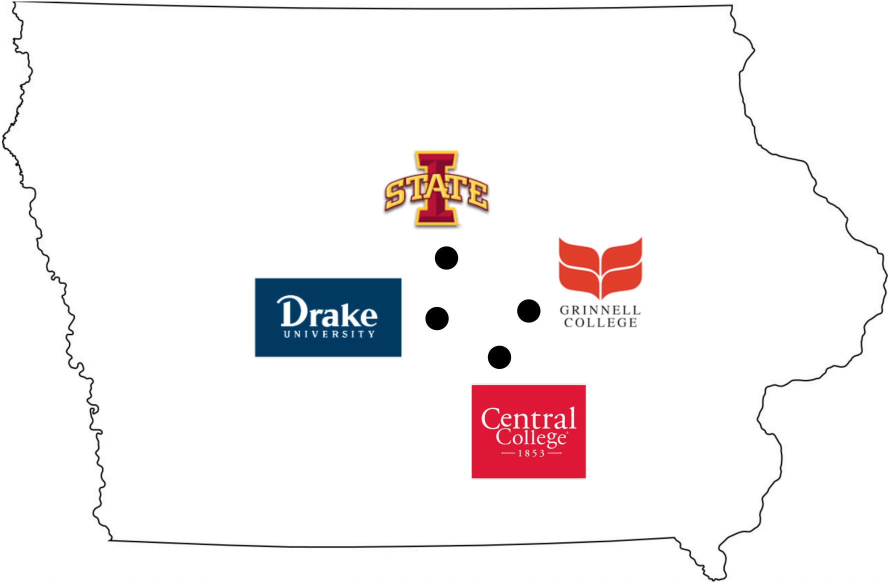
Randomness Extraction from a Computability-Theoretic Point of View

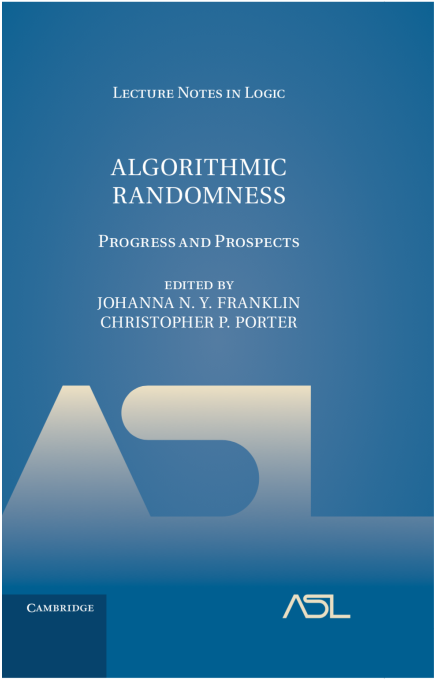
Christopher P. Porter
Drake University

Joint work with Doug Cenzer

Online Logic Seminar
May 14, 2020

ICICL
Iowa Colloquium on Information, Complexity, and Logic




LECTURE NOTES IN LOGIC
ALGORITHMIC RANDOMNESS
PROGRESS AND PROSPECTS
EDITED BY
JOHANNA N. Y. FRANKLIN
CHRISTOPHER P. PORTER
CAMBRIDGE
ASL

Motivation: Von Neumann's Trick

Suppose you have a biased coin, i.e. a coin such that

$$\mathbb{P}(H) = p$$

and

$$\mathbb{P}(T) = 1 - p$$

for some $p \in (0, \frac{1}{2})$, and you would like to use it to simulate a fair coin.

Von Neumann discovered a clever trick for doing so.

Von Neumann's Trick

Given a string such as:

010010111101011101000001101011110101111001

Von Neumann's Trick

Given a string such as:

010010111101011101000001101011110101111001

Step 1: Split the string into blocks of two.

Von Neumann's Trick

Given a string such as:

01 00 10 11 11 01 01 11 01 00 00 01 10 10 11 11 01 01 11 10 01

Step 1: Split the string into blocks of two.

Von Neumann's Trick

Given a string such as:

01 00 10 11 11 01 01 11 01 00 00 01 10 10 11 11 01 01 11 10 01

Step 1: Split the string into blocks of two.

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 11 11 01 01 11 01 00 00 01 10 10 11 11 01 01 11 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** 11 01 01 11 01 00 00 01 10 10 11 11 01 01 11 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** ** 01 01 11 01 00 00 01 10 10 11 11 01 01 11 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** ** 01 01 ** 01 00 00 01 10 10 11 11 01 01 11 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** ** 01 01 ** 01 ** 00 01 10 10 11 11 01 01 11 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** ** 01 01 ** 01 ** ** 01 10 10 11 11 01 01 11 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** ** 01 01 ** 01 ** ** 01 10 10 ** 11 01 01 11 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** ** 01 01 ** 01 ** ** 01 10 10 ** ** 01 01 11 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** ** 01 01 ** 01 ** ** 01 10 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Von Neumann's Trick

Given a string such as:

01 ** 10 ** ** 01 01 ** 01 ** ** 01 10 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 10 ** ** 01 01 ** 01 ** ** 01 10 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 01 01 ** 01 ** ** 01 10 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 01 ** 01 ** ** 01 10 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 01 ** ** 01 10 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 01 10 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 0 10 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 0 1 10 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 0 1 1 ** ** 01 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 0 1 1 ** ** 0 01 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 0 1 1 ** ** 0 0 ** 10 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 0 1 1 ** ** 0 0 ** 1 01

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 0 1 1 ** ** 0 0 ** 1 0

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

Von Neumann's Trick

Given a string such as:

0 ** 1 ** ** 0 0 ** 0 ** ** 0 1 1 ** ** 0 0 ** 1 0

Step 1: Split the string into blocks of two

Step 2: Delete all instances of 00 and 11

Step 3: Replace all instances of 01 with 0 and all instances of 10 with 1.

The resulting string is 010000110010.

Formally

Von Neumann's trick gives us a monotone function $\phi : 2^{<\omega} \to 2^{<\omega}$ satisfying

- $\phi(\sigma 00) = \phi(\sigma 11) = \phi(\sigma)$,
- $\phi(\sigma 01) = \phi(\sigma)0$, and
- $\phi(\sigma 10) = \phi(\sigma)1$

for every $\sigma \in 2^{<\omega}$ of even length.

Why does this work?

