Coin Tossing, Randomness Extraction, and Algorithmic Randomness

Christopher P. Porter

UF GMA Colloquium

March 25, 2015

Motivating Question

Suppose you are given a biased coin, i.e. a coin such that

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

and

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

for some $p \neq 1/2$, and you are asked to use it to simulate a fair coin.

Can this be done?

The Plan of Attack

1. Some technicalities
2. Von Neumann's Trick
3. Generalizing the problem
4. Addressing the general problem with algorithmic randomness

A Bit of Notation

Hereafter, we will represent the event H by 0 and the event T by 1.

Let $2^{<\omega}$ denote the set of all finite binary strings.

Let 2^{ω} denote the set of all infinite binary sequences.

More Notation

Given $\sigma, \tau \in 2^{<\omega}$, $\sigma \preceq \tau$ means σ is an initial segment of τ.

Similarly, given $\sigma \in 2^{<\omega}$ and $X \in 2^{\omega}$, $\sigma \prec X$ means that σ is an initial segment of X.

Given $X \in 2^{\omega}$ and $n \in \omega$, $X{\upharpoonright}n$ denotes the initial segment of of length n.

A Few More Definitions

2^ω has a natural topology: the basic open sets are of the form:

$$[\![\sigma]\!] := \{X \in 2^\omega : \sigma \prec X\}$$

The Lebesgue measure λ on 2^ω is defined on basic open sets to be

$$\lambda([\![\sigma]\!]) = 2^{-|\sigma|}.$$

Later we will consider other probability measures on 2^ω.

Simulating a fair coin?

What does it mean to simulate a fair coin with a biased coin?

Roughly, we want to use our biased coin to generate a finite string (or even an infinite sequence!) that is indistinguishable from one produced by tossing a fair coin.

Simulating a fair coin?

What does it mean to simulate a fair coin with a biased coin?

Roughly, we want to use our biased coin to generate a finite string (or even an infinite sequence!) that is indistinguishable from one produced by tossing a fair coin.

This is not very helpful.

More precisely... (1)

Suppose we produce the string $\sigma \in 2^{<\omega}$ using our biased coin.

We want some function $\phi : 2^{<\omega} \to 2^{<\omega}$ that will convert σ into an unbiased string $\tau \in 2^{<\omega}$.

For any $\xi \in 2^{<\omega}$,

$$\mathbb{P}\big(\phi(\xi)^\frown 0 \mid \phi(\xi)\big) = \mathbb{P}\big(\phi(\xi)^\frown 1 \mid \phi(\xi)\big) = \frac{1}{2}.$$

More precisely… (2)

We also want ϕ to satisfy the following monotonicity property:

$$\sigma \preceq \sigma' \Rightarrow \phi(\sigma) \preceq \phi(\sigma')$$

Such a function should also be effectively computable: there should be an algorithm for computing the values of ϕ.

Von Neumann's Trick

Von Neumann's Trick

-

Von Neumann's Trick

01

Von Neumann's Trick

010

Von Neumann's Trick

0100

Von Neumann's Trick

01000

Von Neumann's Trick

010000

Von Neumann's Trick

0100000

Von Neumann's Trick

01000001

Von Neumann's Trick

010000010

Von Neumann's Trick

0100000100

Von Neumann's Trick

01000001001

Von Neumann's Trick

01000001001

Von Neumann's Trick

01000001001001010000

01010010000000110100

00000000111000100001

01010001010100100000

00001010011000001010

Von Neumann's Trick

01000001001001010000

01010010000000110100

00000000111000100001

01010001010100100000

00001010011000001010

Step 1: Split the string into blocks of length 2.

Von Neumann's Trick

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

Step 1: Split the string into blocks of length 2.

Von Neumann's Trick

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

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

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

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

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

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

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

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

01 ~~00~~ ~~00~~ 01 ~~00~~ 10 01 01 ~~00~~ 00

01 01 00 10 00 00 00 11 01 00

00 00 00 00 11 10 00 10 00 01

01 01 00 01 01 01 00 10 00 00

00 00 10 10 01 10 00 00 10 10

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

01 ~~00~~ ~~00~~ 01 ~~00~~ 10 01 01 ~~00~~ ~~00~~

01 01 00 10 00 00 00 11 01 00

00 00 00 00 11 10 00 10 00 01

01 01 00 01 01 01 00 10 00 00

00 00 10 10 01 10 00 00 10 10

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

01 ~~00~~ ~~00~~ 01 ~~00~~ 10 01 01 ~~00~~ ~~00~~

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

00 00 00 00 11 10 00 10 00 01

01 01 00 01 01 01 00 10 00 00

00 00 10 10 01 10 00 00 10 10

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

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

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

01 ~~00~~ ~~00~~ 01 ~~00~~ 10 01 01 ~~00~~ ~~00~~

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

00 00 00 00 11 10 00 10 00 01

01 01 00 01 01 01 00 10 00 00

00 00 10 10 01 10 00 00 10 10

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

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

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

01 ~~00~~ ~~00~~ 01 ~~00~~ 10 01 01 ~~00~~ ~~00~~

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

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

01 01 ~~00~~ 01 01 01 ~~00~~ 10 ~~00~~ ~~00~~

~~00~~ ~~00~~ 10 10 01 10 ~~00~~ ~~00~~ 10 10

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

01 01 10 01 01
01 01 10 01
10 10 01
01 01 01 01 01 10
10 10 10 10 10

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

01 01 10 01 01 01 01 10 01 10 10 01
01 01 01 01 01 10 10 10 01 10 10 10

Step 1: Split the string into blocks of length 2.

Step 2: Delete all instances of 00 and 11.

Von Neumann's Trick

01 01 10 01 01 01 01 10 01 10 10 01
01 01 01 01 01 10 10 10 01 10 10 10

Step 1: Split the string into blocks of length 2.

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

0 01 10 01 01 01 01 10 01 10 10 01
01 01 01 01 01 10 10 10 01 10 10 10

Step 1: Split the string into blocks of length 2.

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

0 0 10 01 01 01 01 10 01 10 10 01
01 01 01 01 01 10 10 10 01 10 10 10

Step 1: Split the string into blocks of length 2.

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

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

Step 1: Split the string into blocks of length 2.

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

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

Step 1: Split the string into blocks of length 2.

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

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

Step 1: Split the string into blocks of length 2.

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

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

Step 1: Split the string into blocks of length 2.

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

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

Step 1: Split the string into blocks of length 2.

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

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

Step 1: Split the string into blocks of length 2.

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

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

Step 1: Split the string into blocks of length 2.

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

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

01 01 01 01 01 10 10 10 01 10 10 10

Step 1: Split the string into blocks of length 2.

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

0 0 1 0 0 0 0 1 0 1 1 01

01 01 01 01 01 10 10 10 01 10 10 10

Step 1: Split the string into blocks of length 2.

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

0 0 1 0 0 0 0 1 0 1 1 0

01 01 01 01 01 10 10 10 01 10 10 10

Step 1: Split the string into blocks of length 2.

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

0 0 1 0 0 0 0 1 0 1 1 0
0 0 0 0 0 1 1 1 0 1 1 1

Step 1: Split the string into blocks of length 2.

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

0010000101100000001110111

Step 1: Split the string into blocks of length 2.

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.

Formally...

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

(1) $\phi(\sigma^\frown 00) = \phi(\sigma^\frown 11) = \phi(\sigma)$,

(2) $\phi(\sigma^\frown 01) = \phi(\sigma)^\frown 0$, and

(3) $\phi(\sigma^\frown 10) = \phi(\sigma)^\frown 1$

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

Why does this work? (1)

Recall: $\mathbb{P}(\sigma^\frown 0 \mid \sigma) = p$, $\mathbb{P}(\sigma^\frown 1 \mid \sigma) = 1 - p$

Key observation: For every $\sigma \in 2^{<\omega}$,

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

and

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

Why does this work? (2)

Key observation: For every $\sigma \in 2^{<\omega}$,

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

and

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

The conditional probability that the next bit output by ϕ is a 0, given that it has read either 01 or 10, is 1/2.

Changing things up (1)

The probability measure on 2^ω induced by a coin with bias p is called a **Bernoulli p-measure**.

Suppose now we are given a sequence of biased coins, and we are asked to simulate a fair coin by tossing each biased coin consecutively.

Changing things up (2)

The probability measure on 2^ω induced by such a sequence of coins is called a generalized Bernoulli measure, denoted $\mu_{\vec{p}}$, where $\vec{p} = (p_0, p_1, \ldots)$ and $p_i \in [0,1]$ for every $i \in \omega$.

Then for every $\sigma \in 2^{<\omega}$ of length n,

$$\mu_{\vec{p}}(\sigma^\frown 0 \mid \sigma) = p_n.$$

More changes (1)

What if we generalize even further, so that the sequence of coins we toss depends not only on the trial but also on the previous outcomes?

In this most general case, we're dealing with arbitrary probability measure on 2^ω.

More changes (2)

In these more general settings, von Neumann's trick won't work.

Is there a more general effective procedure for converting such biased random sequences into unbiased random sequences?

More changes (2)

In these more general settings, von Neumann's trick won't work.

Is there a more general effective procedure for converting such biased random sequences into unbiased random sequences?

Yes, if we recast the question in terms of algorithmic randomness.

Computable measures

In what follows, we will restrict our attention to computable probability measures on 2^ω.

A measure μ on 2^ω is **computable** if there is a computable function $\widehat{\mu} : 2^{<\omega} \times \omega \to \mathbb{Q}$ such that for every $\sigma \in 2^{<\omega}$ and every $i \in \omega$,

$$|\mu([\![\sigma]\!]) - \widehat{\mu}(\sigma, i)| \leq 2^{-i}.$$

Random Sequences

We will also restrict our attention to sufficiently random sequences.

How do we guarantee that a sequence is sufficiently random?

There are a number of ways to make the notion of random sequence precise.

The Law of Large Numbers

$X \in 2^\omega$ satisfies the law of large numbers if

$$\lim_{n\to\infty} \frac{\#_0(X\restriction n)}{n} = \frac{1}{2}$$

where $\#_0(\sigma)$ is the number of 0's in $\sigma \in 2^{<\omega}$.

The Law of Large Numbers

$X \in 2^\omega$ satisfies the law of large numbers if

$$\lim_{n\to\infty} \frac{\#_0(X\upharpoonright n)}{n} = \frac{1}{2}$$

where $\#_0(\sigma)$ is the number of 0's in $\sigma \in 2^{<\omega}$.

Not sufficient for randomness:

The Law of Large Numbers

$X \in 2^\omega$ satisfies the law of large numbers if

$$\lim_{n \to \infty} \frac{\#_0(X \upharpoonright n)}{n} = \frac{1}{2}$$

where $\#_0(\sigma)$ is the number of 0's in $\sigma \in 2^{<\omega}$.

Not sufficient for randomness:

010101010101010101010101...

Normal Sequences

For $\sigma, \tau \in 2^{<\omega}$, let $\#_\sigma(\tau)$ denote the number of occurrences of σ as a subword of τ.

$X \in 2^\omega$ is **normal** if for every $\sigma \in 2^{<\omega}$,

$$\lim_{n\to\infty} \frac{\#_\sigma(X\restriction n)}{n} = 2^{-|\sigma|}.$$

Normal Sequences

For $\sigma, \tau \in 2^{<\omega}$, let $\#_\sigma(\tau)$ denote the number of occurrences of σ as a subword of τ.

$X \in 2^\omega$ is **normal** if for every $\sigma \in 2^{<\omega}$,

$$\lim_{n \to \infty} \frac{\#_\sigma(X\restriction n)}{n} = 2^{-|\sigma|}.$$

Not sufficient for randomness:

Normal Sequences

For $\sigma, \tau \in 2^{<\omega}$, let $\#_\sigma(\tau)$ denote the number of occurrences of σ as a subword of τ.

$X \in 2^\omega$ is **normal** if for every $\sigma \in 2^{<\omega}$,

$$\lim_{n\to\infty} \frac{\#_\sigma(X\restriction n)}{n} = 2^{-|\sigma|}.$$

Not sufficient for randomness:

01000110110000010100 11...

Normal Sequences

For $\sigma, \tau \in 2^{<\omega}$, let $\#_\sigma(\tau)$ denote the number of occurrences of σ as a subword of τ.

$X \in 2^\omega$ is **normal** if for every $\sigma \in 2^{<\omega}$,

$$\lim_{n\to\infty} \frac{\#_\sigma(X\restriction n)}{n} = 2^{-|\sigma|}.$$

Not sufficient for randomness:

0 1 00 01 10 11 000 001 010 011...

Stochastic Sequences (1)

Let $\psi : 2^{<\omega} \to \{0,1\}$ be a computable function.

Stochastic Sequences (1)

Let $\psi : 2^{<\omega} \to \{0,1\}$ be a computable function.

