce infinite families of closed sets all of whose finitely based closed subsets have rational generating functions. Our results therefore link to the many recent papers where particular closed sets have been enumerated (for example, [3, 5, 8]). In the final section we indicate how we hope our approach may be extended.

We conclude this section by recalling some basic facts about transducers.

For our purposes a transducer is essentially a (non-deterministic) finite automaton with output symbols (from an alphabet Γ) as well as input symbols (from an alphabet Δ). We allow ϵ inputs as well as ϵ outputs. A transducer defines a relation between Δ^* and Γ^* in a natural way. That is to say, for every path in the transducer from the starting state to one of the final states, let the sequence of input labels be α and the sequence of output labels β (all ϵ's being omitted of course); then (α, β) is a related pair.

In any transducer we can interchange the input and output symbols on each transition to obtain another transducer. Therefore

Lemma 1 *If $\mathcal{R}$ is a transducer relation so also is the transpose relation $\mathcal{R}^t$.*

Let $\mathcal{L} \subseteq \Delta^*$ and define

$$\mathcal{L}\mathcal{R} = \{\beta \in \Gamma^* \mid \text{ there exists } \alpha \in \mathcal{L} \text{ with } (\alpha, \beta) \in \mathcal{R}\}$$

The main result we need from the theory of transducers appears as exercise 11.9 in [6]. For completeness, and to establish notation, we include the proof.

Proposition 2 *If $\mathcal{R}$ is a transducer relation and $\mathcal{L}$ is a regular subset of Δ^* then $\mathcal{L}\mathcal{R}$ is regular.*

Proof: Let P be the set of states of the transducer, μ the transition function (mapping $P \times (\Delta \cup \{\epsilon\})$ into subsets of $P \times (\Gamma \cup \{\epsilon\})$), p_0 the initial state, and E the set of final states.

Let M be a finite automaton recognising $\mathcal{L}$. Suppose that M has set of states Q, transition function δ, initial state q_0, and set of final states F. Extend the definition of δ so that $(q, \epsilon) \mapsto q$ is a valid transition for all $q \in Q$.

Now define an automaton N as follows. The set of states is $P \times Q$, the initial state is (p_0, q_0), and the set of final states is $E \times F$. The transitions are

defined as follows. If there are transitions

$$p_1 \xrightarrow{d,g} p_2$$

and

$$q_1 \xrightarrow{d} q_2$$

(where $p_1, p_2 \in P$, $q_1, q_2 \in Q$, $d \in \Delta \cup \{\epsilon\}$ and $g \in \Gamma \cup \{\epsilon\}$) then N has a transition

$$(p_1, q_1) \xrightarrow{g} (p_2, q_2)$$

We prove that the new automaton recognises the set $\mathcal{LR}$. Let β be any string in $\mathcal{LR}$. By definition of $\mathcal{LR}$ we may choose a string $\alpha \in \mathcal{L}$ with $(\alpha, \beta) \in \mathcal{R}$. Then we have transducer transitions

$$p_0 \xrightarrow{a_1,b_1} p_1 \xrightarrow{a_2,b_2} \cdots \xrightarrow{a_n,b_n} p_n$$

with $p_n \in E$ witnessing that $(\alpha, \beta) \in R$. Then we have $\alpha = a_1 \ldots a_n$, $\beta = b_1 \ldots b_n$ (where, possibly, ϵ symbols may occur). We also have transitions of M

$$q_0 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$$

with $q_n \in F$ witnessing that $\alpha \in \mathcal{L}$. Then, by definition, we have transitions

$$(p_{i-1}, q_{i-1}) \xrightarrow{b_i} (p_i, q_i)$$

in N demonstrating that β is accepted by N.

We reverse this argument to get the converse. Suppose that $\beta \in \Gamma^*$ is accepted by N via a sequence of transitions

$$(p_{i-1}, q_{i-1}) \xrightarrow{b_i} (p_i, q_i)$$

where $\beta = b_1 \ldots b_n$ with each $b_i \in \Gamma \cup \{\epsilon\}$. By definition of N there exist $a_1, \ldots, a_n \in \Delta \cup \{\epsilon\}$ and state transitions

$$p_{i-1} \xrightarrow{a_i,b_i} p_i$$

of the transducer, and transitions

$$q_{i-1} \xrightarrow{a_i} q_i$$

of M. This proves that $\alpha = a_1 \ldots a_n \in \mathcal{L}$ and $(\alpha, \beta) \in \mathcal{R}$ as required. $\blacksquare$

2 Bounded classes

In this section we consider permuting machines as 'black boxes' into which input tokens are inserted and from which they eventually emerge as output tokens. So, at any point of a computation there may be some tokens which are 'inside' the machine (in the machine's memory) awaiting output. The chief hypothesis of this section is that, for some constant k, the machine can contain no more than k tokens at a time (so if it is full to capacity it must output a token before further input is possible). Such machines are said to be k-bounded.

