

THE ENUMERATION OF PERMUTATIONS SORTABLE BY POP STACKS IN PARALLEL

REBECCA SMITH

Department of Mathematics
SUNY Brockport
Brockport, NY 14420

VINCENT VATTER

Department of Mathematics
Dartmouth College
Hanover, NH 03755

We show that the set of permutations sortable by k pop stacks in parallel has a regular insertion encoding and construct the (finite) recognizing automaton for this language. This shows that these permutations have a rational generating function, verifying a conjecture of Atkinson and Sack.

A stack is a last-in first-out sorting device with push and pop operations. Knuth [7] showed that a permutation π can be sorted (meaning that by applying push and pop operations to the sequence of entries $\pi(1), \dots, \pi(n)$ we can output the sequence $1, \dots, n$) if and only if π avoids the permutation 231, i.e., if and only if there do not exist three indices $1 \leq i_1 < i_2 < i_3 \leq n$ such that $\pi(i_1), \pi(i_2), \pi(i_3)$ are in the same relative order as 231. Shortly thereafter Tarjan [10], Even and Itai [5], and Pratt [9] studied networks which have multiple stacks either in series or in parallel. The questions typically studied for such networks include:

- Can the set of sortable permutations be characterized by a finite set of forbidden permutations (e.g., $\{231\}$ in the stack case)?
- How many permutations of each length can be sorted?

For $k \geq 2$ stacks in series or in parallel the answer to the first question is no, due to Murphy [8] and Tarjan [10], respectively. The enumeration of the sortable permutations is unknown in both cases.

Date: March 8, 2009

Key words and phrases. algorithms, data structures, insertion encoding, permutation, pop stack, sorting
AMS 2000 Subject Classification. 05A17, 06A07

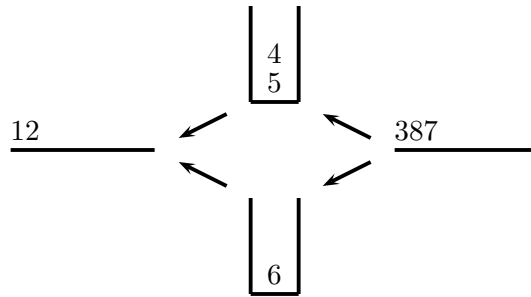


Figure 1: Sorting with two parallel pop stacks

Here we consider *pop stacks*, which are stacks where the only way to move an element from the stack to the output is to pop everything in the stack (in last-in first-out order). Avis and Newborn [4] introduced pop stacks and studied their properties when placed in series, which by their interpretation meant that when the entire set of items currently in the i th pop stack is popped, it is pushed onto the $(i + 1)$ st pop stack. They proved that the set of permutations sortable by k pop stacks in series can be characterized by a finite set of forbidden permutations and provided the enumeration of these permutations for every k ; as k approaches infinity, these sets increasingly resemble the “separable permutations”. Atkinson and Stitt [3] gave a simpler derivation of these enumeration results using what is now known as the substitution decomposition, and considered a slightly different serial construction for pop stacks. In their serial construction, one saves the output of the i th pop stack — essentially, in a queue between the pop stacks — and inputs these entries into the $(i + 1)$ st pop stack with push operations. Atkinson and Stitt provided the rational generating function for 2 pop stacks linked serially in this manner.

In [2], Atkinson and Sack proved that the set of permutations sortable with k pop stacks in parallel is also characterized by a finite set of forbidden permutations, provided the enumeration of these permutations in the $k = 2$ case, and conjectured that for all k , these permutations have a rational generating function. Our aim in this paper is to prove their conjecture.

First consider an example, sorting the permutation 25164387 with two pop stacks in parallel. We begin by pushing the 2 onto one of the stacks. Then since we do not want the 5 to appear before the 2 in the output, we must push the 5 onto the other stack. Following that, we push the 1 onto the stack already containing the 2 (since we must clear the entire stack when we perform the pop operation) and then pop that stack yielding an initial output of 12. Next we push the 6 onto the stack vacated by the 1 and 2. At this point, we push the 4 onto the stack containing the 5 to get to the stage shown in Figure 1. Following this, we push the 3 onto the stack containing the 4 and 5 and then pop that stack for an output of 12345. We can then pop the stack containing the 6 to get 123456 in the output. Then push 8 onto one of the empty stacks. At this point, we may push 7 onto either stack, but for consistency we will push it onto the stack already containing the 8 and then pop that stack. This gives us our complete, sorted output 12345678.