Using ψ, we can extract a subsequence from $X \in 2^\omega$ as follows:

Stochastic Sequences (1)

Let $\psi : 2^{<\omega} \to \{0,1\}$ be a computable function.

Using ψ, we can extract a subsequence from $X \in 2^\omega$ as follows:

- If $\psi(\epsilon) = 1$, we include the first bit of X in our subsequence; if $\psi(\epsilon) = 0$, we exclude the first bit.

Stochastic Sequences (1)

Let $\psi : 2^{<\omega} \to \{0,1\}$ be a computable function.

Using ψ, we can extract a subsequence from $X \in 2^\omega$ as follows:

- If $\psi(\epsilon) = 1$, we include the first bit of X in our subsequence; if $\psi(\epsilon) = 0$, we exclude the first bit.
- After n steps, if $\psi(X{\upharpoonright}n) = 1$, we include the (n+1)st bit of X in our subsequence; otherwise, we exclude the (n+1)st bit.

Stochastic Sequences (2)

Let $X{\restriction}_{\psi(X)}$ denote the subsequence extracted from X using ψ.

Stochastic Sequences (2)

Let $X{\upharpoonright}_{\psi(X)}$ denote the subsequence extracted from X using ψ.

$X \in 2^\omega$ is stochastic if

Stochastic Sequences (2)

Let $X{\restriction}_{\psi(X)}$ denote the subsequence extracted from X using ψ.

$X \in 2^\omega$ is stochastic if

(1) X satisfies the law of large numbers, and

Stochastic Sequences (2)

Let $X\restriction_{\psi(X)}$ denote the subsequence extracted from X using ψ.

$X \in 2^\omega$ is stochastic if

(1) X satisfies the law of large numbers, and

(2) for every computable selection rule $\psi : 2^{<\omega} \to \{0,1\}$, $X\restriction_{\psi(X)}$ satisfies the law of large numbers.

Stochastic Sequences (3)

Not sufficient for randomness:

There is a stochastic sequence $X \in 2^\omega$ such that for all n,

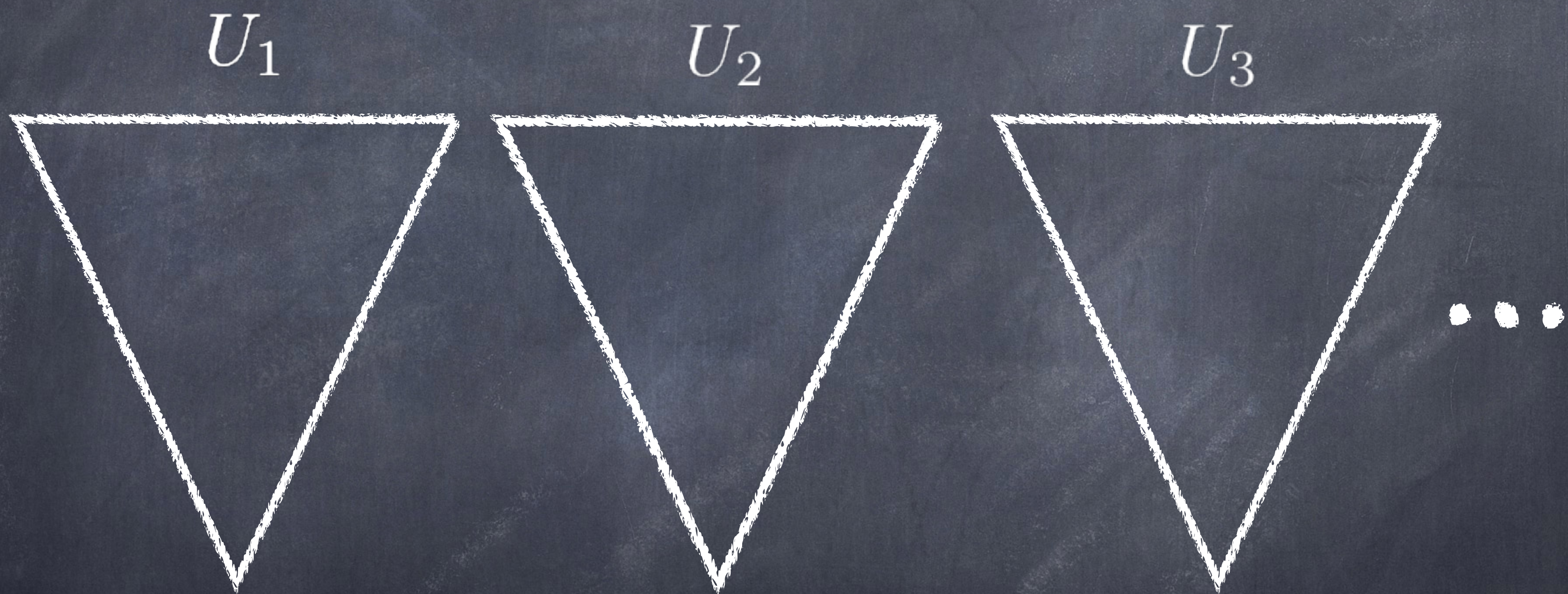
$$\frac{\#_0(X{\upharpoonright}n)}{n} > \frac{1}{2}\,.$$

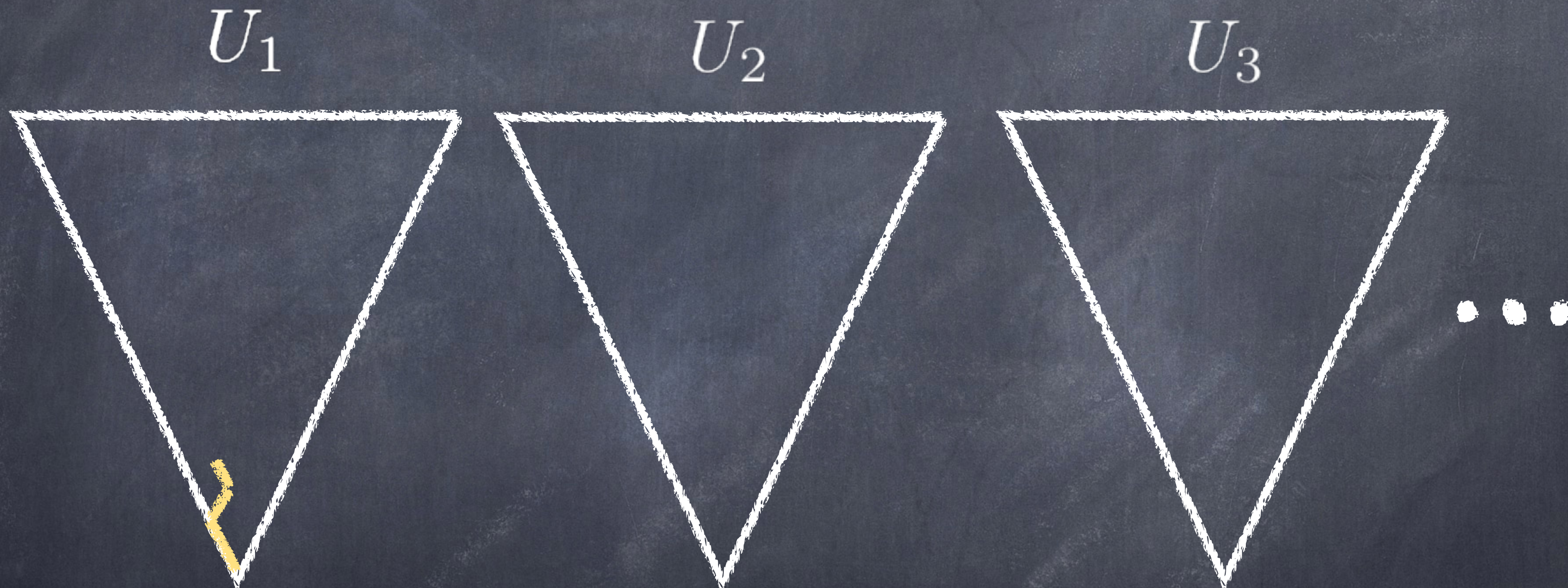
Martin-Löf Tests

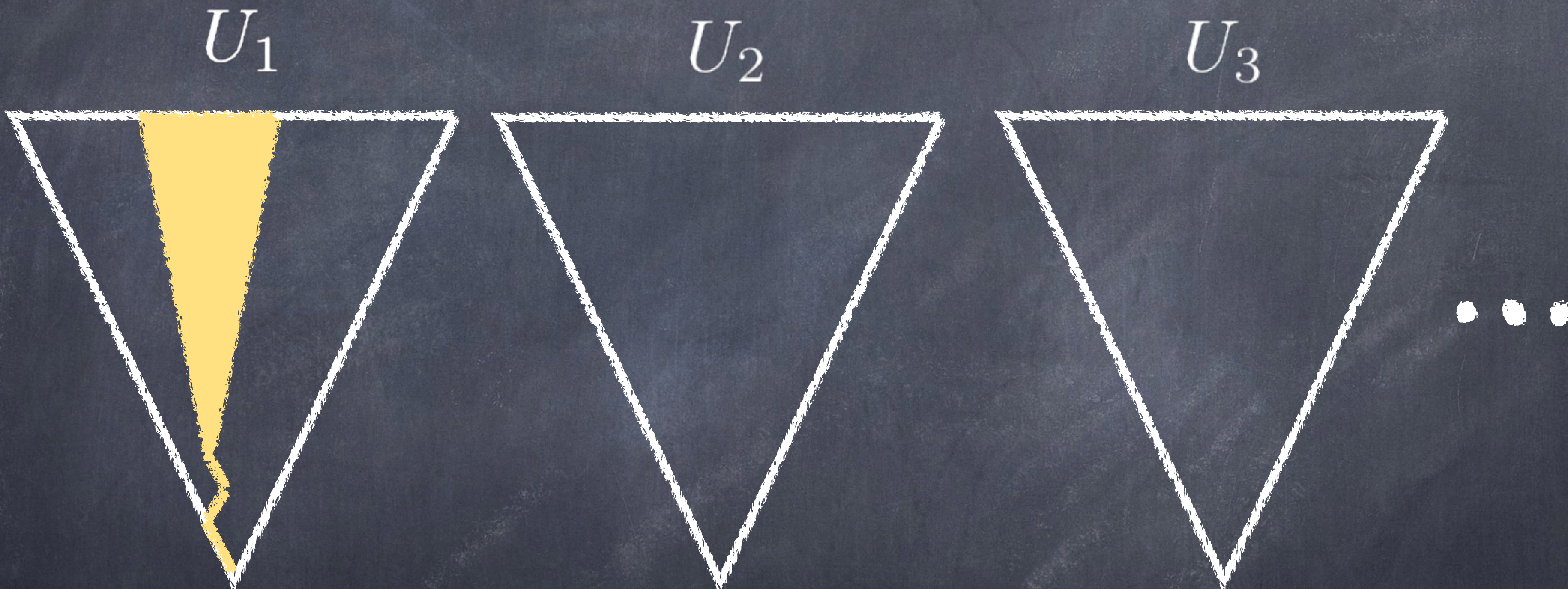
A Martin-Löf test is a sequence $(U_i)_{i\in\omega}$ of open subsets of 2^ω that are uniformly enumerated by some effective procedure and satisfy