If we consider a k-bounded machine as a generator of permutations then no permutation of length $k+1$ that begins with $k+1$ can be generated from the input $1, 2, \ldots, k+1$. Thus the closed sets associated with k-bounded machines are subsets of the closed set Ω_k whose basis consists of the $k!$ permutations $k+1, a_1, \ldots, a_k$ where $a_1, \ldots, a_k$ ranges over all permutations of $1, 2, \ldots, k$.

We shall see shortly that permutations in Ω_k may be encoded as words in a k-letter alphabet. Anticipating this, we define a subset of Ω_k to be regular if its encoded form is a regular set. We shall show that a closed subset $\mathcal{X}$ of Ω_k is regular if and only if its basis is regular. The proof of this result is, in principle, constructive in the sense that a recognising finite automaton for $\mathcal{X}$ can be built from one that recognises its basis and vice versa. In the course of proving this result we shall prove that it is decidable whether a regular subset of Ω_k is a closed subset.

Let $\pi = \pi_1 \pi_2 \ldots \pi_n$ be a permutation of length n. Its *rank* encoding is the sequence

$$E(\pi) = p_1 p_2 \ldots p_n$$

where

$$p_i = |\{j \mid j \geq i, \ \pi_j \leq \pi_i\}|$$

is the rank of π_i among $\{\pi_i, \pi_{i+1}, \ldots, \pi_n\}$.

Obviously, $\pi \in \Omega_k$ if and only if $\pi_1 \pi_2 \ldots \pi_n$ has no subsequence of length $k+1$ whose first element is the largest in the subsequence and this is precisely the condition that $p_i \leq k$ for all i. Thus every subset of Ω_k encodes as a subset of $[k]^*$.

Proposition 3 Ω_k *is regular.*

Proof: It is easy to see that a word $p = p_1 p_2 \ldots p_n$ is the encoding of some permutation if and only if

$$p_{n+1-i} \leq i \text{ for all } i \tag{1}$$

(and, if this condition holds, the permutation can readily be calculated). In fact, for $p \in [k]^*$ the above inequalities may fail to hold only for $i = 1, 2, \ldots, k-1$. Let F be the set of all words of length at most $k-1$ for which (1) does not hold. We now have

$$E(\Omega_k) = [k]^* \setminus [k]^* F,$$

which is a regular set. ∎

Example 1 Consider the closed subset $\mathcal{X}$ of Ω_2 whose basis is $312, 321, 231$. The first two basis elements ensure that, indeed, $\mathcal{X} \subseteq \Omega_2$ so the permutations of $\mathcal{X}$ encode as words in the alphabet $\{1, 2\}$ and end with a 1. It is readily checked that the third basis element restricts these words by prohibiting consecutive occurrences of the symbol 2.

The set of words that do contain consecutive 2s is described by the regular expression $[2]^* 22 [2]^*$ and so is regular. However $E(\mathcal{X})$ is the complement of this regular set within the regular set $E(\Omega_2)$ and so is also regular. Thus $\mathcal{X}$ is a regular closed set. The generating function of $E(\mathcal{X})$ is well-known to be

$$\frac{1}{1 - x - x^2}$$

and, since E is one-to-one, this is also the generating function of $\mathcal{X}$.

This easy example serves to illustrate that the condition of avoiding a permutation translates into restrictions on encodings although they are generally much more complicated than the ones above. The argument that proves regularity is a very special case of more general arguments to come.

Transportation networks are another source of examples. Theorem 1 of [1] proves that the closed sets associated with these are all regular. That paper also contains an example to show that regular closed sets need not be finitely based.

We also note that not every closed subset of Ω_k is regular. Indeed, as shown in [2], there are uncountably many closed subsets in Ω_k, if $k \geq 3$; but there are only countably many regular languages over $[k]$.