As this example illustrates, there is a canonical algorithm for sorting a permutation of $\{1, 2, \dots, n\}$ with k pop stacks in parallel. This algorithm can be characterized by the following rules:

- (S1) If the topmost entry of a stack is 1 greater than the most recently output symbol, pop this stack. (Also, if no symbols have been output and the topmost entry of a stack is 1, pop this stack.)
- (S2) If the next entry in the input is 1 less than the top entry on one of the stacks, push this entry onto that stack.
- (S3) If the next entry in the input is not 1 less than the top entry on any of the stacks, push it onto an empty stack.

One can see that this algorithm will result in any sortable permutation being sorted. First suppose that the most recently output symbol was i , and that the topmost entry of a stack is $i + 1$. It follows by (S2) that this stack contains $i + 1, i + 2, \dots$, and so popping this stack, i.e., applying (S1), only serves to free up a stack not otherwise available for further use. Second, pushing an entry i onto a stack where the current top entry is $i + 1$, applying (S2), ensures that these two entries will be output in the correct order. Finally, if we need to push a new element i onto a stack and there is no stack with $i + 1$ as its top entry, we must push it onto a new stack, that is, we must apply (S3). (Otherwise, if we push i onto a stack with top entry $j \neq i + 1$, then when that stack is popped, our output permutation will contain the adjacent pair of entries ij and therefore will not be the identity permutation.)

In order to enumerate these permutations we use a symmetry of the *insertion encoding* introduced by Albert, Linton, and Ruškuc [1], which is a correspondence between permutation classes and formal languages. This correspondence in its normal form associates to each permutation a word describing how it evolved from bottom (the entry 1) to top (the entry n). We instead consider a symmetry of the insertion encoding, which describes how permutations are built from left to right. At each stage until the desired permutation has been constructed (these stages are called configurations), at least one open slot (represented by \diamond s, we label slots from bottom to top) exists in the intermediate configuration, and to proceed to the next configuration we insert a new rightmost entry (which is therefore the leftmost of the entries we have yet to insert) into one of the slots. This insertion can occur in one of four possible ways:

- f_j : fill the j th slot, removing it,
- b_j : insert a new bottommost entry into the j th slot,
- t_j : insert a new topmost entry into the j th slot,
- m_j : insert a new “middle entry” into the j th slot, introducing another slot.

For example, the encoding of the permutation 25341 is $\mathbf{m}_1\mathbf{t}_2\mathbf{b}_2\mathbf{f}_2\mathbf{f}_1$, which corresponds to the following sequence of insertions:

$$\begin{array}{ccccccc}
 & & \diamond & & \pi(2) & & \pi(2) & & \pi(2) \\
 & & & & \diamond & & \diamond & & \diamond \\
 \diamond & \xrightarrow{\mathbf{m}_1} & \pi(1) & \xrightarrow{\mathbf{t}_2} & \pi(1) & \xrightarrow{\mathbf{b}_2} & \pi(3) & \xrightarrow{\mathbf{f}_2} & \pi(3) & \xrightarrow{\mathbf{f}_1} & \pi(3) \\
 & & \diamond & & \diamond & & \pi(1) & & \pi(1) & & \pi(1) \\
 & & & & \diamond & & \diamond & & \diamond & & \pi(5)
 \end{array}$$

From this we see that $\pi(5), \pi(1), \pi(3), \pi(4), \pi(2) = 1, 2, 3, 4, 5$, or in other words that $\pi = 25341$.

One advantage of this encoding for our purposes is quite easy to see. Suppose that we are attempting to sort π with k pop stacks in parallel and that the insertion encoding of π contains more than $k + 1$ open slots in some configuration, say configuration i (considering \diamond to be configuration 0). This means that among the entries $\pi(1), \dots, \pi(i)$, there are at least $k + 1$ vertical “gaps” between them which in turn implies, via the canonical sorting algorithm, that in order to input all these entries we would need $k + 1$ pop stacks, a contradiction. Therefore if π can be sorted by k pop stacks in parallel, it can never have more than $k + 1$ open slots, and so the insertion encoding for the set of all permutations that can be sorted by k pop stacks in parallel is a language over a finite alphabet.

