Permutations generated by stacks and deques

M. H. Albert[*] M. D. Atkinson[*] S. A. Linton[†]

June 23, 2009

Abstract

Lower and upper bounds are given for the the number of permutations of length n generated by two stacks in series, two stacks in parallel, and a general deque.

Keywords: Deque, parallel stacks, serial stacks, permutation, enumeration, growth rate.

Mathematics Subject Classification: Primary 05A05, 05A16; Secondary 68P05

1 Introduction

In the exercises of section 2.2.1 of the first edition of volume 1 of *The Art of Computer Programming* [6] Donald Knuth raised some fascinating questions about the number of permutations that various data structures could generate. The most famous of these was about the number of permutations that could be generated by a stack which he solved completely. Another scenario that he completely solved concerned the number of permutations that could be generated by an output-restricted deque. In Exercise 2.2.1.13 he asked the same question for general deques and in the exercises of section 5.2.4 in volume 3 [7] he introduced the problem of generating permutations through serial compositions of stacks.

Knuth's work soon inspired two follow-up papers. Tarjan [12] considered networks of stacks and queues, particularly serial and parallel compositions. For queues his results were fairly complete but for stacks he ended by commenting on the difficulty of these problems. Pratt [9] studied the cases of two stacks in parallel and general deques. While all three authors realised that the permutations generated by the various configurations could be described by forbidding certain patterns to occur in the permutations it was Pratt who drew explicit attention to the subpermutation relation:

[*]Department of Computer Science, University of Otago
[†]School of Computer Science, University of St Andrews

> "…the subtask relation on permutations is even more interesting than the networks we were characterizing. This relation seems to be the only partial order on permutations that arises in a simple and natural way, yet it has received no attention to date."

Despite this remark the study of permutation classes defined by forbidden patterns did not resume until Simion and Schmidt revived it in [10]. Since then there have been many papers on the subject but there remain still some significant unanswered questions arising out of the work of Knuth, Tarjan, and Pratt. Some of these questions are the focus for this paper and we shall attack them using tools that have been developed for the study of permutation classes. Specifically we shall address the following three problems:

1. How many permutations of length n can be generated by two stacks connected in series?
2. How many permutations of length n can be generated by two stacks connected in parallel?
3. How many permutations of length n can be generated by a general deque?

While we cannot give exact answers to these problems we shall prove lower and upper bounds on "growth rates". Suppose that t_n is the number of permutations of length n generated by one of these three systems. We shall prove that $\lim_{n\to\infty} \sqrt[n]{t_n}$ exists and give lower and upper bounds on the value of this limit.

Our paper is organised as follows. In section 2 we shall define our terms and state the problems precisely. Section 3 explains how the upper bounds are obtained and section 4 explains how lower bounds are obtained.

2 Preliminaries

We give the basic terminology of pattern classes and permuting machines thereby defining the common background of all three enumeration problems.

The "subtask" relation that Pratt referred to is nowadays called the "subpermutation" relation and is defined as follows. Let π, σ be two permutations of length m, n and suppose that σ has a subsequence of length m that is isomorphic to π (its terms are ordered relatively the same as the terms of π). Then we say that π is a subpermutation of σ and write $\pi \preceq \sigma$. For example 231 is a subpermutation of 13542 because of the subsequence 352 (or 342).

A *pattern class* is a set of permutations closed under taking subpermutations. A pattern class P is said to respect direct sums if, whenever $\alpha, \beta \in P$, we have $\alpha \oplus \beta \in P$ ($\alpha \oplus \beta$ is the permutation which can be written as the juxtaposition of two sequences α', β' which are isomorphic to α, β respectively and for which every term in α' is less than every term in β').