8

2.1 A transducer to delete a letter from a word

Let $\pi = \pi_1 \pi_2 \ldots \pi_n$ be a permutation in Ω_k and let $p = p_1 p_2 \ldots p_n$ be its rank encoded form. Let π' be the permutation obtained from π by deleting π_i (and relabelling appropriately) and let $p' = p'_1 \ldots p'_{i-1} p'_{i+1} \ldots p'_n$ be its encoded form. We put

$$\partial_i p = p'$$

call this the ith derivative of p. The process of passing from p to p' is called *deleting a letter* from p. We shall show how this may be done without "looking at" π.

Example 2 Let $p = 2331211$ representing the permutation $\pi = 2451637$. Then removing the 6th element of π results in the permutation $\pi' = 234156$ whose encoding is $p' = 222111$.

Suppose we have to delete the ith letter from p. We compute p' by scanning p from the right. For the positions to the right of p_i, each p_j represents the rank of some element of π among its successors, so these ranks will be unchanged by the deletion. Therefore until we reach p_i itself (which we delete) nothing changes. But for $j < i$ we need to know whether or not $\pi_j > \pi_i$ (so that we can tell whether or not to reduce p_j by 1). To do this we keep track of a variable r_j defined as the rank of π_i in the set $\{\pi_{j+1}, \ldots, \pi_n\}$ (the number of symbols in this set that are less than or equal to π_i). Clearly

$$\pi_j > \pi_i \text{ if and only if } p_j > r_j$$

Provided we have r_j we can decide whether we should reduce p_j. But, as the pointer j moves to the left, we can easily update r_j. Clearly, if $\pi_j > \pi_i$ then $r_{j-1} = r_j$; and if $\pi_j < \pi_i$ then $r_{j-1} = r_j + 1$. We therefore get Algorithm 1.

Two easy observations make this into a finite state algorithm. The first is the natural programming trick to use a single variable r in place of r_j. The second looks odd as a programming trick but is nevertheless essential. When $r_j \geq k$ the first alternative of the **if** is not followed nor is it followed thereafter; so we 'freeze' r_j to the value k once it reaches k. The result is Algorithm 2.

It is now easy to define a transducer for the relation

$$\mathcal{D} = \{(p, p') \mid p' \text{ is obtained by deleting one letter from } p\}$$

Algorithm 1 First form of the deletion algorithm

for $j := n$ **downto** $i + 1$ **do**
  $p'_j := p_j$
end for
$r_{i-1} := p_i$
for $j := i - 1$ **downto** 1 **do**
  if $p_j > r_j$ **then**
    $p'_j := p_j - 1; r_{j-1} := r_j$
  else
    $p'_j := p_j; r_{j-1} := r_j + 1$
  end if
end for

Algorithm 2 Second form of the deletion algorithm

for $j := n$ **downto** $i + 1$ **do**
  $p'_j := p_j$
end for
$r := p_i$
for $j := i - 1$ **downto** 1 **do**
  if $p_j > r$ **then**
    $p'_j := p_j - 1$
  else
    $p'_j := p_j$
    if $r < k$ **then**
      $r := r + 1$
    end if
  end if
end for

The transducer begins in a 'picking' state 0. Once it picks a letter to delete it passes through a sequence of states numbered according to the variable r in Algorithm 2. The transducer for the case $k = 3$ is shown in Figure 2.1.

Note 1 Strictly speaking, what we have constructed is a transducer for a relation where the words in question are read from right to left. To avoid notational clutter we make the convention that all finite automata and transducers read their input from right to left. Of course any conclusion that we reach of the form "$\mathcal{L}$ is a regular language" is independent of the direction of reading since $\mathcal{L}$ is regular if and only if its reverse is regular.
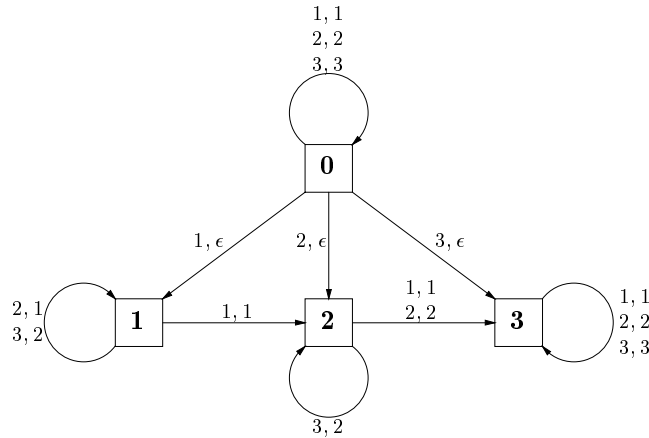

