Iterative classes

Contents

1 Regular specifications

1.1 Recall: admissible combinatorial specification

Definition. A **admissible 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 expression Φ_i is an admissible construction referring to the neutral class $\mathcal{E}$ some atomic classes $\mathcal{Z}_i$ and the classes $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(r)}$. If the graph of dependencies among these classes is acyclic, then the specification is **iterative**, otherwise it is **recursive**.

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

1.2 Regular specifications

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

Definition. An iterative specification (no recursion) that only involves atoms, the neutral element, combinatorial sums, cartesian products, and sequence constructions is said to be a **regular specification**. A subset of words from a finite alphabet is called a **language**. 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 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*[1] which generates the words of the language. But there is no restriction on whether words in the language are generated uniquely. Such a non-unique specification would not give the correct combinatorial class – multiple copies of some words would appear. In particular, the generating function obtained as in Lecture 4 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 that their OGF is always nice.

[1] This is the same as an iterative specification involving atoms, $+$, $\times$ and SEQ() but in different notation

Theorem. *Every regular language $\mathcal{L}$ has a rational OGF. That is, $L(z) = \frac{P(z)}{Q(z)}$ where $P(z)$ and $Q(z)$ are polynomials.*

Exercise. Prove this theorem.

A harder question is the inverse: For which rational functions $L(z) = \frac{P(z)}{Q(z)}$ is there a regular language whose generating function is precisely $L(z)$?

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 $(x_1, \dots, x_k)$ 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. We start by treating natural numbers as a sequence of a single atom "$\circ$".

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,

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

A composition is 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 direct combinatorial argument showing that there are exactly 2^{n-1} compositions of n.

Now we are well placed to consider some variants: What if want all the parts to have size at most r? Here we only need to replace the class of positive integers $\textsc{Seq}_{\geq 1}(\mathcal{Z})$ with $\textsc{Seq}_{1\ldots r}(\mathcal{Z})$.

If we want to count compositions with at most k parts, then restrict to sequences of at most k integers.

Here are a few examples of special integer compositions. We find their ogf's by consulting the table at the end of Section 2.6 of Lecture 4.

Type of integer composition	Specification	OGF
all compositions	$\textsc{Seq}(\textsc{Seq}_{\geq 1}(\mathcal{Z}))$	$\dfrac{1}{1-\frac{z}{1-z}}$
with all parts $\leq r$	$\textsc{Seq}(\textsc{Seq}_{1\ldots r}(\mathcal{Z}))$	$\dfrac{1}{1-\frac{z-z^{r+2}}{1-z}}$
with exactly k parts	$\textsc{Seq}_{=k}(\textsc{Seq}_{\geq 1}(\mathcal{Z}))$	$\left(\dfrac{z}{1-z}\right)^k$
with exactly k parts, each $\leq r$	$\textsc{Seq}_{=k}(\textsc{Seq}_{1\ldots r}(\mathcal{Z}))$	$\left(\dfrac{z-z^{r+1}}{1-z}\right)^k$
with $\leq k$ parts and no part equals 3	$\textsc{Seq}_{\leq k}(\textsc{Seq}_{\geq 1}(\mathcal{Z})-\mathcal{Z}^3)$	$\dfrac{1-\left(\frac{z}{1-z}-z^3\right)^{k+1}}{1-\left(\frac{z}{1-z}-z^3\right)}$
with an odd number of parts and each part equals 2 or 3	$\textsc{Seq}_{\text{odd}}(\mathcal{Z}^2+\mathcal{Z}^3)$	$\dfrac{z^2+z^3}{1-(z^2+z^3)^2}$

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

```
[fontsize=\small,fontfamily=courier,fontshape=tt,frame=single,label=\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 succinct 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`.

In the following example, we count the circular arrangements of n zeros and ones, where two necklaces are the same if one can be rotated to equal the other.

```
[fontsize=\small,fontfamily=courier,fontshape=tt,frame=single,label=\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.

```
[fontsize=\small,fontfamily=courier,fontshape=tt,frame=single,label=\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 compositions with at most 5 parts of the integer 25. Find the number of compositions of 25 in which each part is at most 5.