A *permuting machine* is a device that accepts a stream σ of input and produces an output stream τ that is a rearrangement of its input. The permutational behaviour of a permuting machine M is represented by its set $A(M)$ of *allowable pairs*: those pairs (σ,τ) such that τ is a possible output if M is presented with σ as input. The machine is said to be *oblivious* if whenever

$$(x_1x_2\cdots x_n, y_1y_2\cdots y_n)\in A(M)$$

and ρ is a bijection with $x_i' = \rho(x_i)$ then (putting $y_i' = \rho(y_i)$) we have

$$(x_1'x_2'\cdots x_n', y_1'y_2'\cdots y_n')\in A(M)$$

It is said to have the *subsequence* property if whenever

$$(x_1x_2\cdots x_n, y_1y_2\cdots y_n)\in A(M)$$

and $x_{i_1}x_{i_2}\cdots x_{i_m}$ is a subsequence of $x_1x_2\cdots x_n$ whose terms appear as the subsequence $y_{j_1}y_{j_2}\cdots y_{j_m}$ in $y_1y_2\cdots y_n$ then

$$(x_{i_1}x_{i_2}\cdots x_{i_m}, y_{j_1}y_{j_2}\cdots y_{j_m})\in A(M)$$

The oblivious property tells us that the behaviour of M does not depend on the names of the symbols in the input stream so we might as well rename them as $1,2,\ldots,n$. Then the possible outputs of M are permutations of length n and we call them the permutations *generated* by M. The subsequence property tells us that any subpermutation of a permutation generated by M is also generated by M; in other words, the permutations generated by M form a pattern class.

In most cases of interest the pattern class associated with a machine will respect direct sums because, if α,β are permutations that a machine can generate we can feed the machine with input $1,2,\ldots,a,a+1,\ldots,a+b$ where $a=|\alpha|$ and $b=|\beta|$ and allow it to output α as it processes $1,2,\ldots a$ and then output a permutation isomorphic to β as it processes $a+1,\ldots,a+b$.

Now suppose that $t=(t_0,t_1,\ldots)$ is an infinite sequence. If the limit

$$g(t)=\lim_{n\to\infty}\sqrt[n]{t_n}$$

exists it is called the *growth rate* of t and t_n can be crudely approximated by $g(t)^n$. It is suspected that, whenever t is the enumeration sequence of some proper pattern class, then $g(t)$ exists. If the pattern class respects direct sums then, certainly, $g(t)$ exists and is finite (a consequence of an argument of Arratia [2] and the celebrated Marcus-Tardos theorem [8]).

The three systems studied in this paper can all be modelled by permuting machines. Figure 1 depicts two stacks combined in series. An input sequence is transferred, symbol by symbol, to an output stream. Each symbol enters the right stack (the push operation I), is eventually transferred to the left stack (the operation T pops it from the right stack and pushes it onto the left stack)

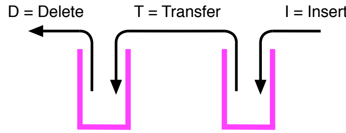

Figure 1: Two stacks in series

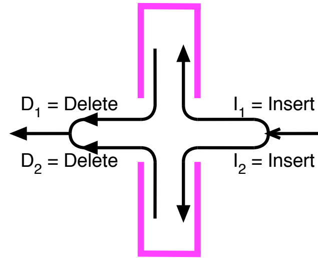

Figure 2: Two stacks in parallel

and is ultimately popped into the output stream (operation D). A computation of this machine is caused by a sequence of n I operations, n T operations, and n D operations. There is no restriction on the order in which the operations I, T, D are carried out other than the natural restriction that pop operations cannot be carried out on an empty stack. Pop operations always apply to the top symbol of a stack, while push operations place a symbol on top of a stack.

Figures 2 and 3 show the permuting machines that define the operation of two parallel stacks and a general deque. In each case there are two operations (I_1, I_2) that enable the next input symbol to enter the system and two operations (D_1, D_2) to transfer a symbol to the output stream. Again these operations operate on the ends of the structures shown.

In all three cases it is clear that the permuting machine is oblivious, has the subsequence property and respects direct sums. Therefore the sets of permutations generated by the machines form pattern classes and have well-defined growth rates.

