Generic Enumerative Combinatorics using Automata

Joseph Slote

Introduction

Frequently in enumerative combinatorics we are presented with a countable collection of finite sets $S = \{S_1, S_2, \ldots\}$ and seek a counting function, or enumeration, $f(n) = |S_n|$ that counts the number of elements in S_n for all n. In many cases, there exists a bijection between these sets and collections of words in a formal language, suggesting that language hierarchies can be used to classify the complexity of combinatorial problems. Further, automata associated with each class of languages may provide generic counting methods for entire classes of problems. In what follows, I describe a technique to count all combinatorial problems associated with regular languages.

From Combinatorial Problems to Formal Languages

As in [1], given a language $\mathcal{L}$, we define the *n-slice* $\mathcal{L}_n$ of $\mathcal{L}$ as the set of words in $\mathcal{L}$ of length n. Now consider a collection of sets $S = \{S_1, S_2, \ldots\}$. A *language for S* is a language $\mathcal{L}$ and an injective size function $f : \mathbb{N} \to \mathbb{N}$ such that $|S_n| = |\mathcal{L}_{f(n)}|$ for all $n \in \mathbb{N}$. For instance, a language associated with Catalan paths might be

$$\left\{ x = x_1 x_2 \cdots x_n \in \{1, -1\}^* \;\middle|\; \sum_{k=1}^{n} x_k = 0 \text{ and } \sum_{j=1}^{k} x_j \geq 0 \, \forall k < n \right\} \text{ with } f(n) = n.$$

We call a language for S *simplest* if it is of minimal complexity in the formal language hierarchy. We then say the *language class of S* is the formal language class of its is simplest language and write $S' \geq S$ if the language class of S is included in S'.

These definitions provide a natural way to compare the complexity of enumeration problems. Further, each class of languages comes with its own combinatorial structure, often as a type of automata or grammar. These generic structures suggest counting techniques can be developed to address enumeration problems in a specific class *en masse*. Here I demonstrate this ability for the simplest class of languages, the regular languages.

Counting Regular Languages

The following theorem says we can count all regular collections in the same way.

Proposition. *If S is regular, then S has a closed-form matrix-algebraic enumeration.*

Proof by construction. Because S is regular, it has an associated regular language $\mathcal{L}$ with size function s. By definition, there exists a deterministic finite automaton $D = \big(Q = \{q_1, \ldots, q_m\}, \Sigma, \delta, q_1, F\big)$ that decides $\mathcal{L}$. We will determine $|\mathcal{L}_{s(n)}|$ by counting the number of paths of length n from the start state q_1 to each of the final states in F. To that end, define an $m \times m$ adjacency matrix A with elements

$$A_{ij} = |\{x \in \Sigma : \delta(q_i, x) = q_j\}|.$$

Then $(A^{S(n)})_{ij}$ counts the number of paths of length $s(n)$ in D from q_i to q_j. Extracting the paths to the accept states and summing, we have

$$f(n) = |S_n| = |\mathcal{L}_{s(n)}| = \begin{bmatrix} a_1 & \cdots & a_m \end{bmatrix} A^{s(n)} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \text{ where } a_i = \begin{cases} 1 & \text{if } q_i \in F \\ 0 & \text{otherwise.} \end{cases}$$

□

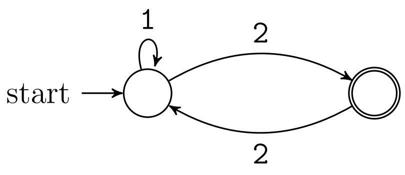
Note that this counting formula also works directly with NFAs that have neither ϵ-transitions nor multiple outgoing edges for the same symbol. Further, in the case where A is diagonalizable, we can use eigenvector/value methods to rewrite $f(n)$ as an analytic function.

In practice this theorem says that once we have a regular expression, DFA, or NFA to describe S, an enumeration comes "for free." In each of these cases there are algorithms (e.g. [2]) that generate A, so all the enumerative combinatorist must do is translate the description of S into a regular language. Often this translation requires very little effort, as is demonstrated in the following example.

Example. *How many ways are there to tile a strip of length $n-1$ with dominoes and monominoes?*

Solution. We are seeking a language $\mathcal{L}$ whose elements of length $s(n)$ describe tilings of a strip of length $n-1$. There are many ways to do this, but one way is to set $\Sigma = \{1, 2\}$ and use 1 to represent a monomino and 22 to represent a domino. We can make $s(n) = n$ if a string of length n denotes an $n-1$ tiling. To do this we must place a single extra symbol at the end of sequences of 1s and 22s; let's use 2. Our language is thus $\mathcal{L} = (1 \cup 22)^*2$, all possible sequences of dominoes and monominoes plus an extra symbol.

To find A, we first convert our regular expression to an NFA:



Because this NFA contains no ϵ-transitions or multiple outgoing edges for the same symbol, we can write an enumeration for directly without conversion to a DFA:

$$f(n) = |\mathcal{L}_{s(n)}| = |\mathcal{L}_n| = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

the familiar matrix form of the n^{th} Fibonacci number. Diagonalization of A yeilds Binet's formula:

$$f(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n.$$

□

Questions for Further Study

- Can we predict from $\mathcal{L}$ whether A is diagonalizable? Does the subclass of regular languages with diagonalizable transition matrices have any special structure?
- What is the complexity of the algorithm that, given a regular expression, derives the counting formula?
- What other formal language classes can be counted?
- Is there a natural language class that is associated with all counting problems that admit generating function descriptions?
- Can we find bounds on the computational complexity of a counting formula given the simplest language for a counting problem?

References

[1] V. Gore, M. Jerrum, S. Kannan, Z. Sweedyk, and S. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, 134(1):59–74, 4 1997.

[2] M. Sipser. *Introduction to the Theory of Computation.* Cengage Learning, 3 edition, 2012.