Recall we are given a biased coin such that for every $\sigma \in 2^{<\omega}$,

$$\mathbb{P}(\ \sigma 0 \mid \sigma\ ) = p \ \text{ and } \ \mathbb{P}(\ \sigma 1 \mid \sigma\ ) = 1 - p.$$

Key Observation: for every $\sigma \in 2^{<\omega}$,

$$\mathbb{P}(\ \sigma 00 \mid \sigma\ ) = p^2, \quad \mathbb{P}(\ \sigma 11 \mid \sigma\ ) = (1-p)^2,$$

and

$$\mathbb{P}(\ \sigma 01 \mid \sigma\ ) = \mathbb{P}(\ \sigma 10 \mid \sigma\ ) = p(1-p).$$

It thus follows that

$$\mathbb{P}\Big(\ \phi(\sigma)0 \ \Big|\ \phi(\sigma)\ \Big) = \mathbb{P}\Big(\ \phi(\sigma)1 \ \Big|\ \phi(\sigma)\ \Big) = \frac{1}{2}.$$

How efficient is Von Neumann's trick?

On average, how many biased bits are required to extract one unbiased bit?

The answer depends on how biased the coin is.

Given a $(p, 1-p)$-coin, on average, we will need $\frac{1}{p(1-p)}$ biased bits to extract a single unbiased bit.

Peres' refinement

In "Iterating Von Neumann's Procedure for Extracting Random Bits" (1992), Yuval Peres studies a sequence of generalizations of von Neumann's trick obtained by iterating von Neumann's procedure.

For each of these procedures, Peres calculates the associated extraction rate.

Given a monotone function $\phi : 2^{<\omega} \to 2^{<\omega}$, the extraction rate of ϕ with respect to the bias p is defined to be

$$\limsup_{n\to\infty} \frac{E(|\phi(x_1, x_2, \ldots, x_n)|)}{n}$$

where the bits x_i are independent and $(p, 1-p)$-distributed and E stands for expected value.

Peres' refinement (continued)

What Peres further shows is that the extraction rates of the various iterations of von Neumann's trick approaches the entropy of the underlying source,

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p).$$

Connections to computability theory?

A range of similar procedures and their corresponding extraction rates have been studied in the randomness extraction literature.

Given this general phenomenon of the extraction rates of various extraction procedures, what can we learn if we approach it from a computability-theoretic point of view?

In particular, what connections are there to algorithmic randomness?

Our methodology

1. For a range of Turing functionals, i.e., effective maps from 2^ω to 2^ω, we examine the corresponding notion of extraction rate.

2. For each such procedure, we investigate which notion of algorithmically random sequence is representative of the associated extraction rate.

Part 1: The Extraction Rates of Turing Functionals

Continuous Functionals

A continuous functional $\Phi : 2^\omega \to 2^\omega$ may be represented by a function $\phi : 2^{<\omega} \to 2^{<\omega}$ such that the following hold for all $\sigma \in 2^{<\omega}$:

(1) $\sigma_1 \prec \sigma_2$ implies $\phi(\sigma_1) \preceq \phi(\sigma_2)$;

(2) For every n, there exists m such that for all $\sigma \in \{0,1\}^m$, $|\phi(\sigma)| \geq n$;

(3) For all $X \in 2^\omega$, $\Phi(X) = \bigcup_n \phi(X \restriction n)$.

We call the function ϕ a representation of Φ.

We can define partial functionals if we do not require condition (2).

A partial or total functional Φ is a *Turing functional* if ϕ is computable.

Martin-Löf randomness

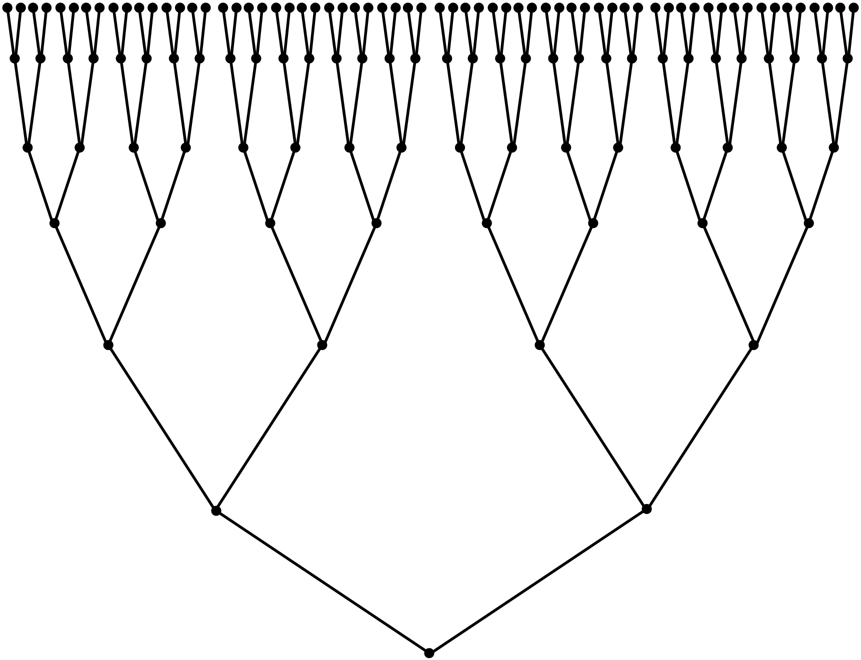
Definition

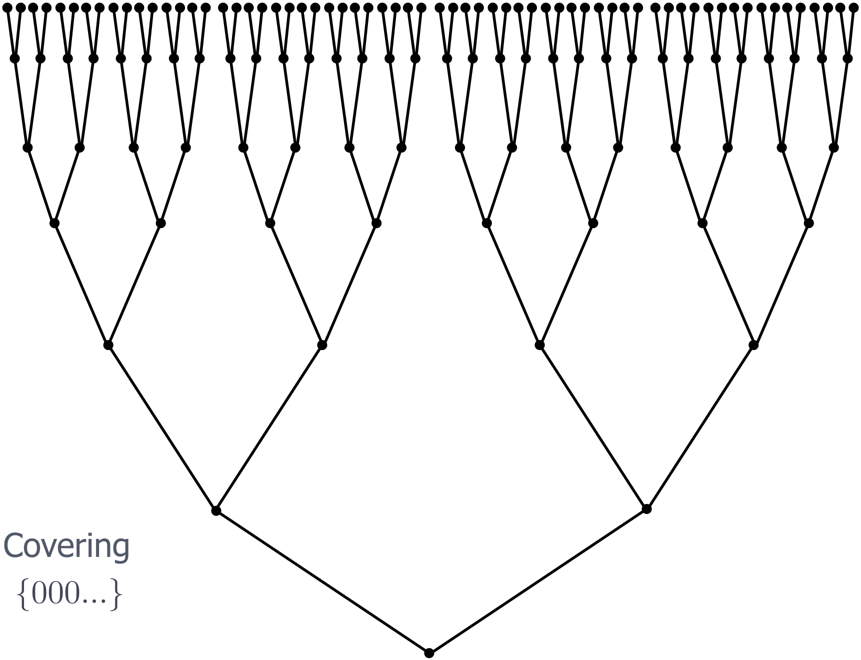
A *Martin-Löf test* is a uniformly Σ^0_1 sequence $(\mathcal{U}_i)_{i\in\omega}$ such that for each i,

$$\lambda(\mathcal{U}_i) \leq 2^{-i}.$$

A sequence $X \in 2^\omega$ *passes the Martin-Löf test* $(\mathcal{U}_i)_{i\in\omega}$ if $X \notin \bigcap_i \mathcal{U}_i$.