3 Upper bounds

Each permutation generated by one of the three systems under consideration arises from a sequence of operations (I, T, D in the case of two stacks in series, I_1, I_2, D_1, D_2 in the case of two stacks in parallel and a general deque). Thus

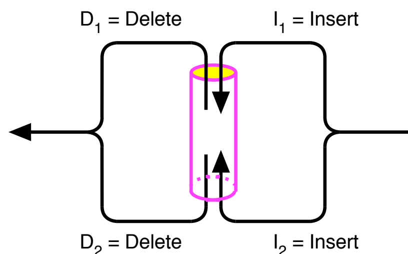

Figure 3: Deque

upper bounds on the numbers of permutations generated by the systems can be obtained by enumerating the number of possible operation sequences. However the number of operation sequences greatly exceeds the number of permutations because, frequently, the same permutation can be generated by many different operation sequences. In this section we show how, nevertheless, counting operation sequences can deliver non-trivial upper bounds on growth rates.

Before giving these details, however, it is interesting to recall Knuth's treatment of stacks and input-restricted deques. In the case of stacks every permutation is generated by a unique operation sequence so the count of operation sequences delivers the number of stack permutations exactly. For input-restricted deques (where we only have the operations I_1, I_2, D_1) this is no longer true. Knuth showed how this difficulty can be overcome by insisting that, when the deque is empty, its next operation must be I_1 rather than I_2 and (more significantly from our point of view) any operation sequence with a segment I_2D_1 should not be counted since the same permutation can be achieved by replacing the segment I_2D_1 with D_1I_2. Subject to these two restrictions operation sequences turn out to be equivalent to permutations and this enabled the enumeration to be calculated exactly.

For our more complicated systems we have not managed to find a complete set of operation sequences that is in one-to-one correspondence with the generated permutations and it is possible that there is no finite set of replacement rules that will define such a set. Nevertheless we have carried out a computer search for pairs of operation sequences that have the same effect on a system. For example, for two stacks in series, $ITDT$ has the same effect as $TITD$ and $ITTD$ has the same effect as $TDIT$; and for two stacks in parallel (and in a deque) the operations I_1 and D_2 commute.

We can describe our search for equivalent operation sequences in general terms that apply to any permuting machine M that is manipulated by operations that transform an input sequence into an output sequence. Suppose $\mathcal{M}$ is the set of operations in question. For two words λ, ρ over $\mathcal{M}$ we write $\lambda \rightarrow \rho$ if, for all words μ_1, μ_2 over $\mathcal{M}$, whenever the operation sequence $\mu_1\lambda\mu_2$ generates a

permutation π, the same permutation can be generated by $\mu_1\rho\mu_2$. For example, for two stacks in series, $TITD \rightarrow ITTD$. These $\lambda \rightarrow \rho$ relations define rewriting rules for words over $\mathcal{M}$. Following the usual procedure in rewriting systems we define a well-founded total order on the set of words over $\mathcal{M}$ that is respected by juxtaposition of words and arrange our rewriting rules $\lambda \rightarrow \rho$ so that $\lambda > \rho$. This ensures that every word over $\mathcal{M}$ can be rewritten into one which has no subword equal to the left-hand side of a rewriting rule.

Suppose we have found all the rewriting rules $\lambda \rightarrow \rho$ for $|\lambda| = |\rho| < n$. Consider the language defined by all words with no left-hand side λ as a subword. This is a regular language and we may construct the finite state automaton that recognises it. We now iterate over all the words W of length n that it accepts and compute their effect on a "generic" state of M (a generic state is one where any W does not cause exceptional behaviour to occur, such as removing an element from an empty stack). Then we examine pairs of words that have duplicate effects and test that even on non-generic states they also have duplicate effects. Such a pair then becomes a rewriting rule of length n.

Having generated all such rules $\lambda \rightarrow \rho$ that our computational resources allow we can be sure that each permutation is defined by an operation sequence that has no subword λ. In fact we know rather more – these operation sequences must have the property that they never attempt to remove an element from an empty stack or deque; but we do not exploit this information.

Instead we again exploit the fact that the set of words that have no subword λ is a regular language and construct a finite automaton that recognises such words. From the state equations of this automaton we can derive the generating function that counts the words in this language and thereby obtain the growth rate of the language. This will be an upper bound on the number of permutations.

Our results are summarised in tables 1, 2 and 3. In these tables we give the upper bound on the growth rates for increasing numbers of relations up to length 16. Naturally, the bounds improve the more relations we use and so the final line (length 16) of these tables gives our best results. We have included the results for lengths less than 16 because they give an indication about what results might be expected if faster and larger computers were used. For the same reason we give similar extra data in the tables in Section 4.