We will use only basic results from the theory of formal languages, which may be found in a variety of texts, for example, Hopcroft, Motwani, and Ullman [6]. A *nondeterministic finite automaton* over the alphabet A consists of a set S of *states*, one of which is designated the *initial state*, a *transition function* δ from $S \times (A \cup \{\varepsilon\})$ into the power set of S , and a subset of S designated as *accept states*. The *transition diagram* for this automaton is a directed graph on the vertices S , with an arc from r to s labeled by a precisely if $s \in \delta(r, a)$. The initial state is designated by an inward-pointing arrow. An automaton *accepts* the word $w_1 \cdots w_m$ if there is a walk from the initial state to an accept state whose arcs are labeled (in order) by w_1, \dots, w_m ; the set of all such words is the *language accepted* by the automaton. A language accepted by a nondeterministic finite automaton is called *recognizable*. By Kleene’s theorem, the recognizable languages are precisely the *regular languages*. Regular languages have many nice properties, but the only one we will make use of is that they have rational generating functions; as the length of the insertion encoding of π is the same as the length of π , it follows that if a set of permutations has a regular insertion encoding, then it has a rational generating function.

In order to prove that the set of insertion encodings of permutations sortable by k pop stacks in parallel forms a regular language, we construct its finite accepting automaton. Suppose that a configuration has t slots and that we have applied the canonical sorting algorithm to sort the corresponding permutation up to this point. The state that the accepting automaton is in at this point is labeled by the vector (s_1, \dots, s_t) , where s_i denotes the number of stacks being used to house entries between slot i and $i + 1$ (or simply above slot t in the $i = t$ case). Thus our states are labeled by vectors of nonnegative integers of length at most $k + 1$ with sum at most k . In fact, we can say a bit more; since the only way

to add a new slot is to use the operation \mathbf{m}_j , in which case we must use a stack to fill the entry between the new slot s_j and s_{j+1} , we can conclude that $s_j > 0$ for all $j < t$. Thus only the final entry of our vectors can be 0, and so it is easy to compute that there are 2^{k+1} states (including the empty vector, ε , which is our accept state).

We now need to describe the transitions between states. These can be briefly described as follows:

$$\begin{aligned}
(s_1, \dots, s_t) &\xrightarrow{\mathbf{f}_j} \begin{cases} (s_2, \dots, s_t) & \text{if } j = 1, \\ \text{reject} & \text{if } j = t, s_t = 0, \text{ and } \sum s_i = k, \\ (s_1, \dots, s_{t-2}, s_{t-1} + 1) & \text{if } j = t, s_t = 0, \text{ and } \sum s_i < k, \\ (s_1, \dots, s_{j-2}, s_{j-1} + s_j, s_{j+1}, \dots, s_t) & \text{otherwise.} \end{cases} \\
(s_1, \dots, s_t) &\xrightarrow{\mathbf{b}_j} \begin{cases} (s_1, \dots, s_t) & \text{if } j = 1 \text{ and } \sum s_i < k, \\ (s_1, \dots, s_{j-2}, s_{j-1} + 1, s_j, \dots, s_t) & \text{if } j \neq 1 \text{ and } \sum s_i < k, \\ \text{reject} & \text{otherwise.} \end{cases} \\
(s_1, \dots, s_t) &\xrightarrow{\mathbf{t}_j} \begin{cases} (s_1, \dots, s_{t-1}, 1) & \text{if } j = t, s_t = 0, \text{ and } \sum s_i < k, \\ \text{reject} & \text{if } j = t, s_t = 0, \text{ and } \sum s_i = k, \\ (s_1, \dots, s_t) & \text{otherwise.} \end{cases} \\
(s_1, \dots, s_t) &\xrightarrow{\mathbf{m}_j} \begin{cases} (s_1, \dots, s_{j-1}, 1, s_j, \dots, s_t) & \text{if } t < k + 1 \text{ and } \sum s_i < k, \\ \text{reject} & \text{otherwise.} \end{cases}
\end{aligned}$$

These transitions are similar, and so we describe only the first set, the \mathbf{f}_j transitions.

Suppose that we are at state (s_1, \dots, s_t) and we read \mathbf{f}_1 . This instructs us to fill the bottommost slot. As we have already output all entries less than this (using the canonical sorting algorithm), this new entry is therefore the next entry we need to output, and also, as this entry is smaller than everything currently in a stack, it can be pushed onto the stack which contains the next smallest entry (or a new stack if all the stacks are currently empty). After this, that stack and all of the other stacks counted by s_1 can be popped into the output yielding the new state (s_2, \dots, s_t) .