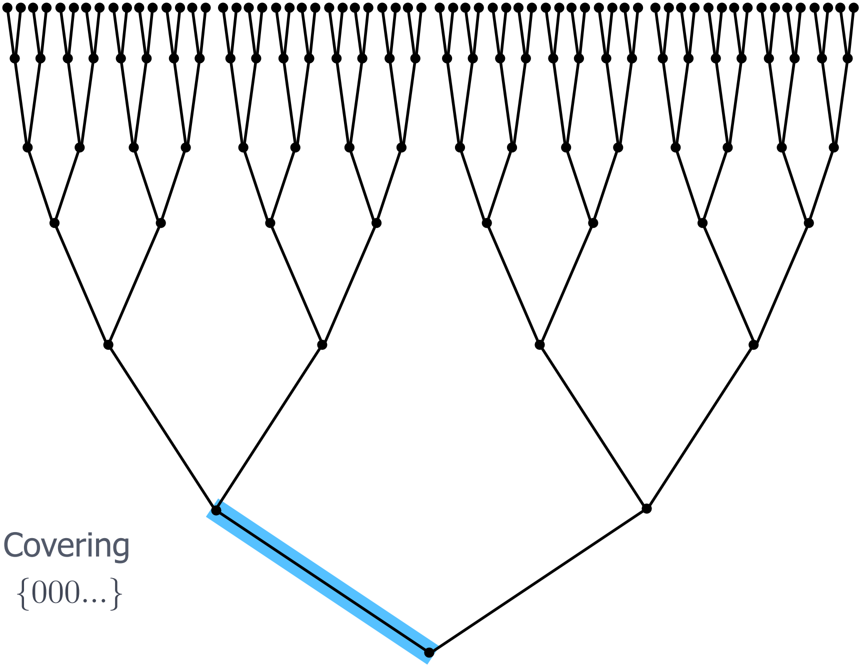
$X \in 2^\omega$ is *Martin-Löf random*, denoted $X \in \mathrm{MLR}$, if X passes *every* Martin-Löf test.





