Iterative classes

Contents

1 Regular specifications

1.1 Recall: combinatorial specification

Definition. A specification for an r-tuple of classes $\mathcal{A} = (\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(r)})$ is a set of r equations

$$\begin{aligned} \mathcal{A}^{(1)} &= \Phi_1(\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(r)}) \\ &\vdots \\ \mathcal{A}^{(r)} &= \Phi_r(\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(r)}) \end{aligned}$$

where each Φ_i is built using the admisible constructions we know as well as the neutral class $\mathcal{E}$ and atomic class $\mathcal{Z}$.

You can think of drawing a graph of dependencies between these classes. If the class is acyclic then the problem can be solved iteratively. If it contains a cycle then the construction is recursive and can hopefully be solved as we solved the tree problem.

Definition. A class is said to be constructible (or specifiable) iff it admits a specification in terms of admissible operators.

We focus recursive and iterative constructions. Consider a graph of dependencies. If it is acyclic, the specification is said to be iterative. If there is a cycle containing $\mathcal{A}$, then we say that $\mathcal{A}$ is recursive.

1.2 Regular specifications

Previously we considered classes which were word families. These looked like $\mathcal{A} = \Phi(\mathcal{Z}_1, \mathcal{Z}_0)$. These are all iterative constructions.

Definition. An iterative specification (no recursion) that only involves atoms, combinatorial sums, cartesian products, and sequence constructions is said to be a regular specification. A language is said to be specification-regular, or, simply regular, if it is combinatorially isomorphic to a class of objects with a regular-specification.

This definition does not (apparently) match the usual definition of a regular language from computer science. In that context we would say a language is regular if there is a regular expression (this is the same as an iterative specification involving atoms, $+$, $\times$ and SEQ() but in different notation), which generates the words of the language. But there is no restriction on whether they are generated uniquely. Such a non-unique specification wouldn't give the correct combinatorial class: multiple copies of some words would appear, and so in particular the generating function would be wrong.

However, it turns out that for any regular language in the sense of computer science, there is always a specification which uniquely generates it. The proof is nontrivial and can involve blowing up the size

of the specification exponentially. See Flajolet and Sedgewick, *Analytic Combinatorics*, Cambridge (2009), Appendix A8, `http://algo.inria.fr/flajolet/Publications/book.pdf`.

A nice result about regular languages is...

Theorem. *Any regular language has a rational ogf.*

You can prove this without too much difficulty. A harder question is the inverse: For which rational functions $Q(z)$ can you derive a regular specification whose generating function is precisely $Q(z)$?

2 Integer compositions $\mathcal{C}$

Another nice class of regular examples come from integer compositions.

Definition. A composition of an integer n is a sequence $(x_1, \ldots, x_k)$ of positive integers so that $n = x_1 + \cdots + x_k$.

For example, there are 8 compositions of 4:

$$\mathcal{C}_4 = \{1+1+1+1,\ 2+1+1,\ 1+2+1,\ 1+1+2,\ 2+2,\ 3+1,\ 1+3,\ 4\}$$

Let us determine a specification for integer compositions. So — let us start by treating natural numbers as a sequence of units.

We can think of the bijection

$$\begin{aligned} 1 &\leftrightarrow \circ \\ 2 &\leftrightarrow \circ\!-\!\circ \\ 3 &\leftrightarrow \circ\!-\!\circ\!-\!\circ \\ 4 &\leftrightarrow \circ\!-\!\circ\!-\!\circ\!-\!\circ \end{aligned}$$

Thus

$$\begin{aligned} \mathbb{N} = \mathcal{I} &\cong \textsc{Seq}_{\geq 1}(\{\circ\}) \\ I(z) &= \frac{z}{1-z} \end{aligned}$$

Then compositions are simply a sequence of natural numbers

$$\begin{aligned} 1+1+1+1 &\leftrightarrow (\circ, \circ, \circ, \circ) \\ 2+1+1 &\leftrightarrow (\circ\!-\!\circ, \circ, \circ) \\ 1+2+1 &\leftrightarrow (\circ, \circ\!-\!\circ, \circ) \\ 1+1+2 &\leftrightarrow (\circ, \circ, \circ\!-\!\circ) \\ &\vdots \\ 4 &\leftrightarrow (\circ\!-\!\circ\!-\!\circ\!-\!\circ) \end{aligned}$$

