CO 330 LECTURE 26 SUMMARY

FALL 2026

Summary

We continued discussing recursive random generation.

Now suppose $\mathcal{A} = \mathcal{B} \times \mathcal{C}$ and we want to generate a random element of $\mathcal{A}_n$. The probability that an element of $\mathcal{A}_n$ has its $\mathcal{B}$ part of size k is

$$\frac{b_k c_{n-k}}{a_n}$$

so to generate uniformly, imagine stacking together all those chunks for $k = 0$, $k = 1$, $k = 2$, etc. which together fill up $(0, 1)$. Then we can choose a random number in $(0, 1)$ and we just need to find which k's chunk it belongs to. This is done in the following:

```
def genA=BxC
  input n
  x = rand()
  k = 0
  s = b_0*c_n/a_n
  while x > s
    k = k+1
    s = s+b_k*c_{n-k}/a_n
  return (genB(k), genC(n-k))
```

Finally what about Seq? We can always rewrite Seq away. This is because $\mathcal{A} = \text{Seq}(\mathcal{B})$ is equivalent to $\mathcal{A} = \mathcal{E} \cup \mathcal{B} \times \mathcal{A}$, see midterm 1 for details. Note that inside a larger specification you'll probably need to introduce a new class to represent each Seq and so you'll end up with more equations than you started with. Also this makes the specification recursive even if it wasn't originally recursive.

We looked at the code from last time and also the code from enumeration.ca.

Now what about labelled classes. For recursive random generation $\mathcal{E}$, $\mathcal{Z}$, and $\cup$ work as before, but we need to think about labelled product. Specifically we need to weight with some binomial coefficients in order take into to account the fact that if $\mathcal{A} = \mathcal{B} \star \mathcal{C}$ then the probability of an element of $\mathcal{A}_n$ has its $\mathcal{B}$ part of size k is

$$\binom{n}{k} \frac{b_k c_{n-k}}{a_n}$$

What about the labels themselves? We can carry them along, but then we need the label set as input as below

```
def genA=B*C
  input n, S
  x = rand()
```

```
  k = 0
  s = binom(n,0)*b_0*c_n/a_n
  while x > s
    k = k+1
    s = s+binom(n,k)*b_k*c_{n-k}/a_n
  T = random subset of S of size k
  return (genB(k, T), genC(n-k, S-T))
```

Alternately, we canjust generate the shape and impose a labelled afterwards because it is easy to generate a random permutation uniformly. However, this doesn't just mean we should use the unlabelled generator and then apply a random permutation to label the output because that won't be uniform among labelled objects. To illustrate the problem we considered rooted trees of size 3. There are two shapes, so from the unlabelled perspective we should generate each with probability 1/2. From the labelled perspective there are 9 labelled rooted trees of size 3. 6 of them have the stick shape and 3 have the branched shape. So if we are generating shapes first and then labelled we need to generate the stick twice as often as the branched one in order to obtain a labelled object uniformly in the end. We do this by keeping the binomial coefficients in the above algorithm, but forgetting the label sets. Specifically:

```
def genA=B*C (for labelling afterwards)
  input n
  x = rand()
  k = 0
  s = binom(n,0)*b_0*c_n/a_n
  while x > s
    k = k+1
    s = s+binom(n,k)*b_k*c_{n-k}/a_n
  return (genB(k), genC(n-k))
```

Next time

Next time we'll truly be on to Boltzmann samplers.

References

- https://enumeration.ca/asymptotics/random-generation/