4 Lower bounds

4.1 General approach

To obtain lower bounds we build on the ideas that first appeared in [3] and were refined in [1]. We impose a bound k on the capacity of the system we are studying and estimate the number of permutations that can be generated using this restricted system. This is equivalent to adding pattern constraints of the

Table 1: Deque upper bounds

Length	Number of relations	Growth upper bound
8	51	8.4925
9	85	8.459
10	175	8.428
11	321	8.410
12	756	8.392
13	1480	8.380
14	3806	8.368
15	7734	8.361
16	21029	8.352

Table 2: Two parallel stacks

Length	Number of relations	Growth upper bound
8	33	8.4606
9	43	8.4474
10	109	8.4087
11	143	8.4031
12	466	8.379
13	615	8.376
14	2366	8.3597
15	3131	8.3578
16	13263	8.3461

Table 3: Two serial stacks

Length	Number of relations	Growth upper bound
8	23	14.201
9	35	14.048
10	71	13.826
11	106	13.747
12	215	13.623
13	368	13.552
14	737	13.477
15	1270	13.433
16	2825	13.374

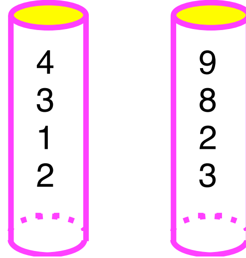

Figure 4: Equivalent dispositions in a deque

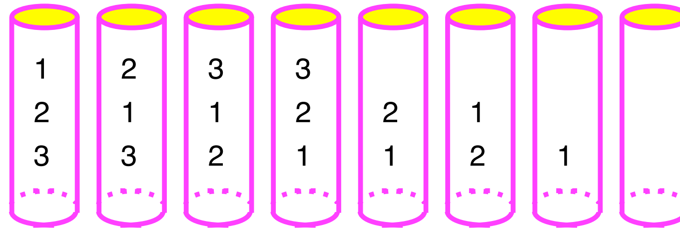

Figure 5: The states of a 3-bounded deque

form $[k+1, \alpha]$ for all permutations α of length k. Under this restriction we can describe the permutations generated by words over the alphabet $\{1, \ldots, k\}$ using the encoding of permutations that encodes each $\sigma = s_1 s_2 \cdots s_n$ by $e(\sigma) = e_1 e_2 \cdots e_n$ where e_i is the rank of s_i in the set $\{s_i, s_{i+1}, \ldots, s_n\}$. For example, 31524 encodes as 31311.

The operation of the system can be modelled by a finite automaton. A *disposition* of symbols is just a filling (partial or complete) of the locations in the system and two dispositions are deemed equivalent if their filled positions correspond in an order isomorphism. A state of the system is then an equivalence class of dispositions which we can conveniently represent by the unique disposition whose filled positions contain the values $1, 2, \ldots, t$ for some $t \leq k$. For example the dispositions of a deque shown in Figure 4 are equivalent, and, for $k = 3$, the states of a deque are given by the representative dispositions in Figure 5.

Every operation of the system can now be modelled by a transition of the automaton with those operations that produce output causing the rank of the output symbol to be output. The automaton can now be determinised and minimised and, from its state equations, we can compute the growth rate of such a k-bounded system which is, of course, a lower bound on the unrestricted

Table 4: Lower bounds for two serial stacks

k	Non-det states	Det States	Min states	Growth lower bound
5	196	171	148	4.99915
6	625	3407	3240	5.96892
7	2055	75411	73592	6.82949
8	6917	1730025	1698923	7.5535
9	23713	41211076		8.156

system. The major bottleneck in this approach is a state explosion in the determinisation phase. This is so severe that we have only a marginal improvement to the lower bound of 8 implied by [4] for the growth rate of the two serial stacks system (see table 4); for $k = 9$ the computation required several days and the deterministic automaton was not minimised. For both deques and for two parallel stacks we have been able to take advantage of some special features of these systems that allow us to take the computations considerably further.