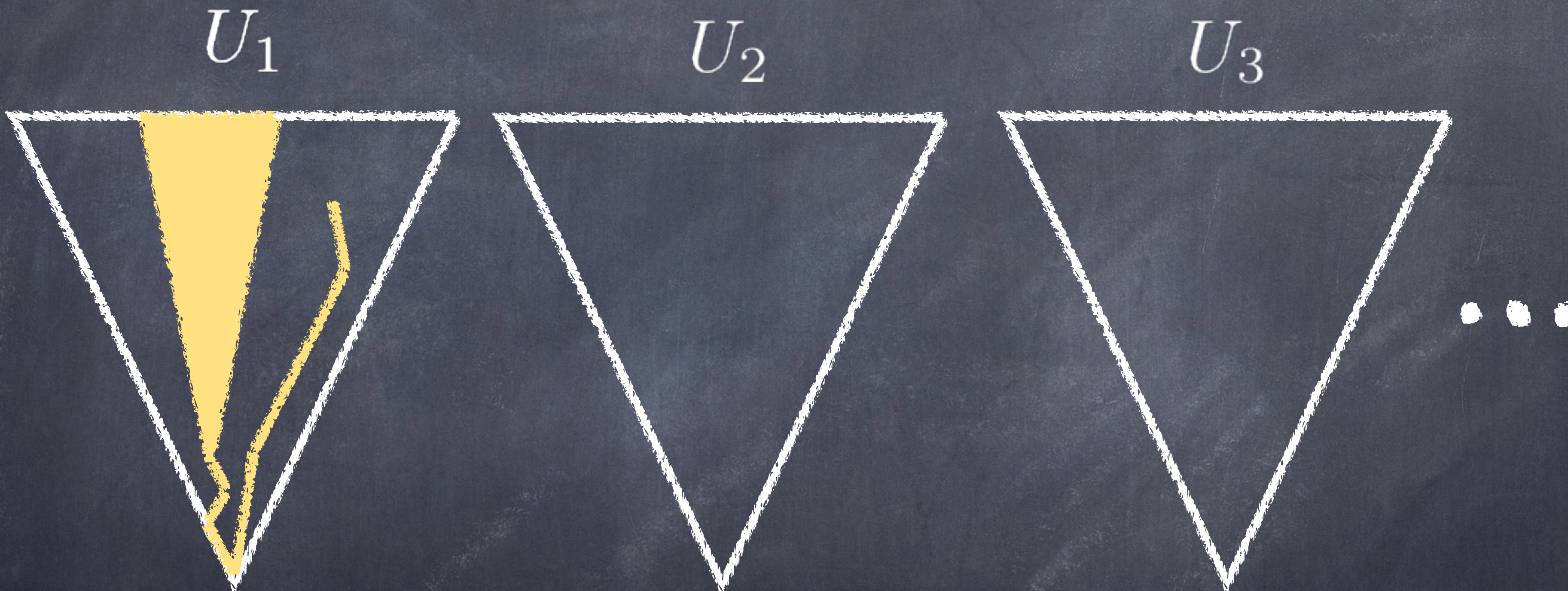
$$\lambda(U_i) \leq 2^{-i}$$

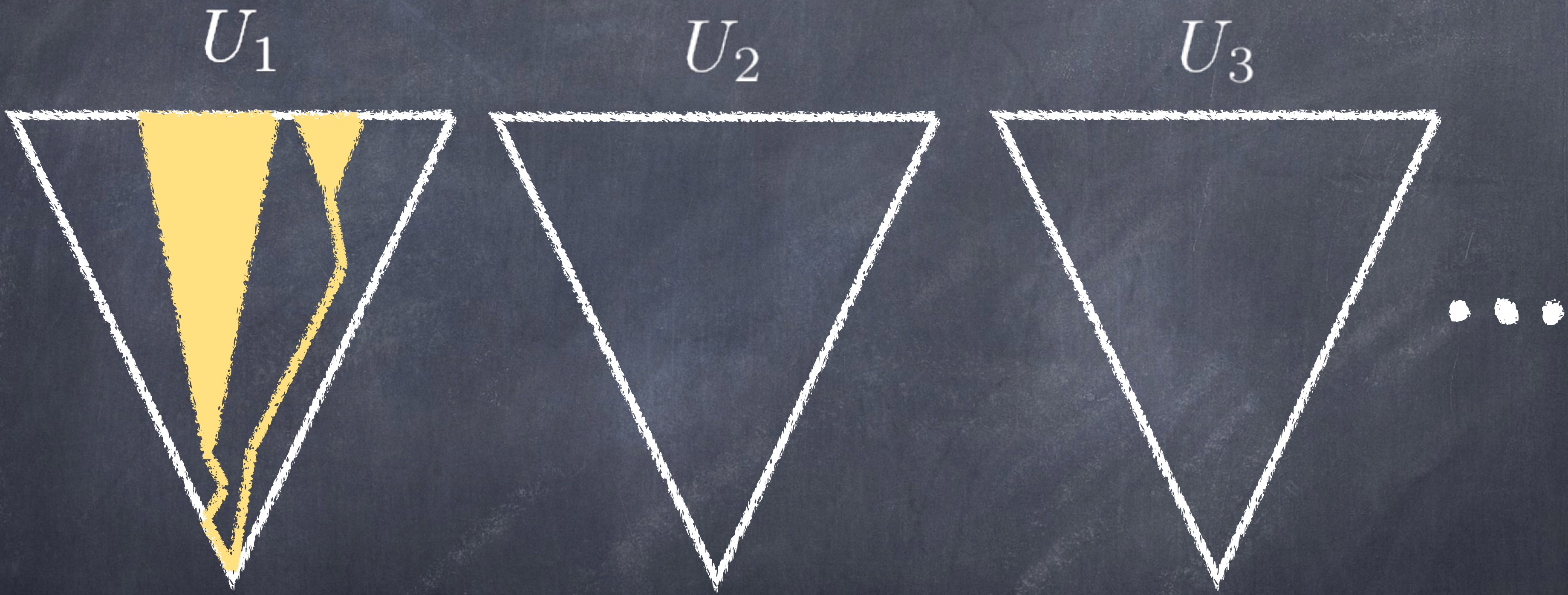
for every $i \in \omega$.

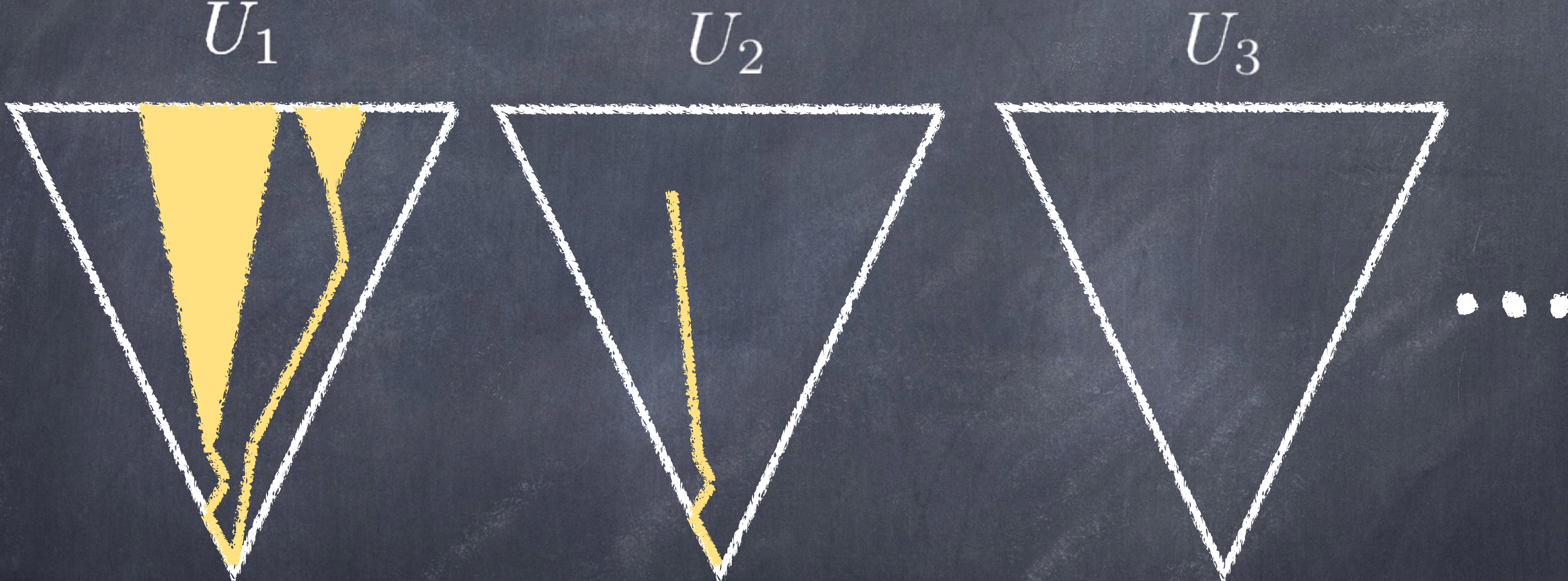

U_1
U_2
U_3
...

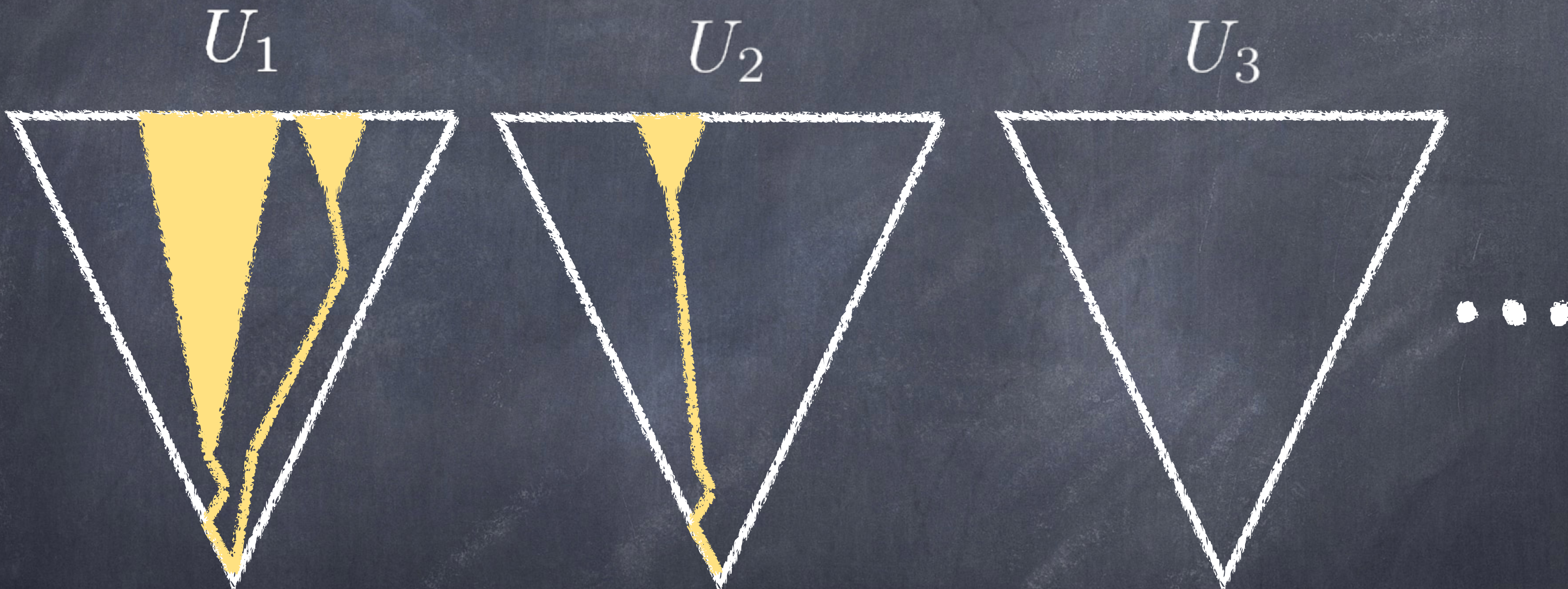

U_1
U_2
U_3
$\dots$


U_1
U_2
U_3
...

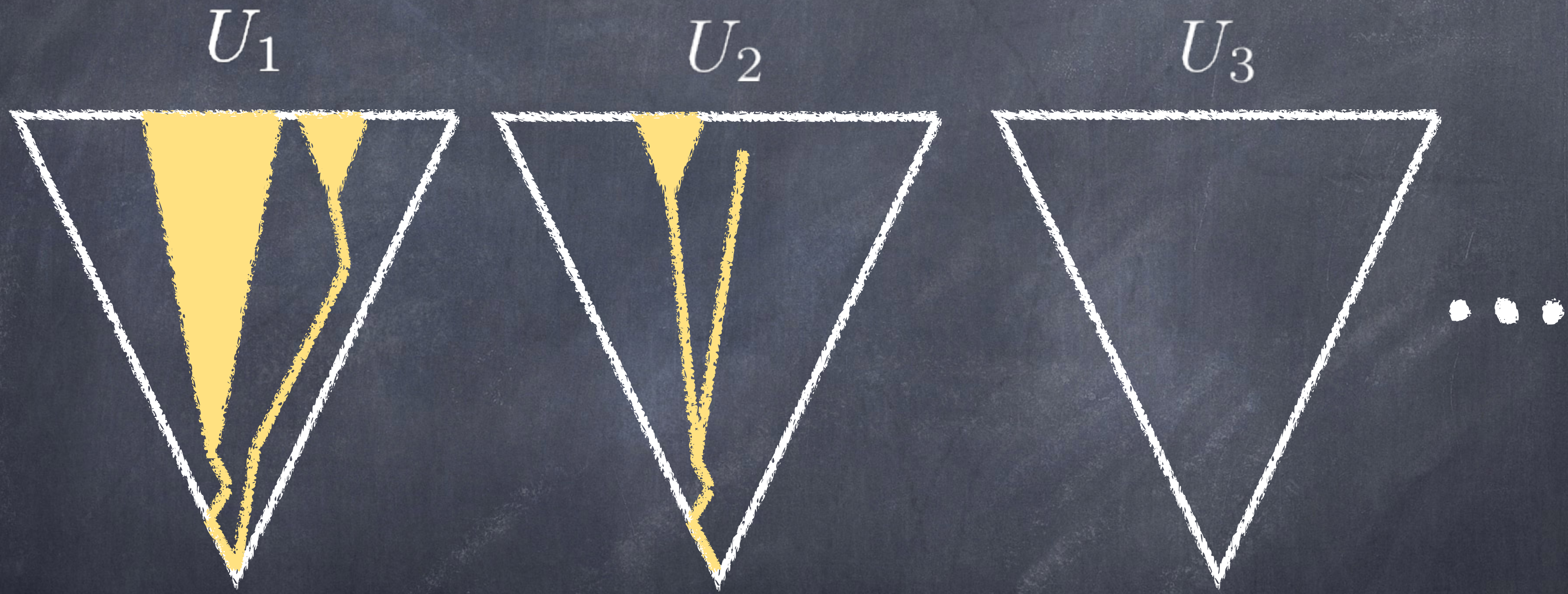

U_1
U_2
U_3
...

