Pattern avoiding permutations are context-sensitive

Murray Elder
School of Computer Science
University of St. Andrews
North Haugh, St. Andrews
Fife, KY16 9SS, Scotland
murray@mcs.st-and.ac.uk

January 7, 2005

Abstract

In this talk we try to connect the fields of formal language theory from Computer Science, and pattern avoidance from Combinatorics.

In particular, we set out to describe the set of all permutations that avoid some pattern q in terms of formal languages, in an attempt to enumerate them. We prove that there is a bijection between the set of permutations which avoid q and a context-sensitive language. This language is a variation of the "insertion encoding" of Albert and Ruskuc.

In formal language theory there is a hierarchy of languages of increasing complexity, starting from regular languages, then context-free, indexed, context-sensitive, decidable and recursively enumerable languages, each one strictly contained in the previous one.

Chomsky and Schutzenberger proved that regular languages have rational generating functions, while (unambiguous) context-free languages have algebraic generating functions. Beyond this, little is known about generating functions for the higher languages. A natural question is to ask whether there is a formal language corresponding to D-finite generating functions. We give examples of indexed and context-sensitive languages whose generating functions are not D-finite.

In spite of this, the connections between languages and pattern avoidance is still worth pursuing, and I hope to put this point across through this talk.

The "insertion encoding" of a permutation is defined as follows. The idea is to build up a permutation by successively inserting the next highest entry in an open "slot," starting with a single open slot, until all slots are filled. One can insert in one of four ways:

Middle: Place the next entry in the "middle" of the slot, creating two slots from one.

Left: Place the next entry on the left of a slot.

Right: Place the next entry to the right of a slot.

Fill: Replace a slot by the next highest entry, thus decreasing the total number of slots by one.

For example, the instructions: "middle, right, left, fill, fill" or *mrlff* build the permutation 34215 as follows:

$$* \to *1* \to *21* \to 3*21* \to 3421* \to 34215$$

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

So the instructions *mrtltff* build the permutation 52134 as follows:

$$* \to *1* \to *21* \to *213* \to *2134 \to 52134$$

We can describe any permutation uniquely using these five symbols. Note that the length of the permutation equals the number of non-t-letters in the codeword, and the number of m-letters equals the number of f-letters minus 1. In addition, the number of t's preceding any non-t-letter is always at most the number of m's minus the number of f's to the left of the letter.

We can formalise these rules to describe the set of all codewords that come from legitimate permutations in terms of an indexed language.

We then consider how to describe the set of all codewords for permutations that avoid some fixed pattern q, and we can do this using a context-sensitive language.

In both cases, we make use of the machine-version of the language (rather than the grammar version) to come up with the languages. In the case of indexed languages we use "nested-stack automata" and for context-sensitive languages we use "linear bounded automata." In the talk, we will define these machines, and show how they prove our theorems.