4.2 Direct construction of the deterministic automaton

We shall discuss the automaton that accepts the (rank-encoded forms of) permutations generated by two stacks in parallel. A similar approach applies to a deque and we shall indicate the necessary modifications at the end of this subsection.

There are many ways in which a pair of parallel stacks can generate a given permutation. Every computation can be described by a sequence of state transitions in the non-deterministic automaton given above. The deterministic version of the automaton manages to keep track of all the ways in which a given permutation can be generated and we shall show how it can be constructed directly. The direct construction is more efficient than applying the usual determinising procedure to the non-deterministic automaton because this procedure generates many more states than necessary and has to be followed up by a state minimisation algorithm.

We shall refer to states in the non-deterministic automaton as N-states and reserve the unqualified word "state" for a state in the deterministic automaton. An N-state defines the contents of the two stacks and, implicitly, the order in which the stacks have been filled

To illustrate the basic idea suppose the first symbol to be output by the parallel stack system is symbol 4. After this symbol has been output the system must be in one of 8 possible N-states (4 of them are shown in the box on the left of Figure 6 and the other 4 have the stacks interchanged). We cannot distinguish between these 8 possible N-states and have to incorporate all of them into a single state of the deterministic automaton. But now, suppose the next symbol

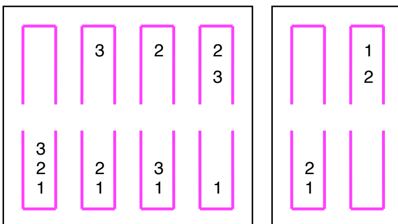

Figure 6: Some N-states

to be output is symbol 1. That means that only some of these N-states were possible and the system now passes into one of the two N-states in the box on the right of Figure 6; the corresponding deterministic state can be described by saying that the two symbols in the system belong to the same stack. In the notation we are about to develop the prior state would be described as (L, L, L) and the successor state as (LL).

In general, states of the deterministic automaton are denoted by sequences of words over a two-letter alphabet $\{L, R\}$ with each word beginning with L. The start state and final state are each the empty sequence. A state $(w_1, \ldots, w_k)$ denotes a set of possible N-states in the following way. The letters of w_1 specify which stack contains the symbols $1, 2, \ldots, |w_1|$ to within choice of stack name; for example $w_1 = LRL$ would specify that, of the smallest three symbols, the middle one was in a different stack to the first and third. The letters of w_2 then specify similar information about the next $|w_2|$ smallest set of symbols, and so on.

The essential property of this notation for states is that it enables us to tell which possible rank-encoded symbols can be output next and the state that each output produces. We now give the the transition rules for the deterministic automaton (and some commentary to explain them). The *height* of a state $(w_1, \ldots w_k)$ is defined as $\sum_{i=1}^{k} |w_i|$.

- From the start state, output r where $1 \leq r \leq k$ and go to state $(L, L, \ldots, L)$ where there are $r-1$ Ls. This corresponds to inserting the first r symbols in the the two stacks (after which symbol r will be on top of one of the stacks) and then outputting symbol r. The resulting state represents all possible ways in which the first $r-1$ symbols can reside in the stacks.
- From a state $(w_1, \ldots w_k)$ of height h output s where $h+1 \leq s \leq k$ and go to a state $(w_1, \ldots, w_k, L, \ldots L)$ with $s-h-1$ new Ls at the end (of height $s-1$). This corresponds to having to insert new symbols into the stacks (in any way at all) up to and including the one (the symbol of rank s) that is to be output, and then outputting it.

- From a state $(w_1, \ldots w_k)$ of height h we need to determine whether it is possible to output a symbol s with $1 \leq s \leq h$. The condition $1 \leq s \leq h$ is the condition that s already resides in one of the two stacks. Such a symbol can be output only if it is at the top of a stack and so we have to examine $(w_1, \ldots, w_k)$ to see whether any of the N-states associated with it have this property.

  To do this let b be such that w_b contains the sth symbol in $w_1, \ldots, w_k$. In fact, let $w_b = uxv$ where x is the sth symbol (u, v or both may be empty).