Now consider the \mathbf{f}_t case when $s_t = 0$. If $\sum s_i = k$, when we read this entry from the input, it is larger than all of the elements currently in the stacks and thus we reject the word since all of the stacks are currently occupied. However, if there is an empty stack ($\sum s_i < k$) then we can push this entry onto that stack, which now occupies the $(t - 1)$ st slot, the t th slot is eliminated, and the new state becomes $(s_1, \dots, s_{t-2}, s_{t-1} + 1)$.

At this point, suppose we read \mathbf{f}_t when $s_t > 0$, and that this letter corresponds to the entry i . Since $s_t > 0$, there is a stack counted by s_t that contains $i + 1$ as its topmost element. Thus we can place i onto that stack. Also, because slot t has been filled, we now only have $t - 1$ slots, leaving us with state $(s_1, s_2, \dots, s_{t-1} + s_t)$.

The case of \mathbf{f}_j for $2 \leq j \leq t - 1$ is similar to the previous $j = t$ case where $s_t > 0$ because

here there is always at least one stack holding entries between the j th and $(j + 1)$ st slots, and so we may push the corresponding entry onto one of these stacks with resulting state $(s_1, \dots, s_{j-2}, s_{j-1} + s_j, s_{j+1}, \dots, s_t)$.

This establishes that the set of insertion encodings of permutations sortable by k pop stacks in parallel constitutes a regular language, from which we may immediately deduce our main result.

Theorem 1. *For any k , the set of permutations that can be sorted by k pop stacks in parallel has a rational generating function.*

We conclude by listing these generating functions for small k . These computations were performed by the Maple package POPSTACKS, available from the second author's homepage.

k	generating function
1	$(x - 1)/(2x - 1)$
2	$\frac{2x^2 - 6x^2 + 5x - 1}{6x^3 - 10x^2 + 6x - 1}$
3	$\frac{18x^6 - 98x^5 + 177x^4 - 148x^3 + 63x^2 - 13x + 1}{72x^6 - 216x^5 + 288x^4 - 201x^3 + 75x^2 - 14x + 1}$
4	$\frac{1584x^{11} - 14016x^{10} + 45432x^9 - 79454x^8 + 86562x^7 - 62624x^6 + 30887x^5 - 10411x^4 + 2352x^3 - 339x^2 + 28x - 1}{7920x^{11} - 40166x^{10} + 96324x^9 - 139728x^8 + 133468x^7 - 87278x^6 + 39670x^5 - 12495x^4 + 2666x^3 - 366x^2 + 29x - 1}$

REFERENCES

- [1] ALBERT, M. H., LINTON, S., AND RUŠKUC, N. The insertion encoding of permutations. *Electron. J. Combin.* 12, 1 (2005), Research paper 47, 31 pp.
- [2] ATKINSON, M. D., AND SACK, J.-R. Pop-stacks in parallel. *Inform. Process. Lett.* 70 (1999), 63–67.
- [3] ATKINSON, M. D., AND STITT, T. Restricted permutations and the wreath product. *Discrete Math.* 259, 1-3 (2002), 19–36.
- [4] AVIS, D., AND NEWBORN, M. On pop-stacks in series. *Utilitas Math.* 19 (1981), 129–140.
- [5] EVEN, S., AND ITAI, A. Queues, stacks, and graphs. In *Theory of machines and computations (Proc. Internat. Sympos., Technion, Haifa, 1971)*. Academic Press, New York, 1971, pp. 71–86.
- [6] HOPCROFT, J. E., MOTWANI, R., AND ULLMAN, J. D. *Introduction to automata theory, languages, and computation*, 2nd ed. Addison-Wesley Publishing Co., Reading, Mass., 2001.

-
- [7] KNUTH, D. E. *The art of computer programming. Volume 3*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1973. Sorting and searching, Addison-Wesley Series in Computer Science and Information Processing.
- [8] MURPHY, M. M. *Restricted Permutations, Antichains, Atomic Classes, and Stack Sorting*. PhD thesis, Univ. of St Andrews, 2002.
- [9] PRATT, V. R. Computing permutations with double-ended queues, parallel stacks and parallel queues. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing* (New York, NY, USA, 1973), ACM Press, pp. 268–277.
- [10] TARJAN, R. Sorting using networks of queues and stacks. *J. Assoc. Comput. Mach.* 19 (1972), 341–346.