Figure 1: Deletion transducer with $k = 3$

Proposition 4 *Let $\mathcal{L} \subseteq E(\Omega_k)$ be regular. Then each of the following subsets is also regular, and finite automata recognising them are effectively computable from an automaton recognising $\mathcal{L}$.*

1. $\{\partial_i p \mid p \in \mathcal{L}, 1 \leq i \leq |p|\}$,

2. $\{p \in E(\Omega_k) \mid \partial_i p \in \mathcal{L}, \text{ for some } i\}$,

3. $\{p \in E(\Omega_k) \mid \partial_i p \in \mathcal{L}, \text{ for all } i\}$.

11

Proof: The first set is $\mathcal{LD}$ and the second is $\mathcal{LD}^t$ both of which are regular by Proposition 2. The third set is

$$
\begin{aligned}
\{p \mid \partial_i p \notin \mathcal{L} \text{ for some } i\}^C &= \{p \mid \partial_i p \in \mathcal{L}^C \text{ for some } i\}^C \\
&= (\mathcal{L}^C \mathcal{D}^t)^C
\end{aligned}
$$

Since regularity is preserved by complements the result follows again from Proposition 2. ∎

2.2 A transducer for deleting any number of letters

Again let $\pi = \pi_1 \pi_2 \ldots \pi_n$ be a permutation in Ω_k and let $p = p_1 p_2 \ldots p_n$ be its rank encoded form. We shall generalise the process described in the previous subsection so that it now deletes *any* number of letters (choosing which ones to delete non-deterministically again). From the resulting algorithm we shall be able to infer the existence of a transducer that defines the relation

$$
\mathcal{H} = \{(p, p') \mid p' \text{ arises by deleting any number of letters from } p\}
$$

In the generalisation a right to left scan takes place as before. But now, rather than setting up a single variable r when the deleted letter is met, we have to set up a different variable every time we come to a letter that is to be deleted.

So, suppose we come to a letter p_d that we intend to delete. Then we define a variable $r(d)$ (whose initial value will be p_d) which will play the same role as the variable r in the previous section. Just as before when we process a letter p_j (either to delete it or compute the value of p'_j) we shall have $r(d)$ equal to the rank of π_d in the set $\{\pi_{j+1}, \ldots, \pi_n\}$ (that is, $r(d)$ is the number of symbols in this set that are less than or equal to π_d).

Processing a particular p_j is then done as follows:

1. if p_j is to be deleted we set up a variable $r(j)$ as just mentioned and update any existing variables $r(d)$; this updating is explained below.

2. if p_j is not to be deleted we must use the variables $r(d)$ so far defined to compute the value of p'_j; and we must update these variables as necessary (see below).

Exactly as before, because of the meaning of each $r(d)$ we have $\pi_j > \pi_d$ if and only if $p_j > r(d)$. Therefore the number of d's for which this occurs is the decrement that has to be applied to p_j to obtain p'_j.

To do the updating of the variable $r(d)$ (so that it has the appropriate value when j is decreased by 1) we notice that any d for which $p_j > r(d)$ means that $r(d)$ is not changed; otherwise it must be increased by 1.

The behaviour of this algorithm when a symbol p_j is processed is governed by the values of the set of variables $r(d)$. In order to turn the algorithm into a transducer to recognise the relation $\mathcal{H}$ we have to demonstrate that only a fixed number of variables taking a fixed set of values is required.

First, we have the same remark as before: any $r(d)$ which reaches the value k can never affect whether p_j should be changed; so such $r(d)$'s can be discarded. The second remark is that the $r(d)$ are ranks of *different* elements within the same set $(\{\pi_{j+1}, \ldots, \pi_n\})$; therefore the values $r(d)$ are distinct and so we shall never have more than $k-1$ of them to store.