Thus, the specification is:

$$\begin{aligned} \mathcal{C} &= \textsc{Seq}(\mathcal{I}) \\ C(z) &= \frac{1}{1-I(z)} = \frac{1}{1-\frac{z}{1-z}} = \frac{1-z}{1-2z}. \end{aligned}$$

This can easily be expanded

$$C(z) = \sum_{n\geq 0} (2^n - 2^{n-1}) z^n = \sum_{n\geq 0} 2^{n-1} z^n.$$

Exercise. Find a simple combinatorial argument for this formula

Now we are well placed to consider some variants: What if the largest part you want to consider is 3? This removes the composition 4 from the above list of compositions of 4, for example. Well, it suffices to restrict the class of integers that we put into a composition.

What if we want compositions with at most k parts? We fiddle with the allowable sequence length.

Compositions		
Type	Spec	ogf
all	$\textsc{Seq}(\textsc{Seq}_{\geq 1}(\mathcal{Z}))$	$\frac{1}{1-\frac{z}{1-z}}$
parts $\leq r$	$\textsc{Seq}(\textsc{Seq}_{1\ldots r}(\mathcal{Z}))$	$\frac{1}{1-\frac{z-z^{r+1}}{1-z}}$
k parts	$\textsc{Seq}_{=k}(\textsc{Seq}_{\geq 1}(\mathcal{Z}))$	$\left(\frac{z}{1-z}\right)^k$

3 Some computer explorations: Playing with specifications

We can use built-in functionality in Maple in order to play around with objects that can be defined by these specifications. The package is `combstruct`, and it allows the user to see the start of the generating function (`gfseries`), to try to find an explicit generating function (`gfsolve`)

3.1 Binary words and variants

Maple

```
> with (combstruct):
> BINWORDS:= {W=Sequence(Union(Z1, Z0)), Z1=Atom, Z2=Atom}:
> gfseries(BINWORDS, unlabelled, z);
   table( [( Z1(z) ) = series(z,z), ( W(z) )
        =series(1+2*z+4*z^2+8*z^3+16*z^4+32*z^5+O(z^6),z,6), ( Z0(z) ) =series(z,z) ] )
> gfsolve(BINWORDS, unlabelled, z);
   {W(z) = -1/(-1+2*z), Z0(z) = z, Z1(z) = z}
```

Remark that `gfseries` outputs a table. This means we can access it directly for more succint output. We can also re-set the number of terms computed in the series command by modifying the `Order` parameter. We can also determine a single coefficient, with `coeff`.

Maple

```
> CYCLICWORDS:={W=Cycle(Union(Z1, Z0)), Z1=Atom, Z0=Atom}:
> gfseries(CYCLICWORDS, unlabelled, z)[W(z)];
   series(2*z+3*z^2+4*z^3+6*z^4+8*z^5+O(z^6),z,6)
> gfsolve(CYCLICWORDS, unlabelled, z);
   {W(z) = Sum(numtheory:-phi(j[1])*ln(-1/(-1+2*z^j[1]))/j[1], j[1] =1 .. infinity),
    Z0(z) = z, Z1(z) = z}

> Order:= 20:
> Wser:= gfseries(CYCLICWORDS, unlabelled, z)[W(z)]:
> coeff(Wser, z, 18);
14602
```

There are 14602 cyclic binary words of length 18.

Maple

```
> COMPS:={C=Sequence(Sequence(Z, card >0))}:
> gfsolve(COMPS, unlabelled, z);

> Order := 20:
> C_ser:=gfseries(COMPS, unlabelled, z)[C(z)]:
```

```
##-------  Compositions with at least five parts
> COMPS_5:={C=Sequence(Sequence(Z, card >0), card>=5)}:
> Cser_5:= gfseries(COMPS_5, unlabelled, z)[C(z)]:

##------- Proportion of compositions of size 18 with at least 5 parts
> coeff(Cser_5, z, 18) / coeff(Cser, z, 18);

> evalf(%);
```

Exercise. Find the number of partitions with at most 5 parts of size 25. Find the number of partitions of size 25 in which each part is at most 5.