  If v contains any occurrence of the symbol x then this output is forbidden, or leads to the fail state (since, among the symbols represented by v, one is above the sth symbol in its stack). If any of $w_{b+1} \ldots w_k$ (w_c say) contains an R then this output is also forbidden (because w_c would contain both an L and an R one of which would represent a symbol on top of the sth symbol).

  Otherwise it is possible to output s. The new state that the automaton passes to reflects the fact that all the symbols that were inserted after s was inserted lie in the stack that did not contain s. This new state is represented by the sequence $(w_1, \ldots, w_{b-1}, w)$ where w is a word of length $|w_b| + |w_{b+1}| + \cdots + |w_k|$ and

  - If u is non-empty (so contains some L) and $xv = LRR \cdots R$ then $w = uRR \cdots R$
  - If u is empty and $xv = LRR \cdots R$ then $w = LL \cdots L$
  - If $xv = RLL \cdots L$ then $w = uLL \cdots L$

  If w is empty then the new state is simply given by $(w_1, \ldots w_{b-1})$

For the deque system we proceed along the same general lines. Here the symbols L, R indicate which end of the deque (rather than which stack) has received an input symbol. When the deque is non-empty we can distinguish whether subsequent symbols are being added to different ends or not using the smallest symbol in the deque as a divider. But we do not encode the smallest symbol in the deque (initially placed there when the deque was empty) as part of a state. Instead we differentiate the situation of the deque being empty (as final and start start) from the situation where it has one symbol only in it (encoding the latter situation as the empty sequence). This adds a little complexity to the state transition rules as we have to make special provision for the start state and we have to amend the transition rules a little so that the presence of the smallest symbol in the deque is not explicitly encoded.

4.3 A quotient automaton

The explicit description of the deterministic automaton (for either two parallel stacks or deques) allows a further optimisation technique.

Let Γ be the state graph of the automaton and let g be its number of vertices (states in the automaton). Let $A = (a_{pq})$ be the $g \times g$ (weighted) adjacency matrix of Γ; so a_{pq} is the number of transitions from state p into state q.

The states are encoded as sequences of words over the alphabet $\{L, R\}$. We define the *shape* of such a word w as the pair $\mathrm{Sh}(w) = (a, b)$ where a is the number of occurrences of L in w and b is the number of occurrences of R. We also define the *signature* of a state $(w_1, \ldots w_k)$ as the sequence $(\mathrm{Sh}(w_1), \ldots \mathrm{Sh}(w_k))$. We then partition the set of states according to their signatures. Let h be the number of signature classes. Denote the class of a state p by $[p]$. Furthermore define a $g \times h$ matrix C with rows indexed by states and columns indexed by signatures where

$$\begin{aligned} c_{p,[q]} &= 1 \text{ if } p \in [q] \\ c_{p,[q]} &= 0 \text{ otherwise} \end{aligned}$$

A routine check from the state transition rules establishes that the number of transitions from a state p to states of some given signature class $[q]$ depends only on the signature of p. This means that there is a well-defined $h \times h$ matrix B with rows and columns indexed by signature classes where

$$b_{[p][q]}$$

is the number of edges from vertex p to vertices in the signature class $[q]$. This matrix is the adjacency matrix of a graph Θ whose vertices are signature classes with $b_{[p][q]}$ edges from $[p]$ to $[q]$.

Lemma 1 $AC = CB$

Proof: The $(p, [q])$ entry of the LHS is

$$\sum_r a_{pr} c_{r,[q]} = \sum_{r \in [q]} a_{pr} = b_{[p][q]}$$

The $(p, [q])$ entry of the RHS is

$$\sum_{[r]} c_{p,[r]} b_{[r][q]} = b_{[p][q]}$$

■

Therefore, for all $n \geq 0$,

$$A^n C = C B^n$$

This equation can be interpreted in terms of paths. The $(p, [q])$ entry of the LHS is the number of paths of length n in Γ from vertex p that end in class $[q]$. The RHS is the number of paths in the graph Θ from the class $[p]$ to the class $[q]$.