The state of the algorithm, as represented by the values of the $r(d)$, is therefore confined to one of a finite number of possibilities. A convenient way of representing the state is as a $(0,1)$ vector $(s_1, \ldots, s_{k-1})$. We set $s_t = 1$ if there is a variable $r(d)$ in the current 'live' set whose value is t; otherwise we set $s_t = 0$. This coding of state allows the automatic 'dropping' of a variable $r(d)$ once it reaches the value k.

Translating the way in which the $r(d)$ are handled, the updating of the variables s_t when a symbol $p_j = e$ is processed is easily seen to be:

$$(s_1, \ldots, s_{k-1}) := (s_1, \ldots, s_{e-1}, 1, s_e, \ldots, s_{k-2})$$

if p_j is to be deleted and

$$(s_1, \ldots, s_{k-1}) := (s_1, \ldots, s_{e-1}, 0, s_e, \ldots, s_{k-2})$$

otherwise. The value of p'_j in the latter case is $p_j - \sum_{f<e} s_f$.

We summarise this discussion in

Proposition 5 *There is a transducer that defines the relation*

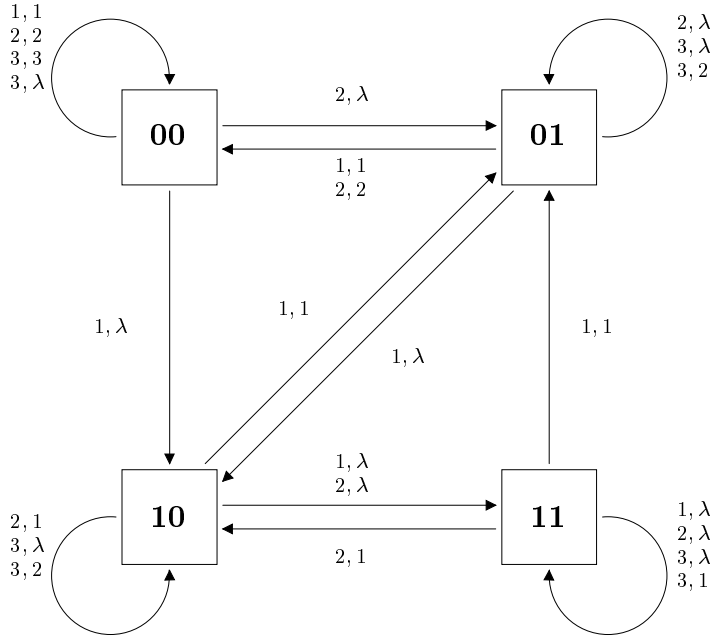$$\mathcal{H} = \{(p, p') \mid p' \text{ arises by deleting any number of letters from } p\}$$

13

Figure 2: Involvement transducer with $k = 3$

The state diagram for the transducer in the case $k = 3$ is shown in Figure 2. Clearly $\mathcal{H}^t$ is the relation of involvement on coded permutations and to reflect this we write $p' \leq p$ if p' can obtained from p by deleting any number of letters.

2.3 Regularity results

In this subsection we state and prove the main results on k-bounded classes.

Theorem 1 *There is an algorithm which decides whether or not a given regular set $\mathcal{L} \subseteq [k]^*$ can be expressed as $\mathcal{L} = E(\mathcal{X})$ for some closed set of permutations $\mathcal{X} \subseteq \Omega_k$.*

Proof: First note that a set $\mathcal{X}$ of permutations is closed if and only if for every $\pi = \pi_1 \pi_2 \ldots \pi_n \in \mathcal{X}$ and every $i = 1, \ldots, n$, we have $\pi \setminus \pi_i \in \mathcal{X}$. Thus, $\mathcal{L} = E(\mathcal{X})$ for some $\mathcal{X}$ if and only if $\{\partial_i p \mid p \in \mathcal{L}, 1 \leq i \leq |p|\} \subseteq \mathcal{L} \subseteq E(\Omega_k)$.

All the three above sets are regular (Proposition 3 and Proposition 4), and the automata accepting them are known, and hence we can decide whether these inclusions hold. ∎

Theorem 2 *A closed subset of Ω_k is regular if and only if its basis is regular.*

Proof: Let $\mathcal{X}$ be a closed set with basis $\mathcal{B}$. Suppose first that $\mathcal{X}$ is regular. By definition $\mathcal{B}$ is the set of all permutations $\pi = \pi_1 \ldots \pi_n$ such that $\pi \notin \mathcal{X}$ but $\pi \setminus \pi_i \in \mathcal{X}$ for all $i = 1, \ldots, n$. Thus

$$E(\mathcal{B}) = (E(\mathcal{X}))^C \cap \{p \mid \partial_i p \in E(\mathcal{X}), \text{ for all } i\},$$

which is a regular set by Proposition 4.

