The Insertion Encoding

Michael Albert

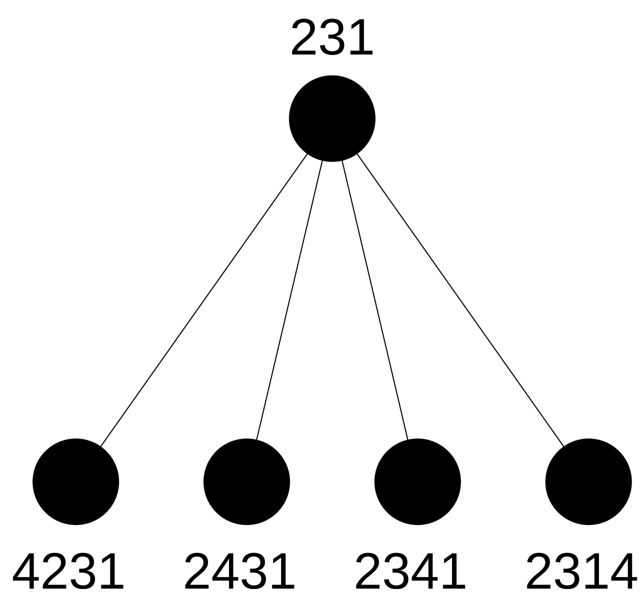
Steve Linton, Nik Ruškuc

Department of Computer Science, University of Otago

CIRCA, University of St. Andrews

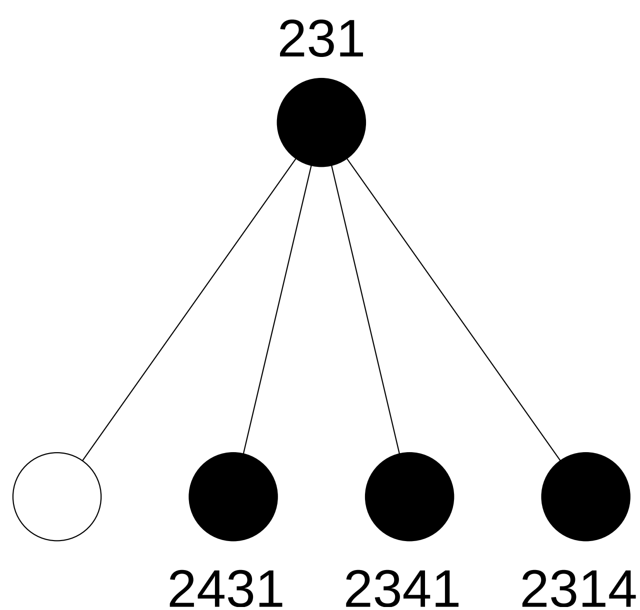
Constructing by inserting

- The construction or generation of a permutation can be thought of as proceeding by successive insertion of a new maximum element.
- The set of all permutations on $[n] = \{1, 2, \ldots, n\}$ is then naturally viewed as the nodes at depth n in a tree, where the children of any node are the permutations obtained from it by insertion of a new maximum element.



Insertions and classes

- The construction of permutations in a pattern class may (and generally does) restrict the positions in which the maximum element can be inserted, thereby pruning the tree.
- This observation is used in the method of *generating trees* (West, etc.)



Generating trees

- Given a permutation π in some pattern class $\mathcal{C}$ there will generally be one or more positions where a new maximum element can be inserted into π without leaving $\mathcal{C}$.
- These positions are called *active sites*.
- But note, after an insertion elsewhere, a previously active site may become inactive.

312-avoiders

Consider the class $\mathcal{A}(312)$ of 312-avoiders. In the permutation

$$2\,1_{\uparrow}\,4_{\uparrow}\,3_{\uparrow}$$

the active sites are marked with $\uparrow$.

The *number* of active sites after the next insertion depends on which site is used (ranging from 4 if the leftmost site is used down to 2 if the rightmost one is used).