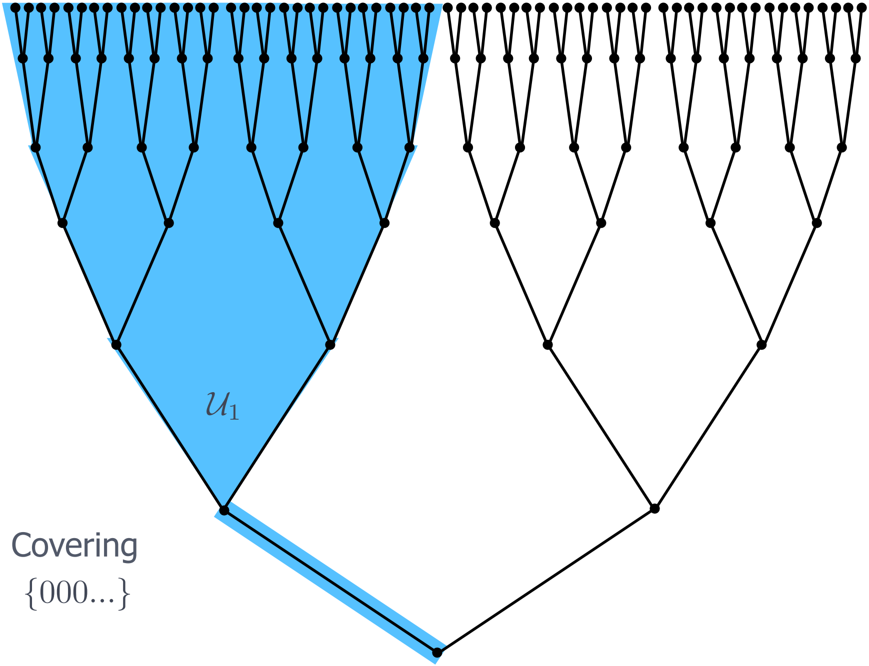
Covering

$\{000\ldots\}$



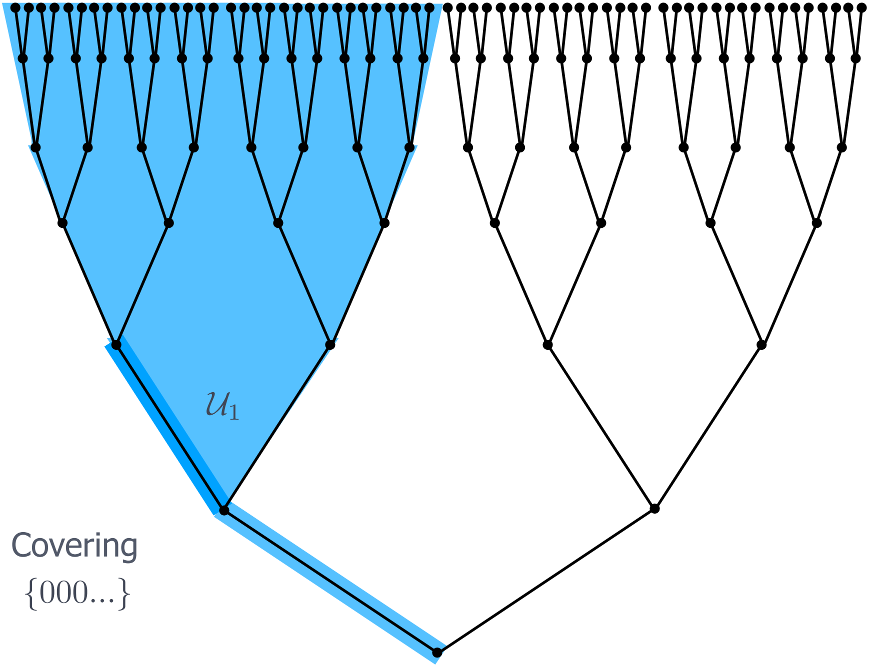
Covering

$\{000\ldots\}$



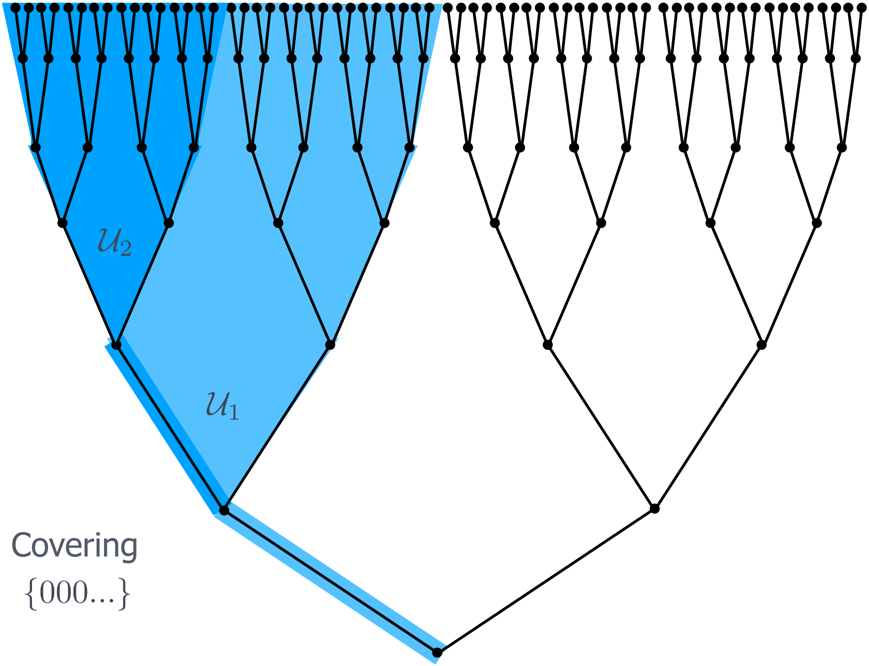
Covering

$\{000\ldots\}$



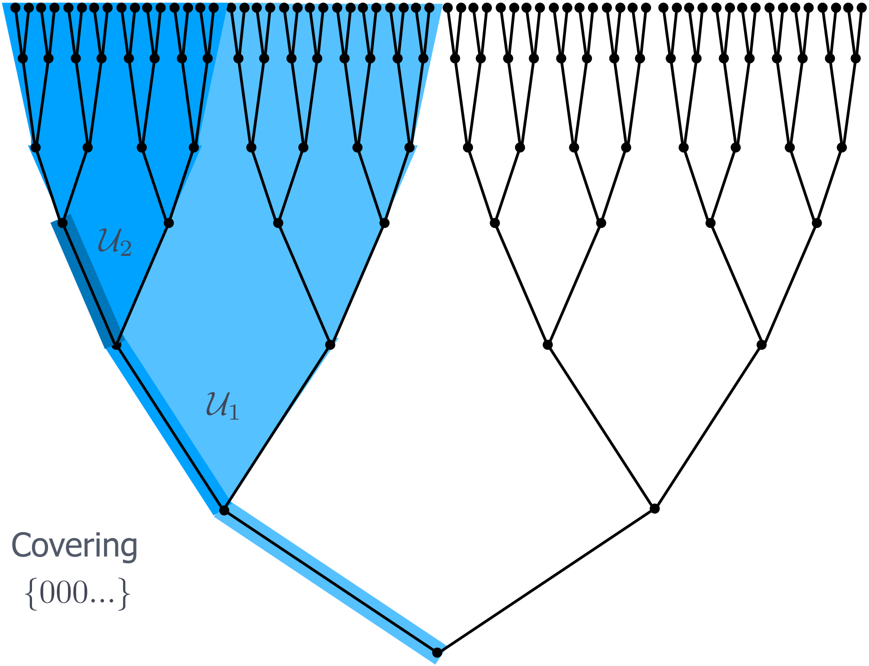
Covering

$\{000\ldots\}$



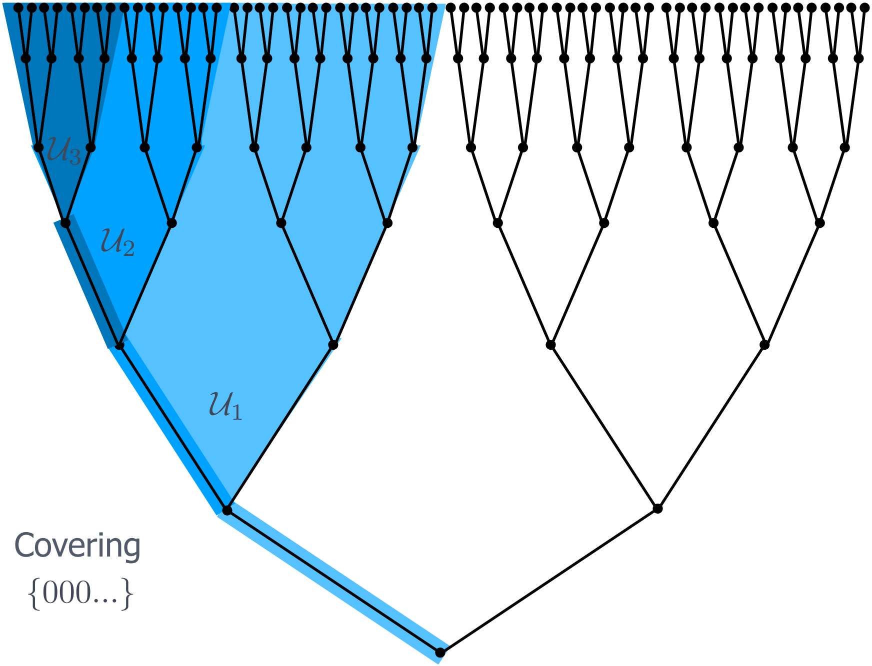
Covering

$\{000\ldots\}$



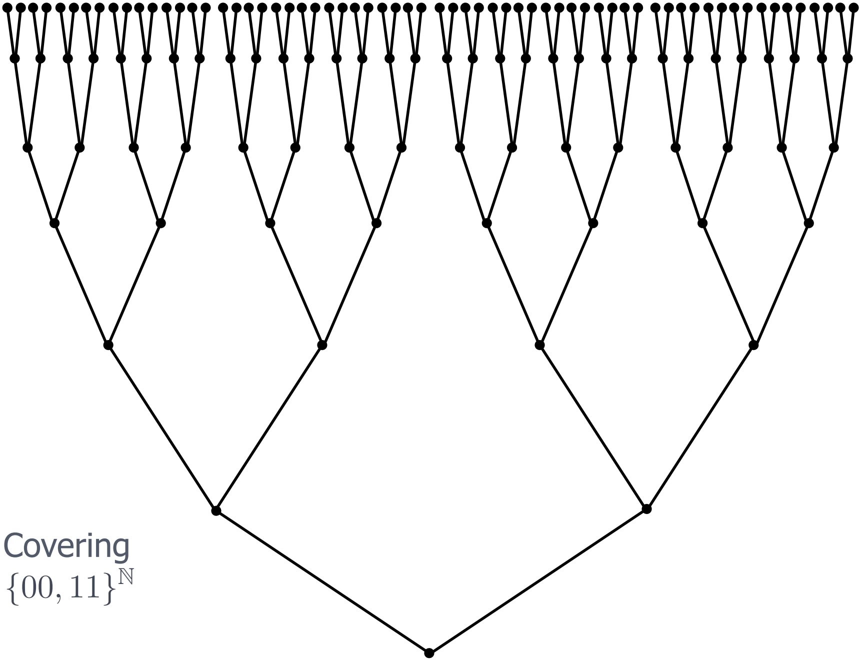
Covering

$\{000\ldots\}$

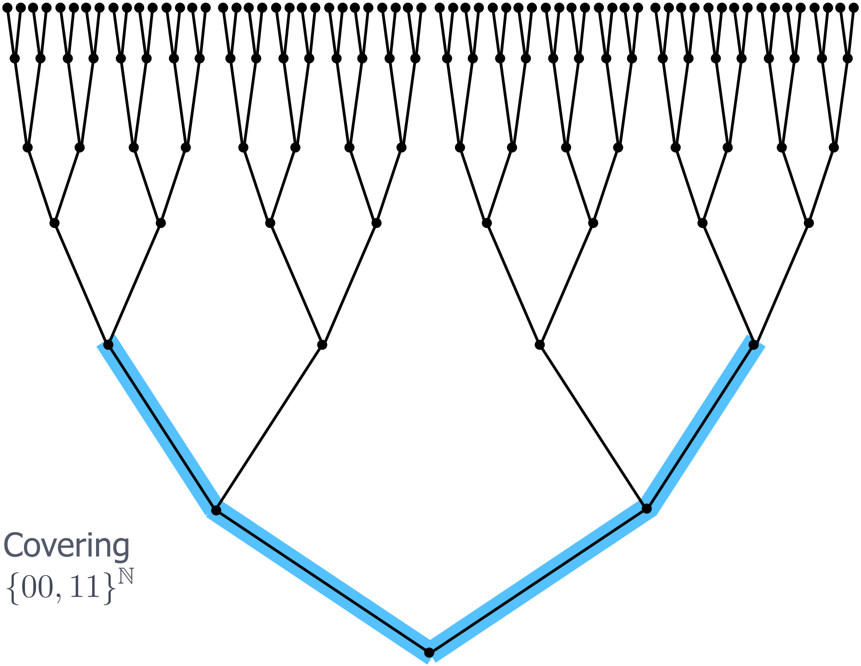


Covering

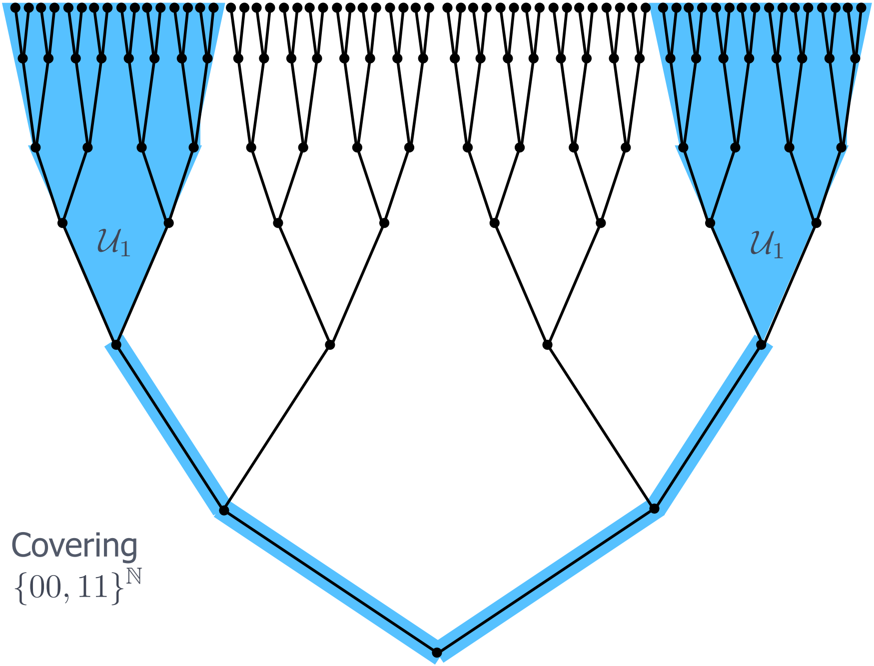
$\{000\ldots\}$



Covering
$\{00, 11\}^{\mathbb{N}}$

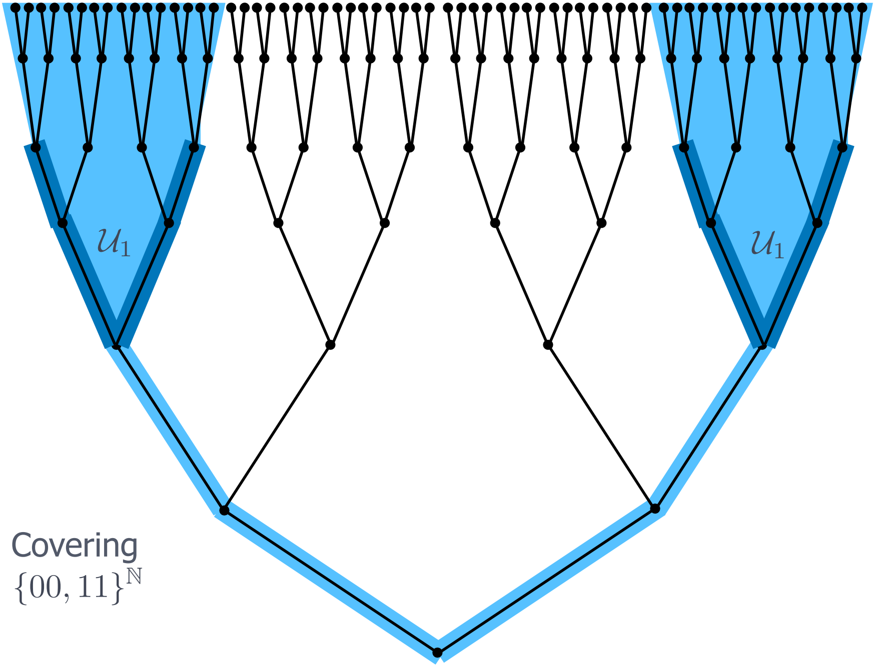


Covering
$\{00, 11\}^{\mathbb{N}}$

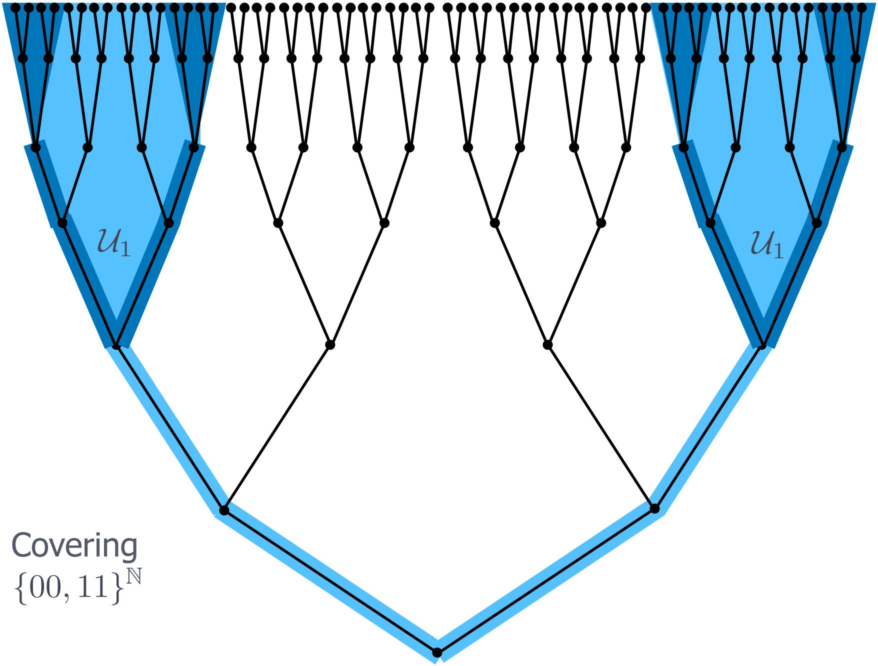


Covering

$\{00, 11\}^{\mathbb{N}}$



Covering
$\{00, 11\}^{\mathbb{N}}$



Covering
$\{00, 11\}^{\mathbb{N}}$

Schnorr Randomness

Definition

A *Schnorr test* is a Martin-Löf test $(\mathcal{U}_i)_{i\in\omega}$ such that for each i,

$$\lambda(\mathcal{U}_i) = 2^{-i}.$$

A sequence $X \in 2^\omega$ *passes the Schnorr test* $(\mathcal{U}_i)_{i\in\omega}$ if $X \notin \bigcap_i \mathcal{U}_i$.

$X \in 2^\omega$ is *Schnorr random*, denoted $X \in \mathrm{SR}$, if X passes *every* Schnorr test.

Fact: $\mathrm{MLR} \subsetneq \mathrm{SR}$.

Randomness with respect to non-uniform measures