For the converse assume that $\mathcal{B}$ is regular. By Proposition 2 the set

$$E(\mathcal{B})\mathcal{H}^t = \{p \mid p' \leq p \text{ for some } p' \in E(\mathcal{B})\}$$

is regular and so its complement

$$(E(\mathcal{B})\mathcal{H}^t)^C = \{p \mid p' \not\leq p \text{ for all } p' \in E(\mathcal{B})\}$$

is also regular. Therefore $(E(\mathcal{B})\mathcal{H}^t)^C \cap E(\Omega_k)$ is regular as well; but this set is $E(\mathcal{X})$ itself. ∎

The regular set operations that we have used (intersection and complementation) are effectively computable in the sense that automata to recognise the resulting languages can be constructed. Therefore we have

Corollary 1 *There is an algorithm which, given an automaton accepting $E(\mathcal{X})$ for some regular closed set $\mathcal{X}$, computes an automaton accepting $E(B)$, where B is the basis of $\mathcal{X}$. The converse is also true.*

This, in turn has the following pleasing consequence:

Corollary 2 *It is decidable whether or not a given regular closed set is finitely based.*

Corollary 3 *The following are true for any closed set $\mathcal{X} \subseteq \Omega_k$ with a regular (in particular, finite) basis:*

15

(i) *the enumeration sequence for $\mathcal{X}$ satisfies a linear recurrence with constant coefficients;*

(ii) *membership in $\mathcal{X}$ can be checked in linear time.*

Proof: (i) $\mathcal{X}$ is in one-to-one length preserving correspondence with $E(\mathcal{X})$ which, being regular, has a rational generating function.

(ii) Both testing for membership in a regular language and the process of encoding permutations are linear. ∎

The first part of this corollary provides a partial (affirmative) answer to a conjecture of Gessel (that all finitely based closed sets have holonomic generating functions).

Theorem 2 allows us to give explicit examples of non-regular closed sets. Let $\mathcal{A}$ be any infinite antichain of permutations contained in Ω_k. An example of such an antichain with $k = 3$ is given in [2]. Let $\mathcal{A}_0 = \{\alpha_{n_1}, \alpha_{n_2}, \ldots\}$ be an infinite subset of $\mathcal{A}$ such that

1. $|\alpha_{n_i}| = n_i$,

2. $n_1 < n_2 < \ldots$ is not a finite union of arithmetic progressions.

Then $\mathcal{A}_0$ is a non-regular infinite antichain and, by Theorem 2, defines a closed set that is not regular.

3   Monotone segment sets

In this section we consider permuting machines with an unbounded memory. The memory is represented by a two-way infinite tape on which is stored an input sequence $1, 2, \ldots, n$, one token per tape square, and a reading head moves up and down the tape. We consider machines M_ϕ which operate under a fixed regime of forward and backward scans of the tape that is specified by a sequence $\phi = f_1 f_2 \ldots f_k$ of $+$ and $-$ signs.

The machine carries out k scans of the tape at the end of which all the input symbols will have been output. The ith scan is from left to right if $f_i = +$ and from right to left if $f_i = -$. During each scan the machine will either skip over a symbol or output it (sequentially onto a second tape say).

Such a computation can be described by a *computation word* $c_1 \dots c_n$ with $1 \le c_i \le k$; the term c_i gives the scan number on which symbol i was output.

Example 3 Let $\phi = (+, -, -)$ so that M_ϕ does one left to right scan and two scans right to left. Suppose that the input tape contains 123456789. Then, supposing M_ϕ is subject to no further constraints it might, in its first scan output $2, 4, 8$, in its second scan output $7, 3$, and in its final scan output $9, 6, 5, 1$. The result is the output permutation 248739651. Notice that there is another computation by this machine that produces the same output permutation (the first scan outputs $2, 4$, the second scan outputs $8, 7, 3$, and the third scan outputs $9, 6, 5, 1$). The computation words for these two computations are 312133213 and 312133223.