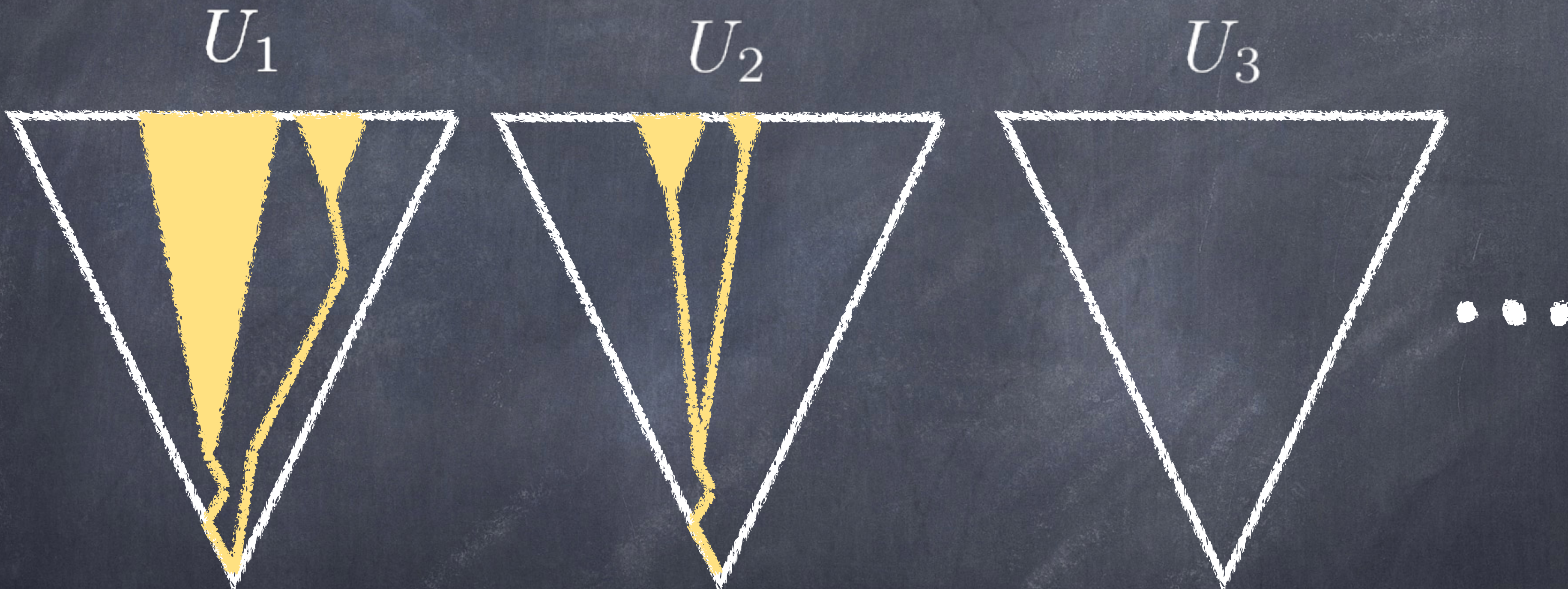

U_1
U_2
U_3
...


U_1
U_2
U_3
...

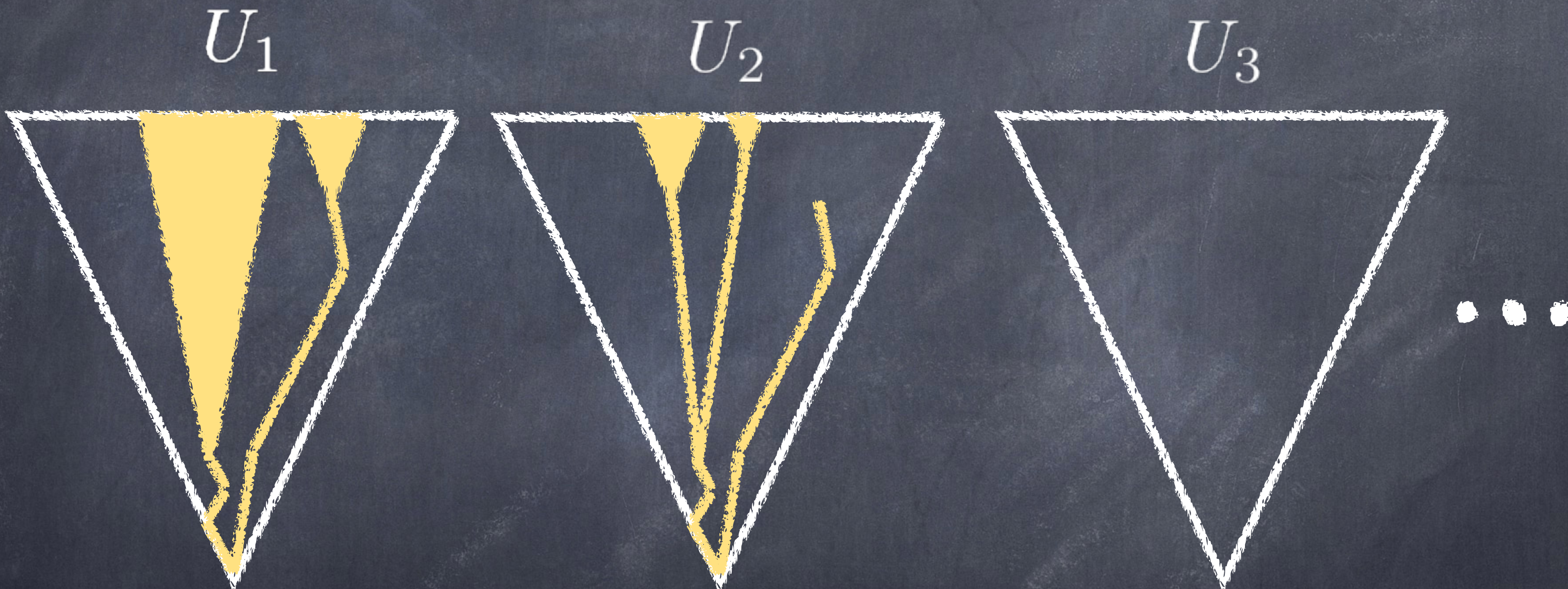


U_1
U_2
U_3
...

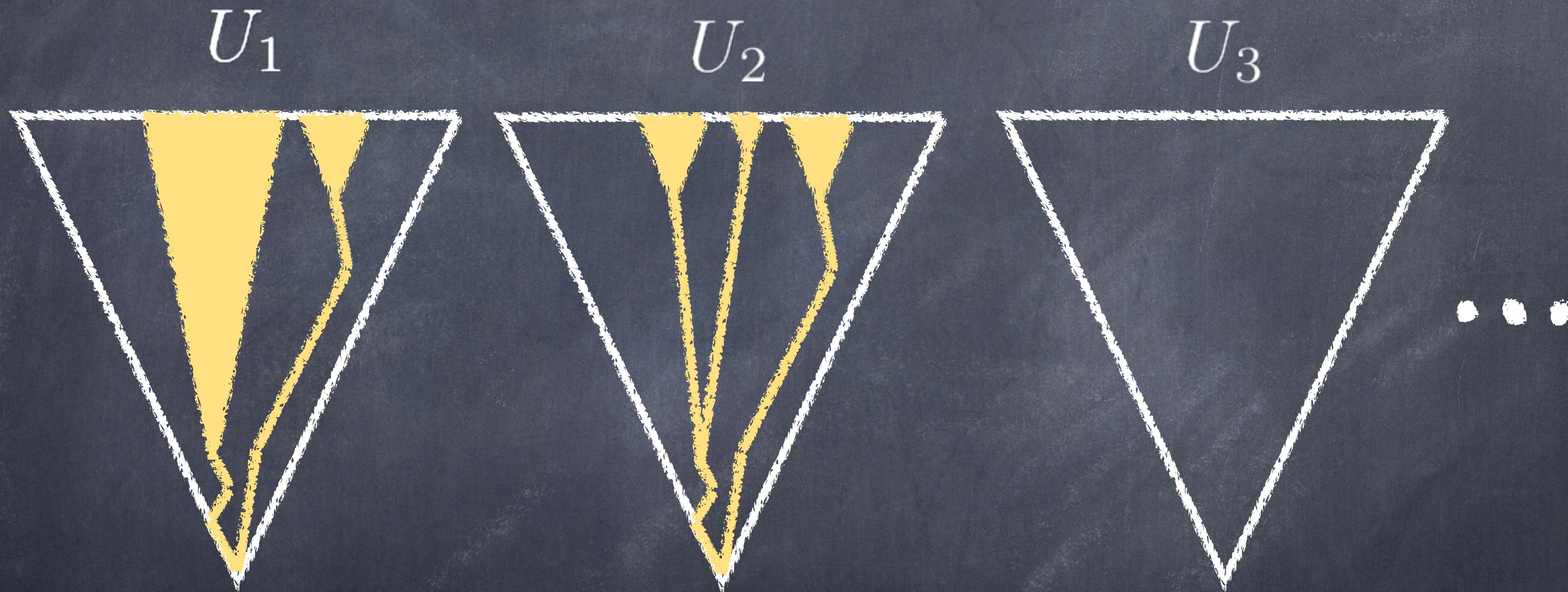

U_1
U_2
U_3
...

