CO 330 LECTURE 27 SUMMARY

FALL 2026

Summary

We discussed how to actually build a Boltzmann sampler. We will again build it recursively from a specification.

Consider the class $\mathcal{E}$, then no matter what x is ϵ is generated with probability $\frac{x^0}{E(x)} = \frac{1}{1} = 1$. So

```
def BoltzmannE
  input x
  return E
```

Similarly for $\mathcal{Z}$ the single atom is generated with probability $\frac{x^1}{Z(x)} = \frac{x}{x} = 1$. So

```
def BoltzmannZ
  input x
  return Z
```

Now suppose $\mathcal{A} = \mathcal{B} \cup \mathcal{C}$ with $\mathcal{B} \cap \mathcal{C} = \emptyset$. What is the probability that $a \in \mathcal{A}$ came from $\mathcal{B}$? Just sum the possibilities: $P_x(\mathcal{B}) = \sum_{b \in \mathcal{B}} x^{|b|}/A(x) = B(x)/A(x)$. So

```
def BoltzmannA=BcupC
  input x
  u = rand()
  if u < B(x)/A(x)
    return Boltzmann B
  else
    return Boltzmann C
```

Last time we left off without defining the Boltzmann sampler for products. Suppose $\mathcal{A} = \mathcal{B} \times \mathcal{C}$. Take $a = (b, c) \in \mathcal{A}$. Then a should be generated with probability

$$\frac{x^{|a|}}{A(x)} = \frac{x^{|b|+|c|}}{B(x)C(x)} = \frac{x^{|b|}}{B(x)} \frac{x^{|c|}}{C(x)}$$

so the two components are independent. This is simpler and faster compared to pure recursive random generation and we get

```
def BoltzmannA=BxC
  input x
  return(BoltzmannB(x), BoltzmannC(x))
```

We tried our running binary tree example and it was a bit disappointing; even though the expected value is correct we get a lot of tiny trees and then occaisonally a huge one. The problem is the distribution of sizes. Assignment 7 has a question that's meant to illustrate this to you. It will take us a few more days to get to how to solve the problem. For now, I told you that some classes have this problem and some don't (or not as much), and that

we will fix it using pointing. However, first we will explain how to do Boltzmann generation for more operations (partly because we'll want them when discussing the fix, and partly so you're in good shape for assignment 7)

So, let's think about SEQ. As we did for recursive random generation we could rewrite specifications to not use SEQ, but for Boltzmann generators there is also a direct way.

Say to generate an element of $\textsc{Seq}(\mathcal{B})$ we first pick k and then generate an element of $\mathcal{B}^k$. The second part is easy, we just saw the components are independent so just make k calls to `BoltzmannB`. The question is with what probability each k should occur. We need

$$P_x((b_1, b_2, \ldots, b_k)) = \frac{x^{|b_1|+\cdots+|b_k|}}{A(x)} = \frac{x^{|b_1|}}{B(x)} \cdots \frac{x^{|b_1|}}{B(x)} \left(B(x)^k(1 - B(x))\right)$$

so we should choose k with probability $(1 - B(x))B(x)^k$. This is a geometric distribution.

Definition 1. *The random variable X is geometrically distributed if $P(X = k) = (1 - \lambda)\lambda^k$ for some $0 < \lambda \leq 1$.*

How do we know $B(x) \leq 1$? We know that $x < \rho$ where ρ is the radius of convergence of $A(x) = 1/(1 - B(x))$ but $B(x)$ is increasing as x increases so when $B(x) = 1$ we get a singularity. So either this singularity is ρ or ρ is smaller. Either way $B(x) \leq 1$.

Taking this together we get

```
def BoltzmannA=Seq(B)
  input x
  k = geometric_rand(B(x))
  return(BoltzmannB(x), ..., BoltzmannB(x)) (k times)
```

NEXT TIME

Next time we'll think about labelled operations, how to make `geometric_rand` and then get back to fixing our disappointing generator.

REFERENCES

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