Table 5: Lower bounds for two parallel stacks

k	States	Signatures	Growth lower bound
12	49209	11128	6.934
13	147624	27608	7.076
14	442869	68504	7.196
15	1328604	169968	7.300
16	3985809	421728	7.389
17	11957424	1046384	7.467
18	35872269	2596288	7.535

Table 6: Lower bounds for a deque

k	States	Signatures	Growth lower bound
12	16405	3031	7.28261
13	49210	7515	7.40386
14	147625	18643	7.50414
15	442870	46251	7.58801
16	1328605	114755	7.65886
17	3985810	284723	7.7192
18	11957425	706450	7.77117
19	35872270	1752834	7.81612
20	107616805	4349122	7.8553
21	322850410	8368833	7.890

In the case that p is the initial state of Γ and q is the final state (both unique in their signature classes) we can infer that the number of paths of length n from p to q in Γ is the same as the number of paths of length n from $[p]$ to $[q]$ in Θ.

This means that we can determine the growth rate of the restricted system by working within the automaton whose state graph is Θ rather than Γ and this great reduction in the number of states allows us to carry out the computations whose results are summarised in Tables 5 and 6.

5 Conclusions and open questions

We have given upper and lower bounds on growth rates for three natural permuting machines as summarised in Table 7.

Our upper bound techniques would apply to any permuting machines whose

Table 7: Growth rate bounds

	Lower bound	Upper bound
Two stacks in series	8.156	13.374
Two stacks in parallel	7.535	8.3461
Deque	7.890	8.352

effect is achieved by sequences of operations from a finite set of possible operations. In principle the techniques for lower bounds could also be applied to other systems although their utility is limited by storage considerations. In both cases the technology of finite automata and regular languages is used and we obtain better approximations to the growth rate the better the automata approximate the permuting machines.

There is a great similarity between two stacks in parallel and a deque. Indeed, any permutation that two parallel stacks can generate can also be generated by a deque since the two ends of the deque can represent the stacks. Our bounds show that the growth rates for these two systems cannot be very different and we suspect that they may be equal. An even more optimistic hope is that the common growth rate is equal to 8.

By contrast we seem to understand two stacks in series rather less. It is possible that this system is intrinsically more complex than either a deque or two parallel stacks. One reason for believing this is that we have no efficient test for whether a given permutation can be generated by two stacks in series, whereas the membership problem for deques and two parallel stacks is in the complexity class $\mathcal{P}$ (see [11, 5]). It is possible that the problem of deciding whether a given permutation is the product of two stack permutations (which is exactly the same as being generated by two serial stacks) is NP-complete.

References

[1] M. H. Albert, M. D. Atkinson, N. Ruškuc: Regular closed sets of permutations, Theoretical Computer Science 306 (2003), 85–100.

[2] R. Arratia: On the Stanley-Wilf Conjecture for the Number of Permutations Avoiding a Given Pattern. Electronic J. Combinatorics 6, No.1, N1, 1–4, 1999.

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

[4] M. D. Atkinson, M. M. Murphy, N. Ruškuc: Sorting with two ordered stacks in series, Theoretical Computer Science 289 (2002), 205–223.

[5] S. Even and A. Itai: Queues, stacks and graphs, in Z. Kohavi and A. Paz, eds., Theory of Machines and Computations, Proc. Internat. Symp. on the Theory of Machines and Computations, Technion – Israel Inst. of Technol., Haifa, Israel, August 1971 (Academic Press, New York, 1971) 71–86.

[6] D.E. Knuth: *Fundamental Algorithms, The Art of Computer Programming* Vol. 1 (First Edition), Addison-Wesley, Reading, Mass. (1968).

[7] D.E. Knuth: *Sorting and Searching, The Art of Computer Programming* Vol. 3 (First Edition), Addison-Wesley, Reading, Mass. (1973).

[8] A. Marcus, G. Tardos: Excluded Permutation Matrices and the Stanley-Wilf Conjecture, J. Combin. Theory Ser. A 107 (2004), no. 1, 153–160.

[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. Simion, F.W. Schmidt: Restricted permutations, Europ. J. Combinatorics 6 (1985), 383–406.

[11] P. Rosenstiehl, R. E. Tarjan: Gauss Codes, Planar Hamiltonian Graphs, and Stack-Sortable Permutations, J. Algorithms 5 (1984), 375–390.

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