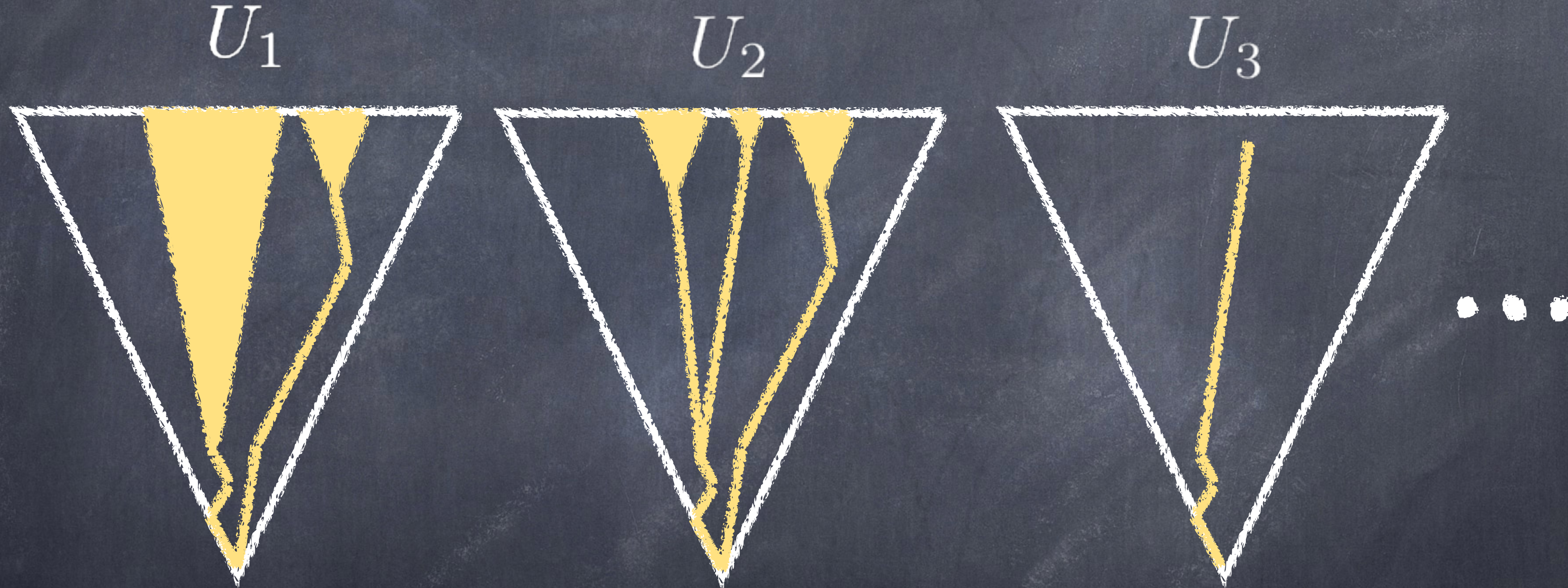


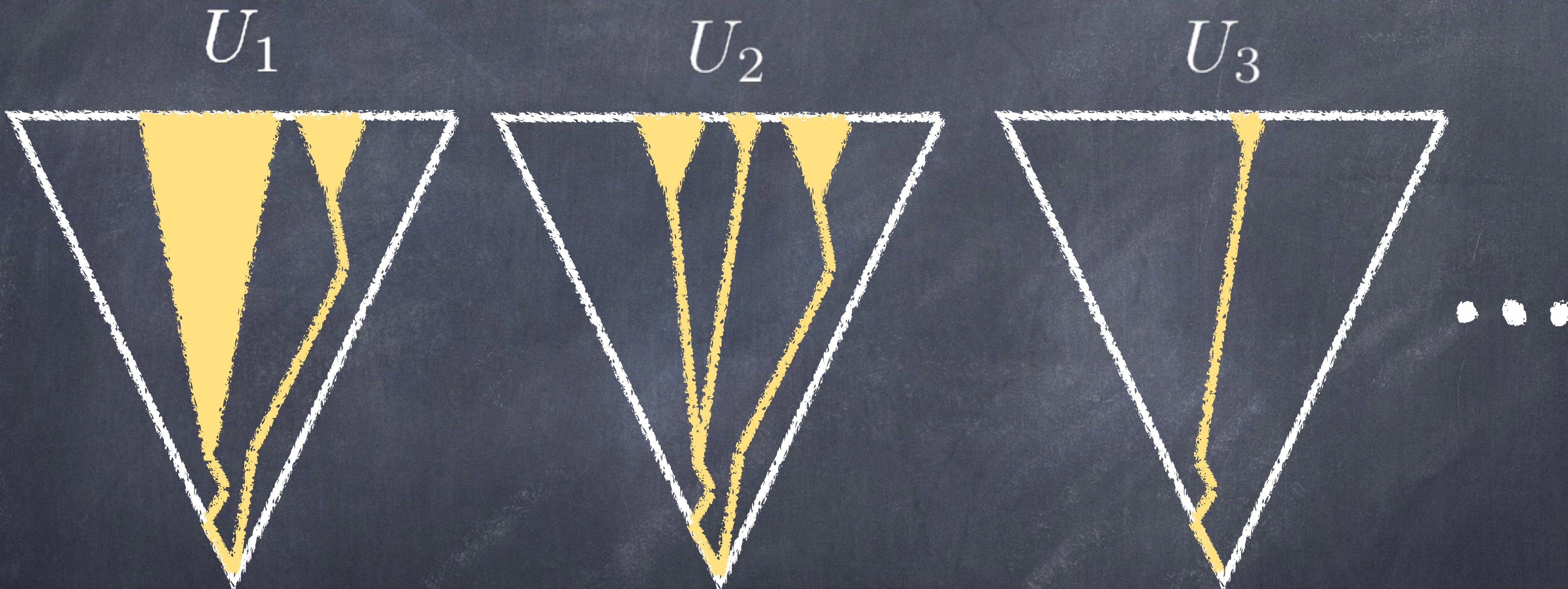
U_1
U_2
U_3
...

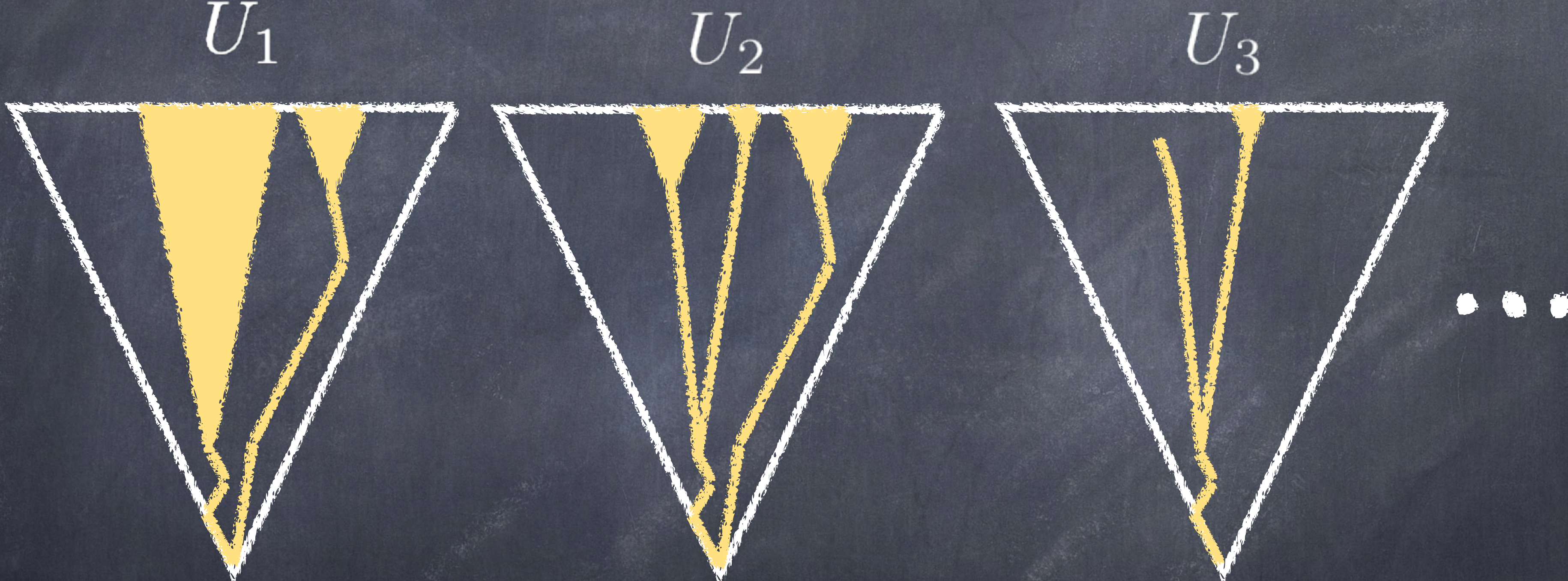


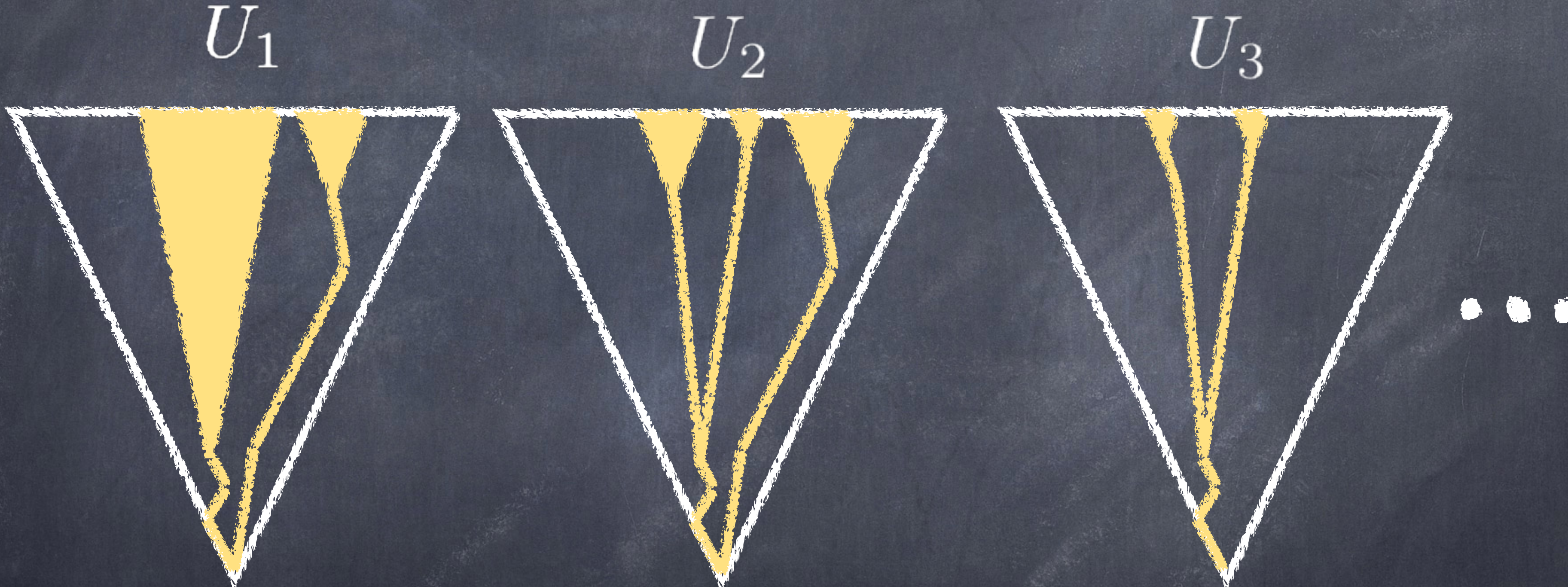
U_1
U_2
U_3
...

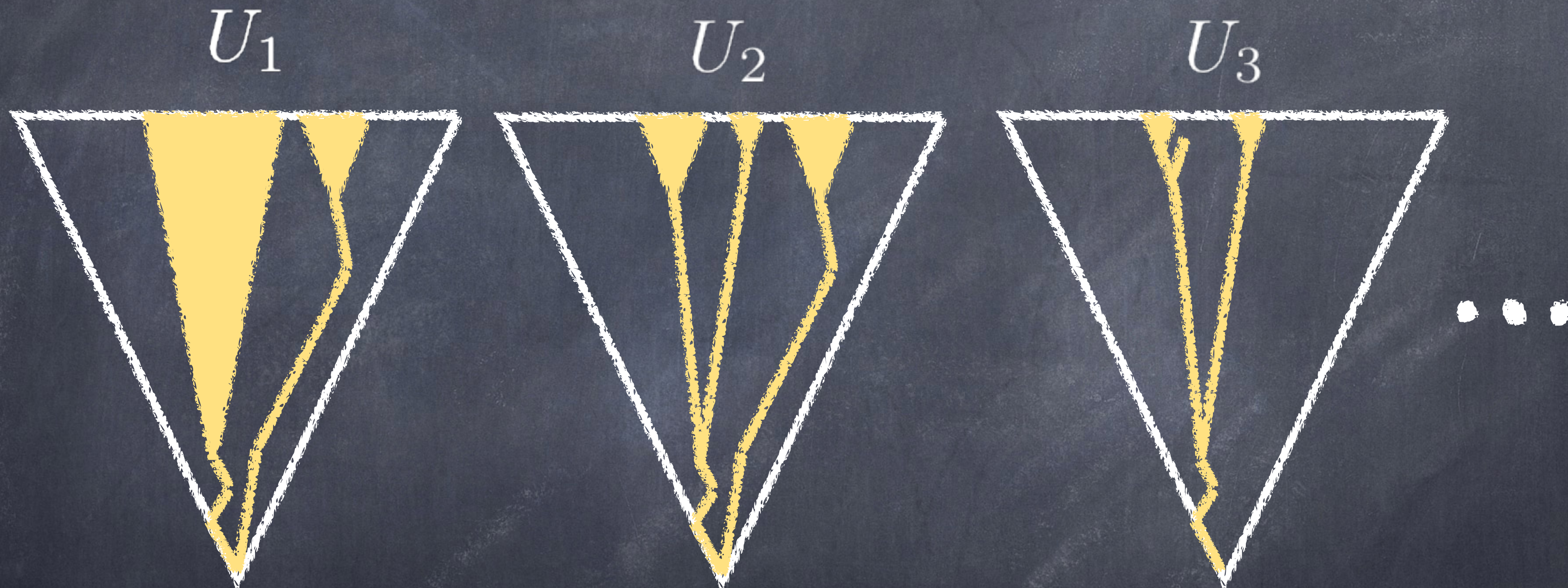

U_1
U_2
U_3
...