We can also define Martin-Löf randomness and Schnorr randomness with respect to a non-uniform computable measure μ:

$$\begin{array}{ll} \mu\text{-Martin-Löf tests:} & \mu(\mathcal{U}_i) \leq 2^{-i} \\ \mu\text{-Schnorr tests:} & \mu(\mathcal{U}_i) = 2^{-i} \end{array}$$

Let MLR_μ denote the collection of μ-Martin-Löf random sequences.

Let SR_μ denote the collection of μ-Schnorr random sequences.

In general, we have $\mathrm{MLR}_\mu \subseteq \mathrm{SR}_\mu$.

Almost Total Turing Functionals

Let $\mathrm{dom}(\Phi) = \{X : \Phi(X) \in 2^\omega\}$.

Let μ be a computable probability measure on 2^ω.

A functional $\Phi : 2^\omega \to 2^\omega$ is *μ-almost total* if $\mu(dom(\Phi)) = 1$.

Lemma

A Turing functional Φ is μ-almost total if and only if $\mathrm{MLR}_\mu \subseteq \mathrm{dom}(\Phi)$.

The canonical representation of a functional

If ϕ is a representation of Φ with the property that $\phi(\sigma)$ is the longest common initial segment of all members of $\{\Phi(X) : \sigma \prec X\}$, we call ϕ the *canonical representation* of Φ.

In general, the canonical representation of a partial computable functional is computable in $\emptyset'$ and need not be computable.

Φ is *nowhere constant* if for any string σ, if $[\![\sigma]\!] \subseteq dom(\Phi)$, then Φ is not constant on $[\![\sigma]\!]$.