Clearly this machine can only output permutations which have a segmentation $\alpha\beta\gamma$ where α is increasing and β, γ are decreasing. However, we do not exclude the possibility that, due to further constraints on the operation of the machine, not all permutations of this form can occur.

In the general case the (closed) set of permutations output by M_ϕ is a subset of

$$\mathcal{W}_\phi = \{\sigma_1\sigma_2 \dots \sigma_k\}$$

where each σ_i is an increasing sequence of symbols if $f_i = +$ and a decreasing sequence otherwise. The main results of this section are that the closed subsets of $\mathcal{W}_\phi$ have linear time recognisers and rational generating functions.

Every computation word c gives rise to a permutation $D_\phi(c) \in W_\phi$. To be precise, if we regard c as a function

$$c : [n] \to [k]$$

then $D_\phi(c)$ is the permutation obtained by concatenating the sets $c^{-1}(1)$ through $c^{-1}(k)$, with the ith set in this concatenation arranged in increasing order if $f_i = +$, and in decreasing order if $f_i = -$. It is easily seen that $\mathcal{W}_\phi$ is the image of $[k]^*$ under the map D_ϕ.

We have observed already that D_ϕ is not one-to-one but clearly each $D_\phi^{-1}(\pi)$ is a finite set (that is, every permutation $\pi \in \mathcal{W}_\phi$ can be obtained in only finitely many ways). We shall find it convenient to call its members the *encodings* of π. This situation differs from that in the previous section in that now a permutation may have several encodings. Nevertheless we define subset $\mathcal{X}$ of $\mathcal{W}_\phi$ to be regular, if $D_\phi^{-1}(\mathcal{X})$ is a regular subset of $[k]^*$.

17

Lemma 2 *Suppose that $s, p \in [k]^*$ and s is a subword of p. Then $D_\phi(s) \preceq D_\phi(p)$. Also suppose that $\sigma \preceq \pi$ are elements of $\mathcal{W}_\phi$. Then for each encoding p of π there exists an encoding s of σ which is a subword of p*

Proof: The first part is immediate. For the remainder, take a subset of the positions in π with pattern σ. Then just take s to be the subword of p on the same positions. ∎

Theorem 3 *Every closed subset of $\mathcal{W}_\phi$ is regular.*

Proof: Let $\mathcal{X}$ be a closed subset of $\mathcal{W}_\phi$ and let B be its basis. By Theorem 2.9 of [2] B is finite. Let $\mathcal{B}$ be the set of all elements of $[k]^*$ which have a subword belonging to $D_\phi^{-1}(B)$. Since $D_\phi^{-1}(B)$ is finite, $\mathcal{B}$ is regular. Suppose that $\pi \in \mathcal{X}$. Then no encoding p of π can contain an element s of $D_\phi^{-1}(B)$ as a subword, for otherwise $D_\phi(s) \preceq \pi$. So $D_\phi^{-1}(\mathcal{X}) \subseteq \mathcal{B}^c$. On the other hand, if $p \in \mathcal{B}^c$, and $\pi = D_\phi(p)$, then $\pi \in \mathcal{X}$ – for if not there is some $\sigma \in B$ with $\sigma \preceq \pi$, and then some encoding s of σ which is a subword of p, a contradiction. So $D_\phi^{-1}(\mathcal{X}) = \mathcal{B}^c$, which is regular. ∎

Corollary 4 *There is a linear time recognition algorithm for any closed subset of $\mathcal{W}_\phi$.*

We cannot immediately deduce that every closed subset of $\mathcal{W}_\phi$ has a rational generating function since the correspondence between $\mathcal{W}_\phi$ and $[k]^*$ is not one-to-one. To get around this difficulty we define, for every $\sigma \in \mathcal{W}_\phi$, a distinguished encoding as follows. Let σ_1 be the longest monotone initial segment of σ consistent with the sign f_1. Having chosen σ_1 we choose the next monotone segment σ_2 (corresponding to f_2) also as long as possible, and we continue in this manner until all of σ has been segmented (necessarily with k or fewer segments). The corresponding encoding $c_1 \ldots c_k$, where $c_i = j$ if $i \in \sigma_j$, is called the *greedy* encoding of σ. (In the example above the first encoding was greedy, the second was not).