The general rule is:

$$(k) \to (k+1)(k) \cdots (2)$$

Summary

- An *active site* is a position where an insertion *may* take place.
- The number of active sites is the number of children of any node in the generating tree for $\mathcal{C}$.
- If the *types* of these nodes can be derived from the types of their parent, then often the class can be enumerated using the resulting recurrences.
- See also the *ECO* (Enumeration of Combinatorial Objects) methodology of the Italian group.

A change of perspective

- Return to the generation of arbitrary permutations by successive insertion. However, consider a specific *target* permutation instead of the set of all permutations.
- Now it makes sense only to consider those positions where an insertion *will* take place. In order to avoid confusing terminology we refer to these as *slots* rather than *active sites*.

3546217

$\diamond$

$\diamond\, 1 \,\diamond$

$\diamond\, 21 \,\diamond$

$3 \diamond 21 \diamond$

$3 \diamond 4 \diamond 21 \diamond$

$354 \diamond 21 \diamond$

$354621 \diamond$

3546217

Types of insertions

- As seen in the example above there are four possible types of insertions:

M In the *middle* of a slot, splitting it into two slots.

L At the *left hand end* of a slot, leaving a slot to the right.

R At the *right hand end* of a slot, leaving a slot to the left.

F *Filling* a slot (leaving no remaining slot).

- Each of these should be subscripted by the number of the slot in which the insertion is taking place.

Decoding an insertion sequence

- Consider $M_1 M_2 F_2 R_1 F_2$:

$$\begin{array}{cc}
\diamond & \\
\diamond\, 1 \,\diamond & M_1 \\
\diamond\, 1 \,\diamond\, 2 \,\diamond & M_2 \\
\diamond\, 132 \,\diamond & F_2 \\
\diamond\, 4132 \,\diamond & R_1 \\
\diamond\, 41325 & F_2
\end{array}$$

- Since there is still a slot open, this does not represent the encoding of a permutation. Appending an F_1 would give us 641325.

The big picture

- The language of permutations in the insertion encoding can be thought of in terms of a stack automaton. In fact the stack is simply used as a counter, k, for the number of slots. The allowed transitions are:

From	To	Using
k	k	L_i, R_i
k	$k+1$	M_i
k	$k-1$	F_i.

$k > 0$ and $1 \leq i \leq k$. The initial state has $k = 1$ and $k = 0$ is the unique final state.

A minor observation

- Any encoding beginning $M_1 M_2 \cdots$ (as in our example) will eventually produce a permutation containing the pattern 312.
- This is clear, since the situation after this beginning is $\diamond 1 \diamond 2 \diamond$, and eventually something will be placed in the leftmost slot, resulting in a 312 pattern.
- Indeed it's clear that if the insertion encoding of π contains any symbol X_j with a subscript other than 1, then π contains the pattern 312 (witnessed by the element which created the left boundary of slot j, the element placed by X_j, and any later insertion with a subscript smaller than j).

$\mathcal{A}(312)$

- The converse of the preceding observation also holds. That is, any insertion encoding which has only 1 subscripts generates a 312-avoider.
- Suppressing the subscript gives a language for 312-avoiders over the alphabet $\{M, L, R, F\}$ whose grammar is:

$$s \to F \,|\, Ls \,|\, Rs \,|\, Mss$$

- This immediately yields the Catalan generating function, and a length preserving, symbol for symbol encoding.
- $\mathcal{A}(321)$ can be handled similarly.

$\mathcal{A}(321)$

- To handle $\mathcal{A}(321)$ it's slightly easier conceptually to insist on an unfillable slot at the right hand end of the permutation.
- The acceptable operations are:
  - If there is exactly one slot, L, or M.
  - If there are two or more slots L_1, F_1, or L_{-1}, M_{-1}.
- This gives a grammar (s encodes 321-avoiders including the empty permutation):

$$\begin{aligned} s &\longrightarrow \epsilon \,|\, Ls \,|\, Mts \\ t &\longrightarrow F_1 \,|\, L_1 t \,|\, M_{-1}tt \,|\, L_{-1}t \end{aligned}$$

Language issues

- The full language for insertion encoding is infinite. Generally speaking this is a problem for using the machinery of formal languages. So how do we restrict to a finite language?
- One possibility (as above) is to restrict the locations where insertions may take place at any time. In order to ensure that a class is obtained some care is needed here (example follows).
- More violently, we could require that the number of slots be bounded. This yields classes with *regular* encodings.