Proposition

If Φ is a total, nowhere constant Turing functional, then the canonical representation ϕ of Φ is computable.

Output-input ratios

Given a functional Φ with representation ϕ, we define the *ϕ-output/input ratio* of $\sigma \in 2^{<\omega}$ to be

$$OI_\phi(\sigma) = \frac{|\phi(\sigma)|}{|\sigma|}.$$

Moreover, we set $OI_\phi(X) = \lim_{n\to\infty} OI_\phi(X\upharpoonright n)$ if this limit exists.

The average output-input ratio

Let μ be a measure on 2^ω and let Φ be a μ-almost functional with representation ϕ.

For $[\![\sigma]\!] = \{X \in 2^\omega : \sigma \prec X\}$, we will hereafter write $\mu([\![\sigma]\!])$ as $\mu(\sigma)$.

The *average ϕ-output/input ratio* for strings of length n with respect to the measure μ is

$$Avg(\Phi, \mu, n) = \sum_{\sigma \in 2^n} \mu(\sigma) OI_\phi(\sigma) = \frac{1}{n} \sum_{\sigma \in 2^n} \mu(\sigma) |\phi(\sigma)|.$$

The extraction rate of a functional

The *μ-extraction rate* of Φ is

$$Rate(\Phi, \mu) = \limsup_{n \to \infty} Avg(\Phi, \mu, n).$$

Part 2: The Extraction Rates of Block Functionals

Block functionals

$\phi : 2^{<\omega} \to 2^{<\omega}$ is an *n-block map* if for any string $\sigma = \sigma_1{}^\frown \ldots {}^\frown \sigma_k{}^\frown \tau$,