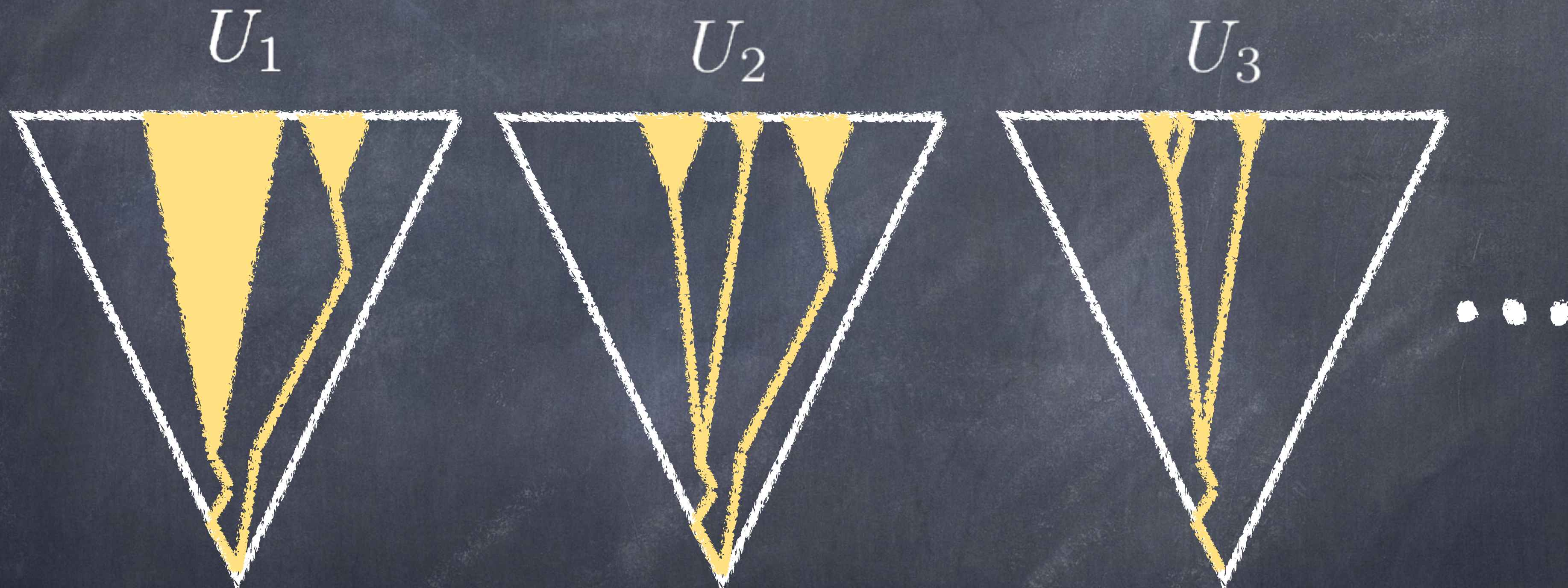

U_1
U_2
U_3
…

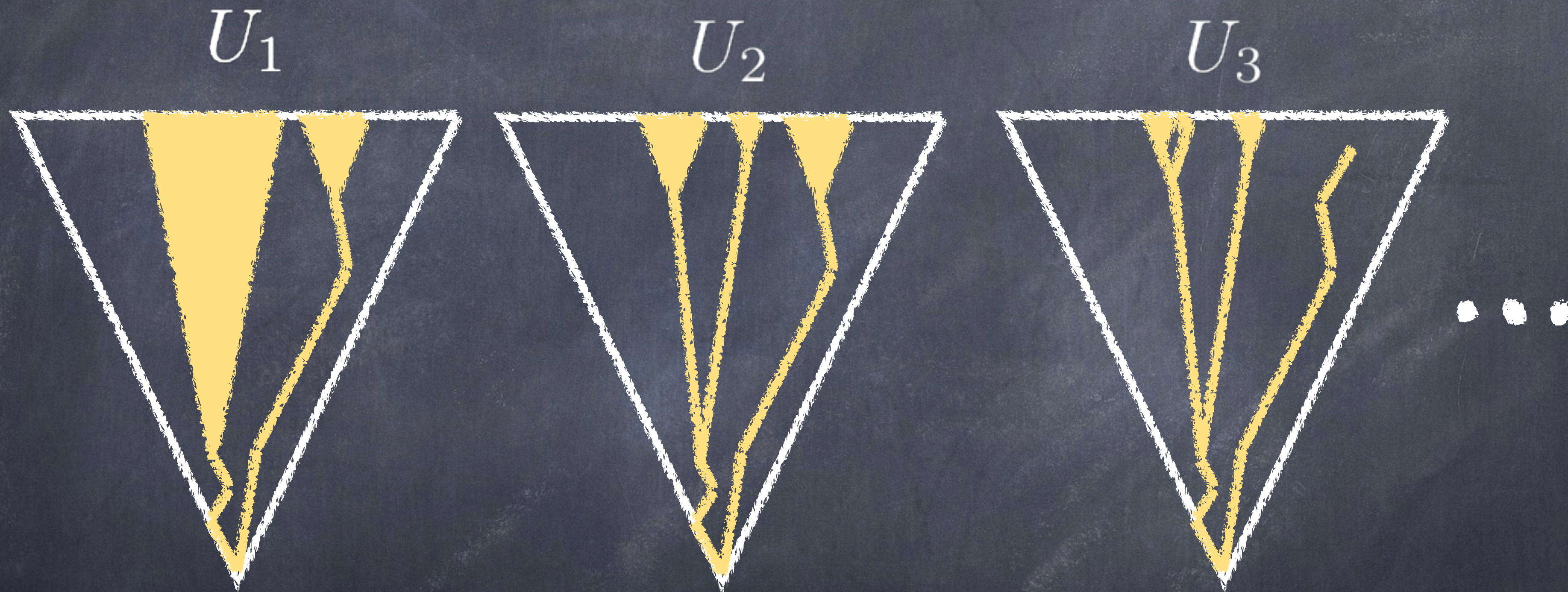

U_1
U_2
U_3
...

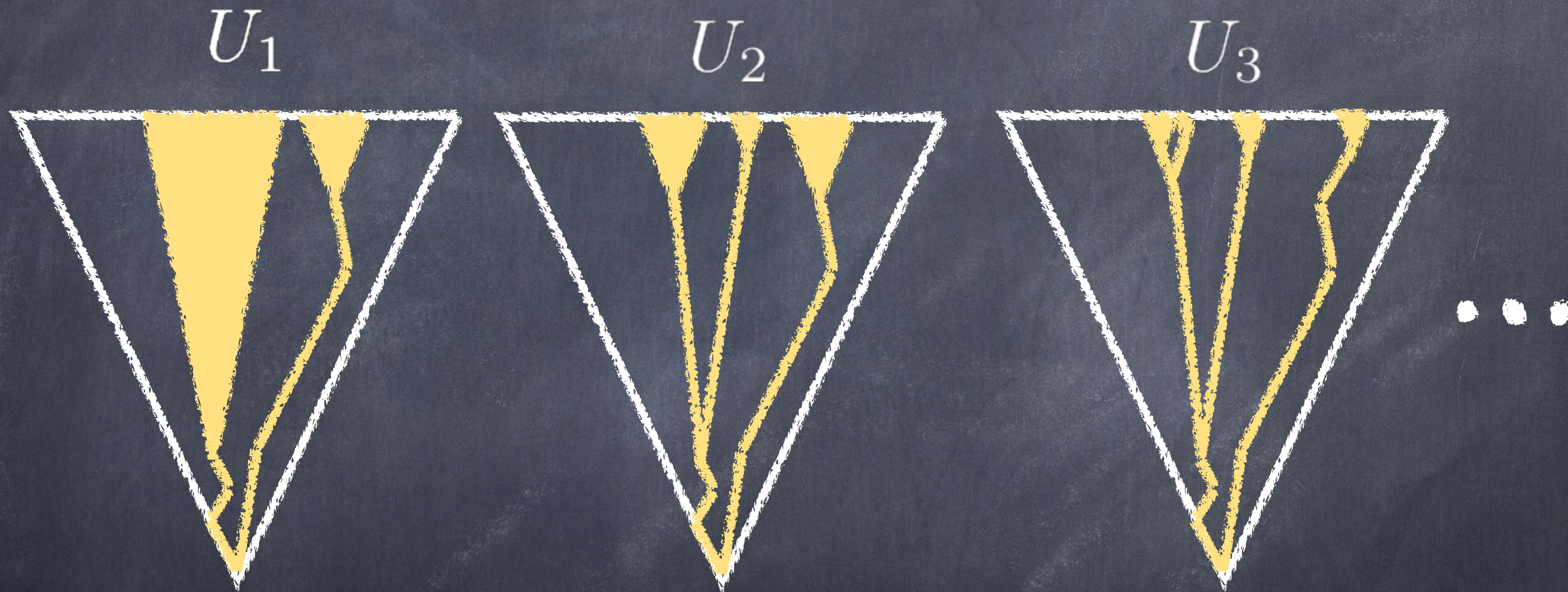

U_1
U_2
U_3
...

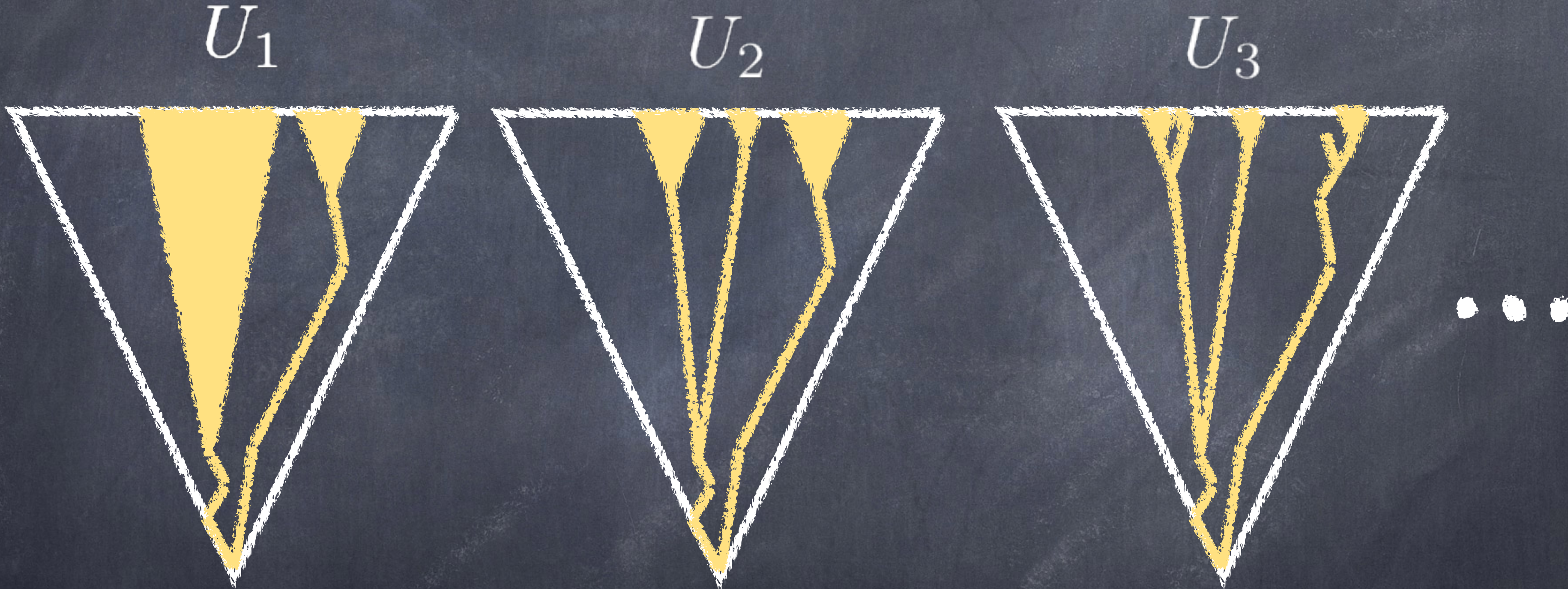

U_1
U_2
U_3
...

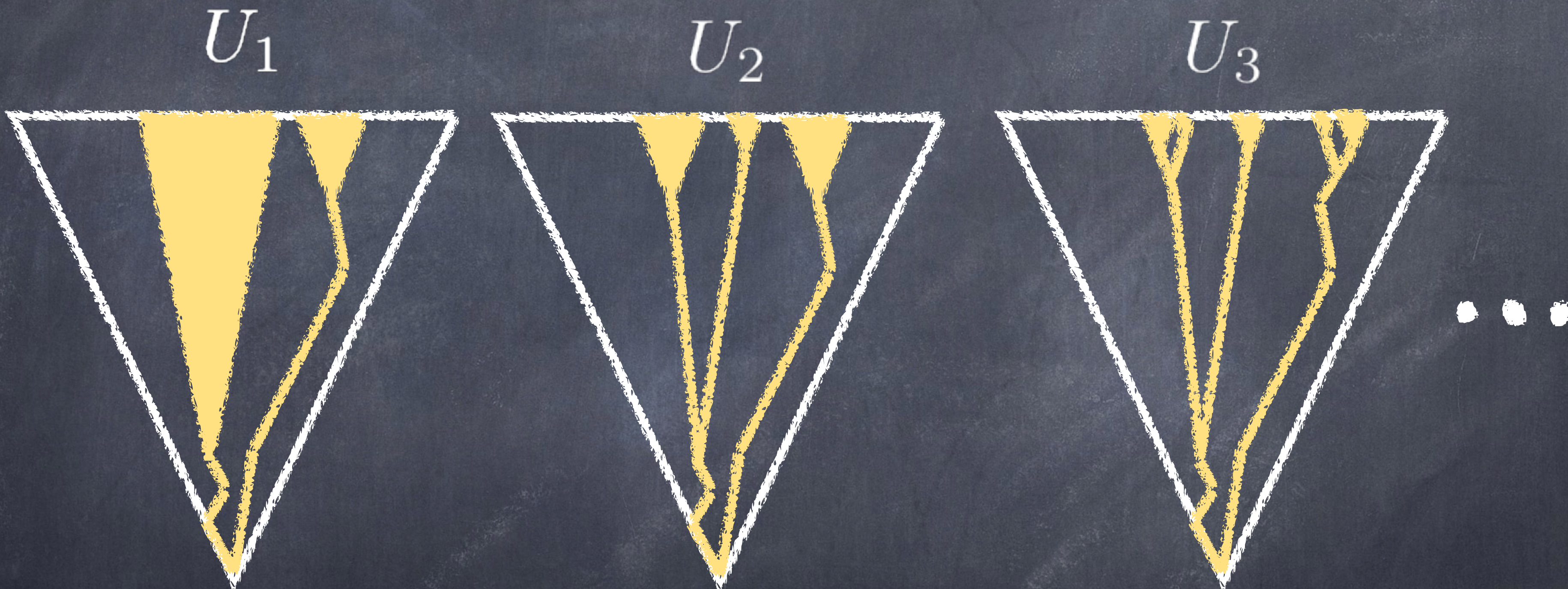

U_1
U_2
U_3
...