Lemma 3 *The greedy encoding of $\mathcal{W}_\phi$ is a regular set.*

Proof: Let p and $q = p + 1$ be any two consecutive positions of ϕ. In the greedy encoding $c_1 \ldots c_n$ of a permutation $\sigma \in \mathcal{W}_\phi$ let the positions where $c_h = p$ be $h = i_1, i_2, \ldots, i_a$ and those where $c_h = q$ be $h = j_1, j_2 \ldots, j_b$. The greedy condition implies one of the following:

18

$[f_p = +, f_q = +]$ Since σ has adjacent segments $i_1, i_2, \ldots, i_a$ and $j_1, \ldots, j_b$ we have $i_a > j_1$; that is, in c, the final p comes after the first q.

$[f_p = +, f_q = -]$ Here σ has adjacent segments $i_1, i_2, \ldots, i_a$ and $j_b, \ldots, j_2, j_1$, so $i_a > j_b$; that is, the final p comes after the final q.

$[f_p = -, f_q = +]$ Similarly, the first p comes before the first q.

$[f_p = -, f_q = -]$ The first p comes before the last q.

Every consecutive $p, p + 1$ gives a restriction on the form of a greedy encoding but these restrictions are all recognisable by a finite automaton thus completing the proof. ∎

Theorem 4 *Every closed subset of $\mathcal{W}_\phi$ has a rational generating function.*

Proof: Let $\mathcal{X}$ be any closed subset of $\mathcal{W}_\phi$. By Theorem 3 $D_\phi^{-1}(\mathcal{X})$ is regular and therefore $D_\phi^{-1}(\mathcal{X}) \cap \mathcal{G}$, where $\mathcal{G}$ is the set of greedy encodings of $\mathcal{W}_\phi$, is also regular. But this set is in one-to-one correspondence with $\mathcal{X}$. ∎

4   Final remarks

We have shown that closed sets are the natural objects to study in the analysis of permuting machines. We have also demonstrated that, when a suitable encoding of permutations is available, finite automata are a powerful tool in this study. Nevertheless many problems remain. In particular, one natural question is how one might extend the automata tools to use context-free encodings. Here one might hope to prove that certain closed sets have an algebraic generating function rather than a rational one. A natural candidate to investigate are the closed subsets of $A(312)$ (which is the language associated with a single stack) for here well-formed bracket sequences encode permutations in a natural way. We hope to report progress on such problems in a subsequent paper.

Another issue is that of practicability. The "effective" methods we have developed for contructing automata frequently lead to automata with very large numbers of states since, in particular, we often need to convert a non-deterministic automaton to its deterministic version. In some special cases

we have managed to contain this state explosion and have carried out these constructions, and this gives hope that more efficient methods may exist.

References

[1] M. D. Atkinson, M. J. Livesey, D. H. Tulley: Permutations generated by token passing in graphs, Theoretical Computer Science 178 (1997) 103–118.

[2] M. D. Atkinson, M. M. Murphy, and N. Ruškuc: Partially well-ordered closed sets of permutations, Order (to appear).

[3] E. Barcucci, A. Del Lungo, E. Pergola, R. Pinzani: Permutations avoiding an increasing number of length increasing forbidden subsequences, Discrete Mathematics and Theoretical Computer Science 4 (2000), 31–44.

[4] M. Bóna: Exact enumeration of 1342-avoiding permutations; A close link with labeled trees and planar maps, Journal of Combinatorial Theory, Series A, 80 (1997) 257–272.

[5] T. Chow, J. West: Forbidden subsequences and Chebyshev polynomials, Discrete Math. 204 (1999) 119–128.

[6] J. E. Hopcroft and J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass. (1979).

[7] D. E. Knuth: *Fundamental Algorithms, The Art of Computer Programming* Vol. 1 (Second Edition), Addison-Wesley, Reading, Mass. (1973).

[8] T. Mansour and A. Vainshtein: Restricted 132-avoiding permutations, Adv. Appl. Math. 26 (2001) 258–269.

[9] V. R. Pratt: Computing permutations with double-ended queues, parallel stacks and parallel queues, Proc. ACM Symp. Theory of Computing 5 (1973), 268–277.

[10] R. E. Tarjan: Sorting using networks of queues and stacks, Journal of the ACM 19 (1972), 341–346.

[11] J. West: Generating trees and the Catalan and Schröder numbers, Discrete Math. 146 (1995), 247-262.