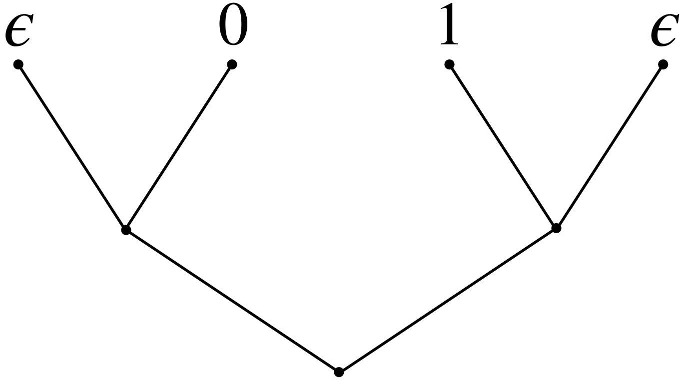
where $|\sigma_i| = n$ for $i = 1, \ldots, k$ and $|\tau| < n$,

$$\phi(\sigma) = \phi(\sigma_1)^\frown \ldots {}^\frown \phi(\sigma_k).$$

ϕ is *non-trivial* if $|\phi(\sigma)| > 0$ for some $\sigma \in 2^n$.

Φ is an *n-block functional* if it has an n-block representation.

Block maps have been studied by Peres, Elias, Pae-Il and others.


ϵ
0
1
ϵ

Bernoulli measures

For $p \in (0,1)$, the Bernoulli measure μ_p on 2^ω is given by

$$\mu(\sigma) = p^{\#_0(\sigma)}(1-p)^{\#_1(\sigma)}.$$

Given a Bernoulli measure μ on $(2^n)^\omega$, we can extend μ to a measure on 2^ω, which we call an *n-step Bernoulli measure* on 2^ω.

A measure μ is *positive* if $\mu(\sigma) > 0$ for all $\sigma \in 2^{<\omega}$.

Proposition

Suppose μ is a positive n-step Bernoulli measure on 2^ω and Φ is a non-trivial n-block functional Φ. Then Φ is μ-almost total.

The main results on block functionals

Theorem

Let μ be a positive n-step Bernoulli measure and Φ a non-trivial n-block functional. Then

$$Rate(\Phi, \mu) = Avg(\Phi, \mu, n).$$

Theorem

Let μ be a computable, positive n-step Bernoulli measure, and let $X \in 2^\omega$ be μ-Schnorr random. Then for every non-trivial n-block functional Φ with canonical representation ϕ,

$$\lim_{n \to \infty} \frac{|\phi(X \restriction n)|}{n} = Rate(\Phi, \mu).$$

A key ingredient

For our proof, we use the following effective version of Birkhoff's Ergodic Theorem:

Theorem (Franklin-Towsner)

Let μ be a computable measure on 2^ω and let $T : 2^\omega \to 2^\omega$ be a computable, μ-invariant, ergodic transformation. Then for any bounded computable function f and any μ-Schnorr random $X \in 2^\omega$,

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} f(T^i(X)) = \int f \, d\mu.$$

A bit of ergodic theory

A transformation $T : 2^\omega \to 2^\omega$ is *μ-invariant* if, for all $\tau \in 2^{<\omega}$,

$$\mu(\tau) = \mu(T^{-1}(\llbracket \tau \rrbracket)).$$

For example, the shift transformation $T(X) = (X(1), X(2), \dots)$ is μ-invariant with respect to any Bernoulli measure μ on 2^ω.

A μ-invariant transformation T is *ergodic* if for every μ-measurable set $\mathcal{A} \subseteq 2^\omega$ with $T^{-1}(\mathcal{A}) = \mathcal{A}$, $\mu(\mathcal{A}) = 0$ or $\mu(\mathcal{A}) = 1$.

We say that μ is ergodic if the shift is ergodic with respect to μ.

For an ergodic measure μ, we may define the *entropy* of μ as

$$h(\mu) = \lim_{n \to \infty} -\frac{1}{n} \sum_{|\sigma|=n} \mu(\sigma) \log \mu(\sigma).$$

Theorem (Franklin-Towsner)

Let μ be a computable measure on 2^ω and let $T : 2^\omega \to 2^\omega$ be a computable, μ-invariant, ergodic transformation. Then for any bounded computable function f and any μ-Schnorr random $X \in 2^\omega$,

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} f(T^i(X)) = \int f \, d\mu.$$

Theorem (Franklin-Towsner)

Let μ be a computable measure on 2^ω and let $T : 2^\omega \to 2^\omega$ be a computable, μ-invariant, ergodic transformation. Then for any bounded computable function f and any μ-Schnorr random $X \in 2^\omega$,

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} f(T^i(X)) = \int f \, d\mu.$$

- Let T be the n-shift operator, which is ergodic with respect to any n-step Bernoulli measure.

Theorem (Franklin-Towsner)

Let μ be a computable measure on 2^ω and let $T : 2^\omega \to 2^\omega$ be a computable, μ-invariant, ergodic transformation. Then for any bounded computable function f and any μ-Schnorr random $X \in 2^\omega$,

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} f(T^i(X)) = \int f \, d\mu.$$

- Let T be the n-shift operator, which is ergodic with respect to any n-step Bernoulli measure.
- Let $f(X) = \dfrac{|\phi(X\restriction n)|}{n}$.

Theorem (Franklin-Towsner)

Let μ be a computable measure on 2^ω and let $T : 2^\omega \to 2^\omega$ be a computable, μ-invariant, ergodic transformation. Then for any bounded computable function f and any μ-Schnorr random $X \in 2^\omega$,

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} f(T^i(X)) = \int f \, d\mu.$$

- Let T be the n-shift operator, which is ergodic with respect to any n-step Bernoulli measure.
- Let $f(X) = \dfrac{|\phi(X\restriction n)|}{n}$.
- Then $\int f \, d\mu = Avg(\Phi, \mu, n) = Rate(\Phi, \mu)$.

Theorem (Franklin-Towsner)

Let μ be a computable measure on 2^ω and let $T : 2^\omega \to 2^\omega$ be a computable, μ-invariant, ergodic transformation. Then for any bounded computable function f and any μ-Schnorr random $X \in 2^\omega$,

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} f(T^i(X)) = \int f \, d\mu.$$

- Let T be the n-shift operator, which is ergodic with respect to any n-step Bernoulli measure.
- Let $f(X) = \dfrac{|\phi(X\restriction n)|}{n}$.
- Then $\int f \, d\mu = Avg(\Phi, \mu, n) = Rate(\Phi, \mu)$.
- $\frac{1}{k}\sum_{i=0}^{k-1} f(T^i(X)) = \frac{1}{k}\sum_{i=0}^{k-1} \dfrac{|\phi(X\restriction_{[ni,n(i+1)]})|}{n} = \dfrac{|\phi(X\restriction nk)|}{nk}$

Part 3: The Extraction Rates of Functionals Induced by DDG-Trees

DDG-trees

Discrete Distribution Generating trees were defined by Knuth and Yao ("The Complexity of Non-Uniform Random Number Generation") in their study of using a fair coin to generate a biased distribution.