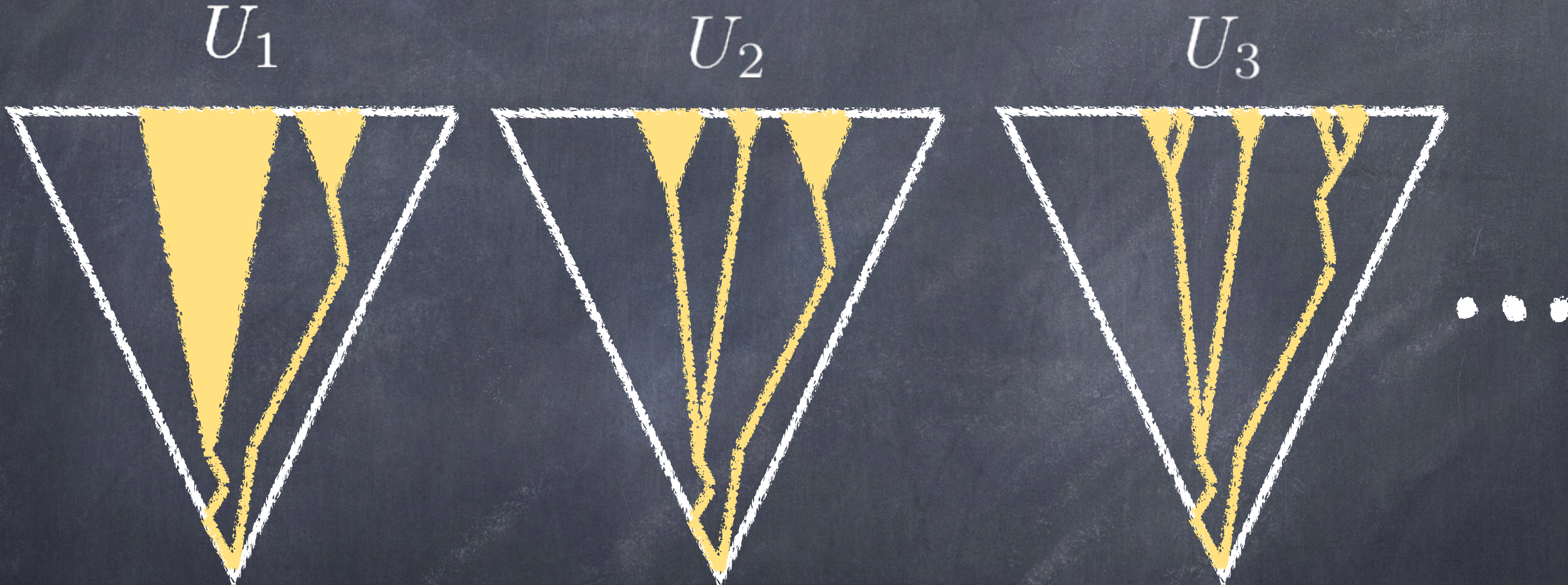



U_1
U_2
U_3
...

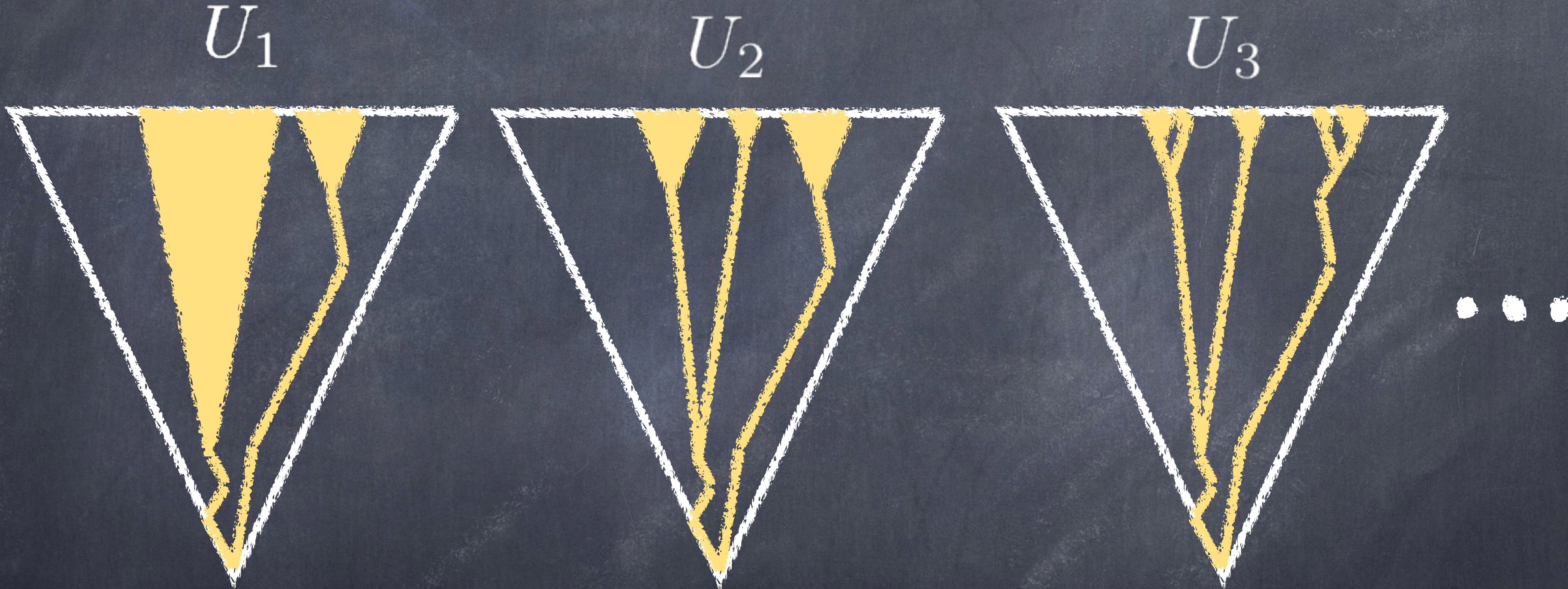

U_1
U_2
U_3
...


U_1
U_2
U_3
…


U_1
U_2
U_3
...



$$\sum_{\sigma \in U_1} 2^{-|\sigma|} \leq \frac{1}{2}$$



$$\sum_{\sigma \in U_1} 2^{-|\sigma|} \leq \frac{1}{2} \qquad \sum_{\sigma \in U_2} 2^{-|\sigma|} \leq \frac{1}{4}$$

U_1 $\quad$ U_2 $\quad$ U_3

$$\sum_{\sigma \in U_1} 2^{-|\sigma|} \leq \frac{1}{2} \qquad \sum_{\sigma \in U_2} 2^{-|\sigma|} \leq \frac{1}{4} \qquad \sum_{\sigma \in U_3} 2^{-|\sigma|} \leq \frac{1}{8}$$

Martin-Löf Randomness

$X \in 2^\omega$ is **Martin-Löf random** if for every Martin-Löf test $(U_i)_{i\in\omega}$,

$$X \notin \bigcap_{i\in\omega} U_i \, .$$

MLR denotes the collection of Martin-Löf random sequences.