The regular case I

- Consider permutations whose insertion encoding only ever contains at most 2 slots. These form a pattern class because the excluded conditions:

$$\diamond \, a \diamond b \diamond$$

  can be represented as a set of permutations.

- These permutations are all those of the form:

$$xaybz$$
$$\{a, b\} = \{1, 2\}$$
$$\{x, y, z\} = \{3, 4, 5\}$$

The regular case II

- The obvious generalization to at most k slots applies.
- The basis of the pattern class of all permutations whose insertion encoding never uses more than k slots at a time consists of the set $\mathcal{B}_k$ of permutations of the form:

$$babab \cdots ab$$

of length $2k+1$ where the b's are from $\{k+1, k+2, \ldots, 2k+1\}$ and the a's from $\{1, 2, \ldots, k\}$.

- Note that this is a rather large basis, it has $k!(k+1)!$ elements.

Vatter's theorem

- Theorem: *(V. Vatter) Let $\mathcal{B}$ be a finite set of permutations. The generating tree for $\mathcal{A}(\mathcal{B})$ is isomorphic to a finitely labelled tree if and only if $\mathcal{B}$ contains both a child of an increasing permutation and of a decreasing permutation.*
- Consequences include the existence of a rational g.f. for such classes and (implicitly) efficient recognition algorithms.

Prior notions of regularity

- In *TCS* **306**, Albert, Atkinson and Ruškuc introduced several notions of regularity for permutation classes, along with a mechanism for moving between classes and their bases.
- Roughly speaking these provided effective methods for constructing a class from its basis and vice versa (where “constructing” means “produce a finite state automaton for”). As corollaries, all the usual nonsense about generating functions and recognition.

Regular classes (Finis)

- The classes covered by Vatter's result are *subclasses* of $\mathcal{A}(\mathcal{B}_k)$ for suitable k as are the classes considered by AAR.
- Theorem: *Let $\mathcal{C}$ be any regularly based subclass of $\mathcal{A}(\mathcal{B}_k)$. Then:*
  - *The language representing the insertion encoding of $\mathcal{C}$ is regular.*
  - *$\mathcal{C}$ has a rational generating function.*
  - *There is a linear time recognition algorithm for $\mathcal{C}$.*
- In fact the full AAR mechanism applies (so we can also go from classes to bases).

Three and four

- West provided enumerations for all pattern classes having a basis element of length 3 and a basis element of length 4 using generating trees in almost all cases.
- In *all* these cases, the class (or one of its isomorphs) is represented by a context free language in the insertion encoding, recognized by a deterministic pushdown automata.
- All these automata are sufficiently simple that the enumerative results follow using the standard enumeration techniques for such languages.

Another example (after Kremer)

- Take $\mathcal{L}_1 = \{M_1, L_1, R_1, F_1, L_2, F_2\}$ and $\mathcal{L}_2 = \{M_1, L_1, R_1, F_1, R_{-1}, F_{-1}\}$.
- Both these languages define pattern classes for the insertion encoding with bases:

$$\{3142, 4132\}$$
$$\{3124, 4123\}$$

- So these classes are equinumerous (large Schroeder numbers).
- Their intersection has enumeration $\binom{2n-2}{n-1}$.

A conjecture

- The subclasses of $\mathcal{A}(312)$ are, in some sense, well-understood (they are all finitely based, all have rational g.f's, and in principle given a basis the g.f. can be computed).
- The same cannot be said of $\mathcal{A}(321)$. But:

  Conjecture: *Every finitely based subclass of $\mathcal{A}(321)$ has an algebraic generating function.*
- This shows the flavour of the area where the insertion encoding should be useful.

Conclusions

- The insertion encoding provides a framework for unifying many (most?) of the known explicit results on permutation class enumeration.
- Given a pattern class it can be used to answer the enumeration, generation and recognition questions pertinent to that class.
- It can also be applied to other collections of permutations (eg. Dumont plus pattern restrictions).
- Much remains to discover . . .

Thank you!