A *DDG-tree* is a tree $S \subseteq 2^{<\omega}$

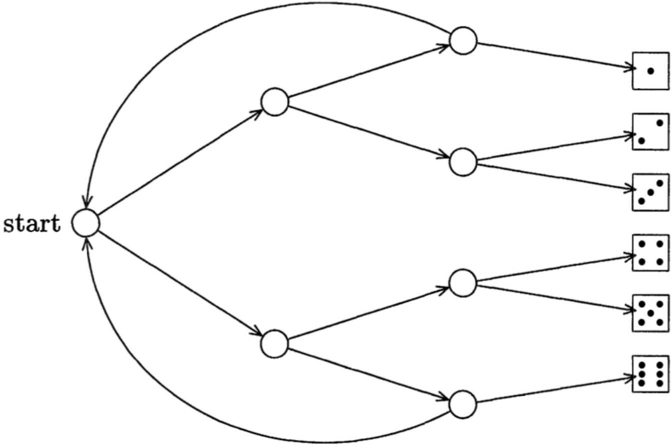
- with terminal nodes $D(S) \subseteq S$,
- together with a labelling function

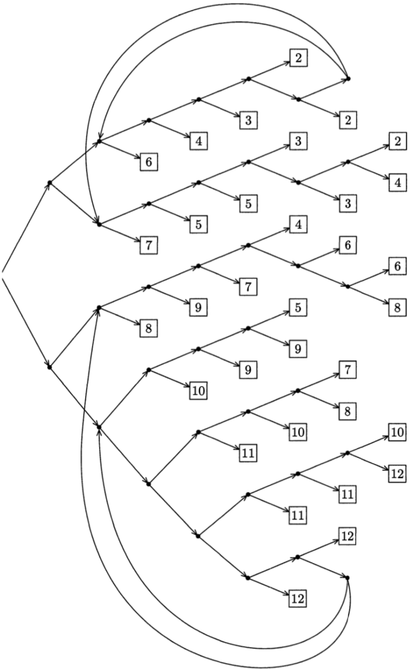
$$\ell_S : D(S) \to A = \{a_1, \ldots, a_k\}$$

- which induces a discrete probability distribution on A by setting

$$p_i = \sum_{\ell_S(\tau) = a_i} 2^{-|\tau|}.$$

We assume that the set $[S]$ of infinite paths through S has measure 0, so that $\sum_{i=1}^{k} p_i = 1$


start


2
2
3
4
3
2
6
4
5
3
5
7
4
6
7
6
9
8
8
5
9
9
10
7
8
10
10
11
12
11
11
12
12

Average running time of extraction

Knuth and Yao define the average running time of randomness extraction by a DDG-tree S to be

$$AvgRT(S) = \sum_{i \in \omega} i \cdot \lambda(\llbracket D(S) \cap 2^i \rrbracket).$$

That is, $AvgRT(S)$ is the average number of input bits needed to produce a single output bit.

The functional induced by a DDG-tree

Given a DDG-tree S, we can define a functional $\Phi_S : 2^\omega \to A^\omega$ by applying the Knuth-Yao procedure successively along initial segments of an input $X \in 2^\omega$.

Extraction rate of Φ_S

Theorem

Let $X \in 2^\omega$ be Schnorr random. Then for every computable DDG-tree S, we have

$$\lim_{n\to\infty} \frac{|\phi_S(X\restriction n)|}{n} = \frac{1}{AvgRT(S)}.$$

The proof of this results depends another effective version of Birkhoff's Ergodic Theorem due to Gács, Hoyrup and Rojas.

By integrating over the Schnorr random sequences, we have:

Corollary

$$Rate(\Phi_S, \lambda) = \frac{1}{AvgRT(S)}.$$

Part 4: The Extraction Rate of the Levin-Kautz Conversion Procedure

A theorem due to Levin and Kautz

Theorem (Levin/Kautz)

For every pair of computable measures μ and ν on 2^ω, there is an effective procedure that transforms every non-computable μ-Martin-Löf random sequence into a ν-Martin-Löf random sequence.

Question

Can we determine the rate at which ν-randomness can be extracted from a μ-random sequence?

Levin-Kautz conversion

We define for computable measures μ and ν, a μ-almost total functional $\Phi_{\mu\to\nu}$ that transforms μ-randomness into ν-randomness.

For non-computable $X \in \mathrm{MLR}_\mu$ and $Y \in 2^\omega$ such that $\Phi_{\mu\to\nu}(X) = Y$ (so that $Y \in \mathrm{MLR}_\nu$),

(i) $(\Phi_{\mu\to\nu} \circ \Phi_{\nu\to\mu})(X) = X$, and
(ii) $(\Phi_{\nu\to\mu} \circ \Phi_{\mu\to\nu})(Y) = Y$,

and thus $X \equiv_T Y$.

The extraction rate for Levin-Kautz conversion

A measure μ on 2^ω is *strongly positive* if there is some $\delta \in (0, \frac{1}{2})$ such that for every $\sigma \in 2^{<\omega}$, $\mu(\sigma 0 \mid \sigma) \in [\delta, 1-\delta]$.

Theorem

Let μ and ν be computable, shift-invariant, ergodic measures that are strongly positive. Then for every non-computable $X \in \mathrm{MLR}_\mu$,

$$OI_{\Phi_{\mu\to\nu}}(X) = \frac{h(\mu)}{h(\nu)}.$$

In particular, in the case that $\nu = \lambda$, we have $OI_{\Phi_{\mu\to\lambda}}(X) = h(\mu)$.

Effective Shannon-McMillan-Breiman Theorem

A key feature of our proof is the following result of Hoyrup's, which follows from yet another effective version of Birkhoff's ergodic theorem.

Theorem (Hoyrup)

Let μ be a computable shift-invariant ergodic measure on 2^ω. Then for every μ-Martin-Löf random sequence $X \in 2^\omega$,

$$\lim_{n\to\infty} \frac{K(X{\upharpoonright}n)}{n} = \lim_{n\to\infty} \frac{-\log \mu(X{\upharpoonright}n)}{n} = h(\mu).$$

A few open problems

Problem: Identify features of an almost total Turing functional that guarantee that its extraction rate is witnessed pointwise by sufficiently random inputs to the functional.

Problem: For each of the classes of functionals discussed here, determine the level of randomness necessary for a sequence to witness the extraction rate.

Thank you!