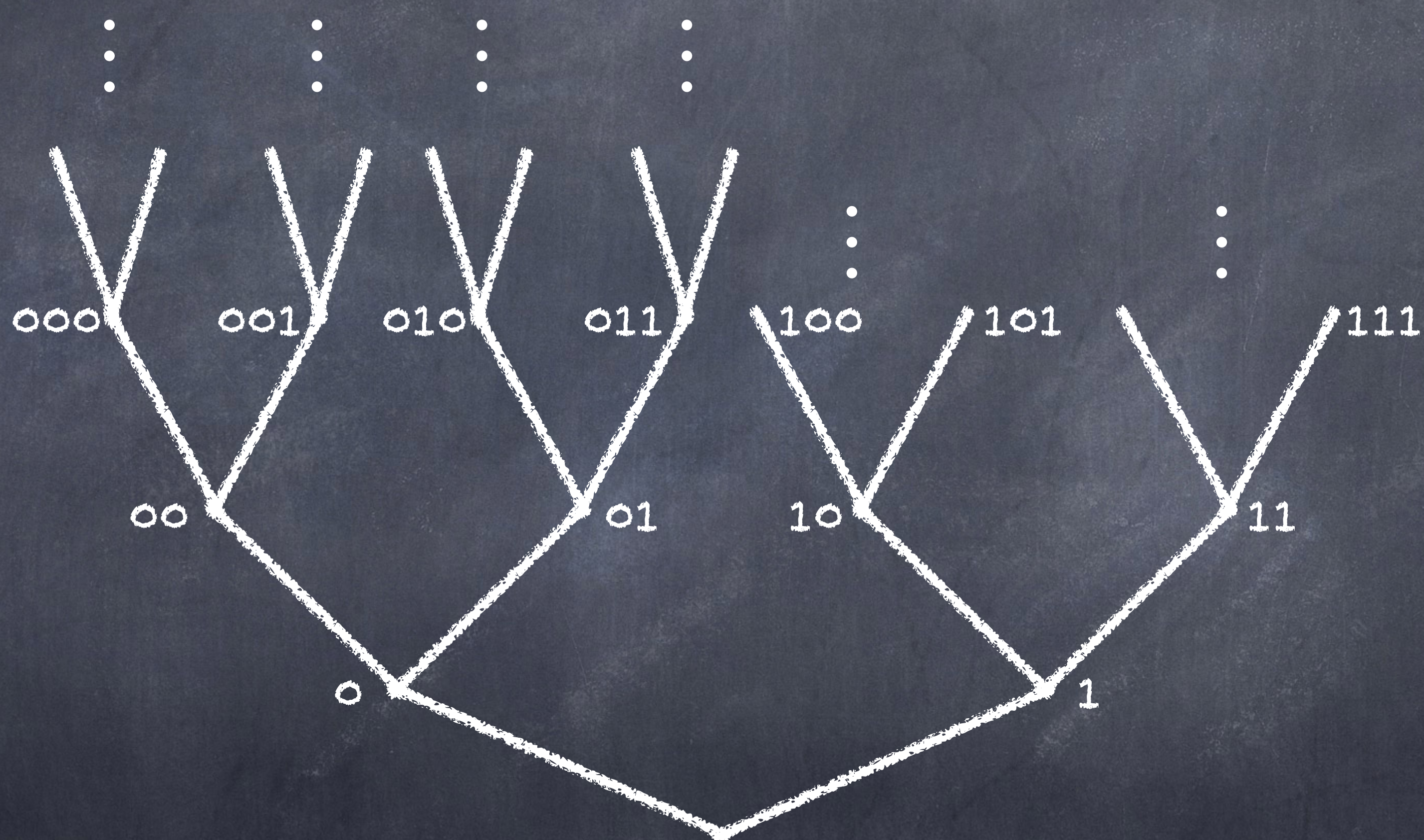
Martin-Löf Randomness

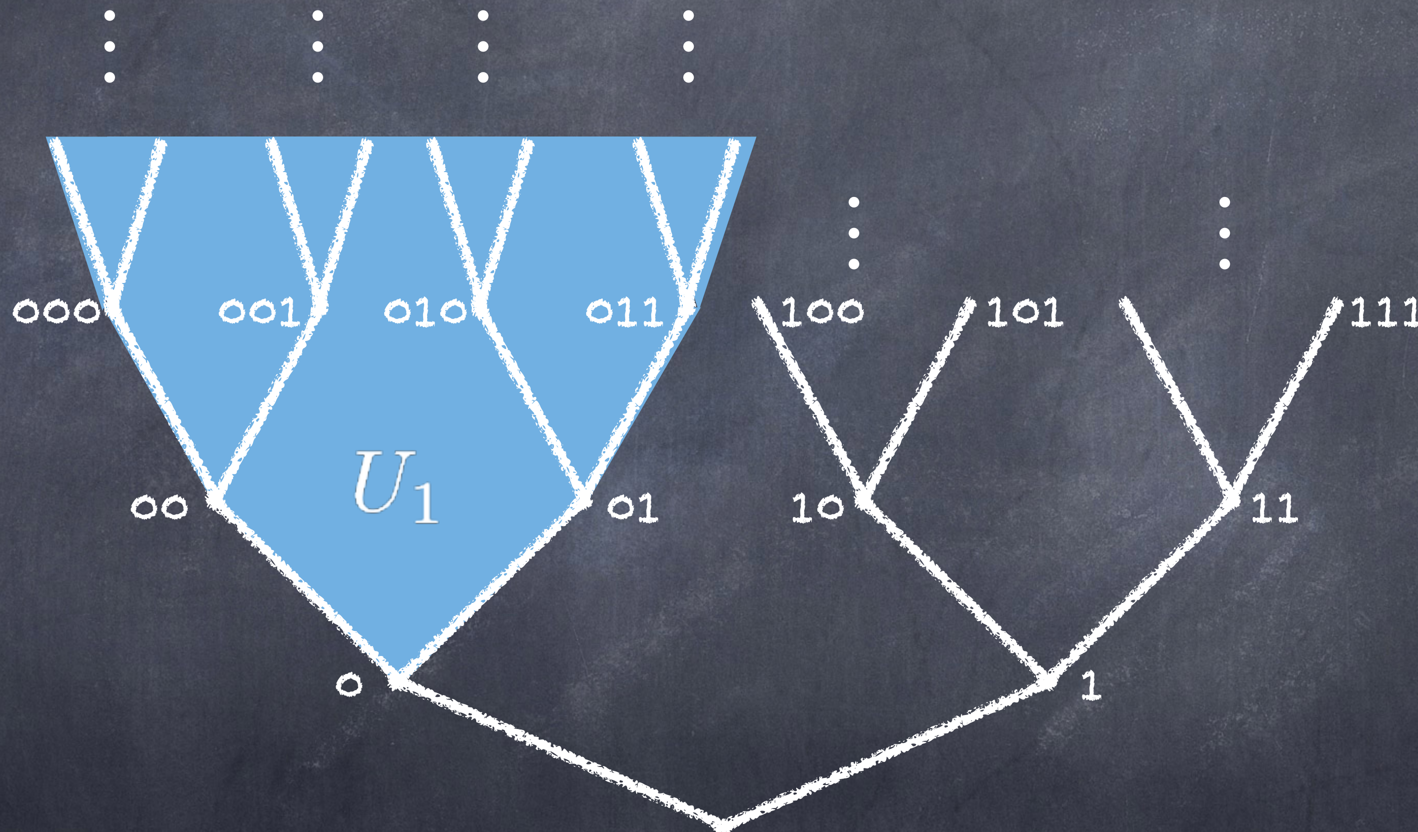
$X \in 2^\omega$ is Martin-Löf random if for every Martin-Löf test $(U_i)_{i\in\omega}$,

$$X \notin \bigcap_{i\in\omega} U_i .$$

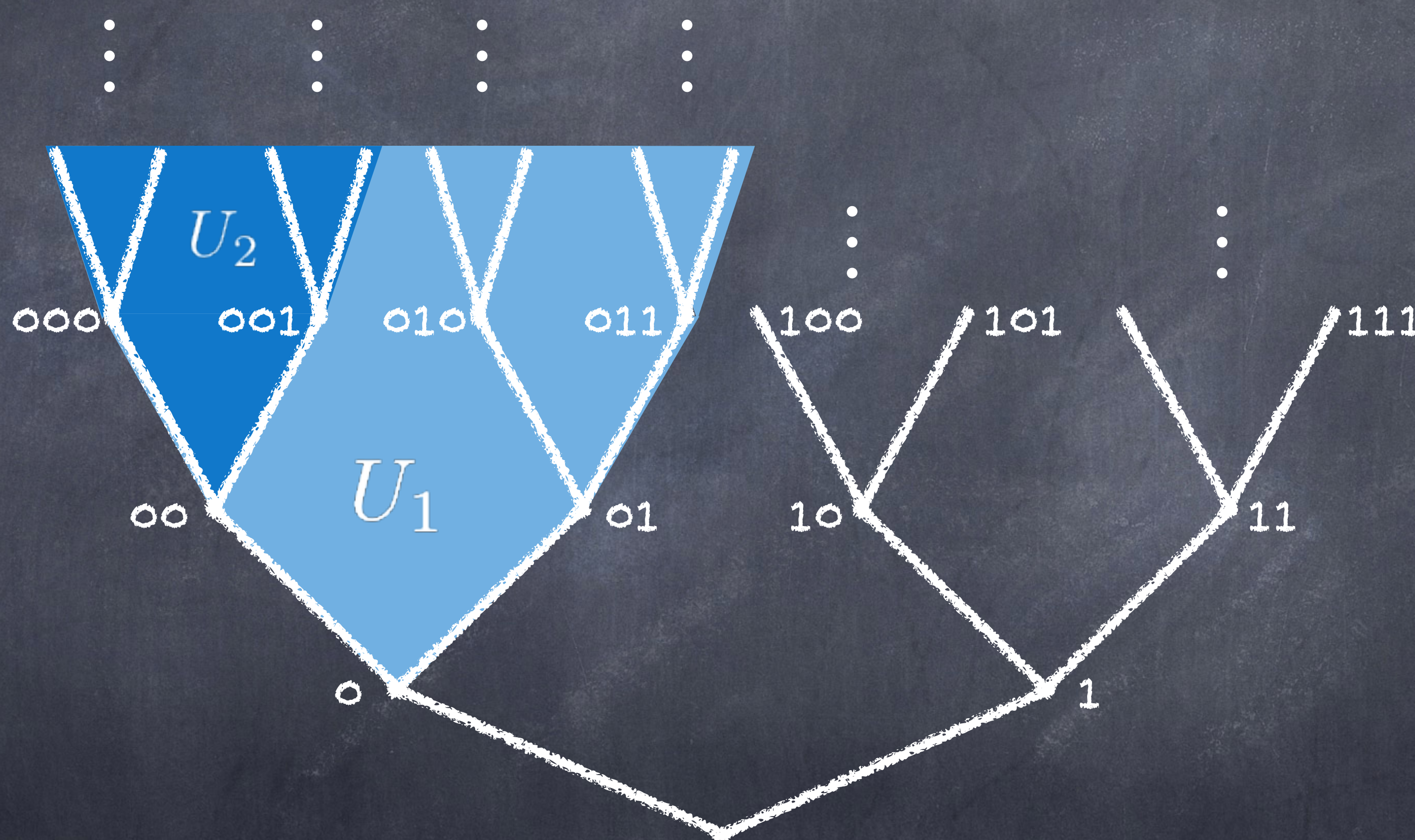
MLR denotes the collection of Martin-Löf random sequences.

Sufficient for randomness?

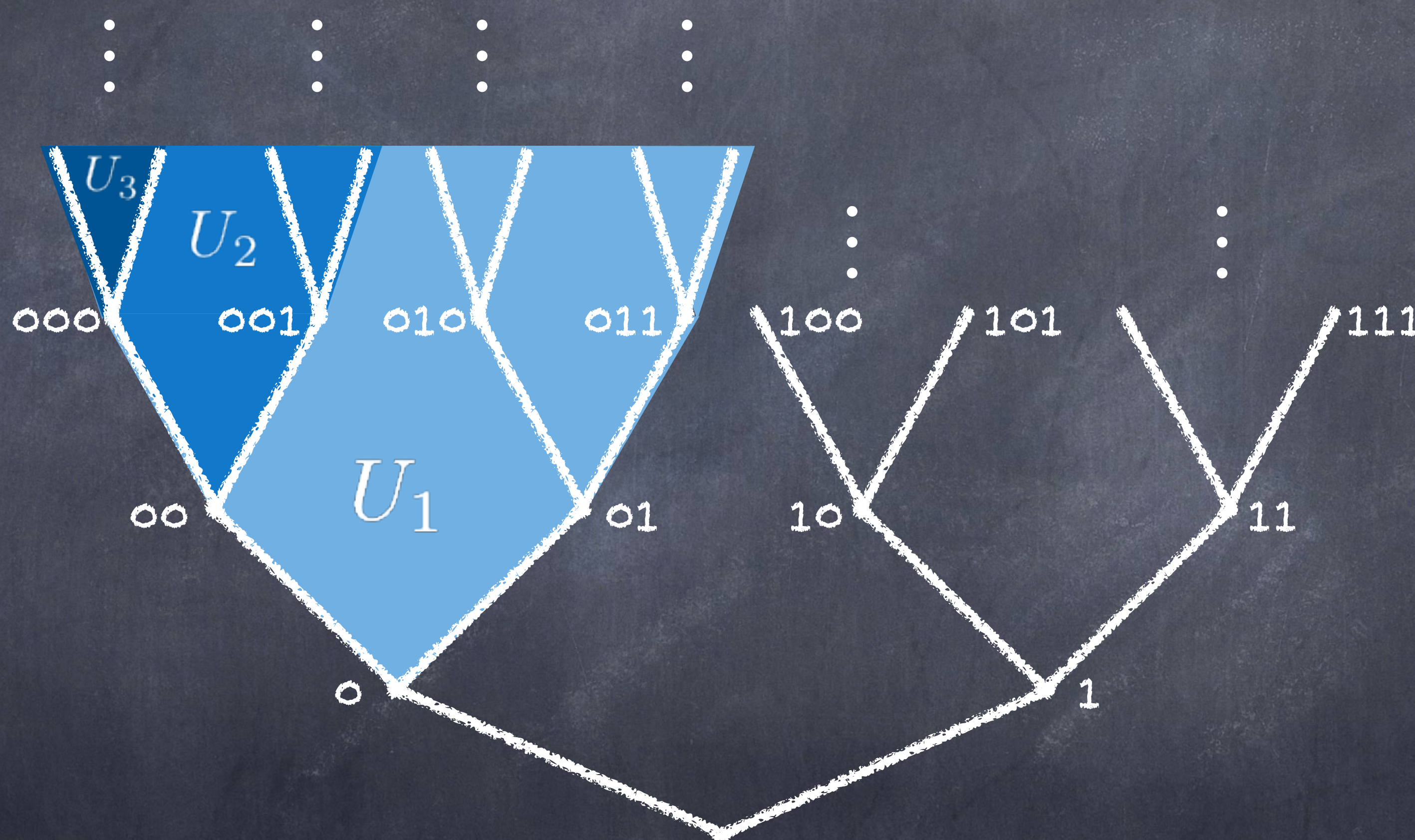

000
001
010
011
100
101
111
00
01
10
11
0
1



000
001
010
011
100
101
111
U_1
00
01
10
11
0
1



U_2
U_1
000
001
010
011
100
101
111
00
01
10
11
0
1



U_3
U_2
U_1
000
001
010
011
100
101
111
00
01
10
11
0
1

MLR for non-uniform measures

For any computable measure μ we can define Martin-Löf randomness with respect to μ.

MLR for non-uniform measures

For any computable measure μ we can define Martin-Löf randomness with respect to μ.

We simply replace the condition

$$\lambda(U_i) \leq 2^{-i}$$

with the condition

$$\mu(U_i) \leq 2^{-i}.$$

MLR for non-uniform measures

For any computable measure μ we can define Martin-Löf randomness with respect to μ.

We simply replace the condition

$$\lambda(U_i) \leq 2^{-i}$$

with the condition

$$\mu(U_i) \leq 2^{-i}.$$

MLR_μ = μ-ML-random sequences

Extraction (1)

Monotonic functions $\phi : 2^{<\omega} \to 2^{<\omega}$ can be extended to maps $\Phi : 2^{\omega} \to 2^{\omega}$:

$$\Phi(X) = \bigcup_{n \in \omega} \phi(X \restriction n)$$

Φ is a Turing functional if it is induced by a computable, monotonic ϕ .

Extraction (2)

Given $X, Y \in 2^\omega$, X is Turing reducible to Y if there is a Turing functional Φ such that $\Phi(Y) = X$.

X and Y are Turing equivalent if there are Turing functionals Φ and Ψ such that $\Phi(Y) = X$ and $\Psi(X) = Y$.

The main result

Levin and Kautz independently proved:

Theorem: Let μ be a computable measure on 2^ω. For every $X \in \mathsf{MLR}_\mu$ such that X is not computable, there is some $Y \in \mathsf{MLR}$ such that X and Y are Turing equivalent.

Some comments (1)

Unlike von Neumann's trick, the Levin/Kautz conversion procedure works for any computable measure (in fact, it works for any non-computable measure as well).

Some comments (1)

Unlike von Neumann's trick, the Levin/Kautz conversion procedure works for any computable measure (in fact, it works for any non-computable measure as well).

However, the conversion requires that we also have access to the underlying biased measure.

Some comments (2)

Joint work with Laurent Bienvenu:

There are biased random sequences such that there is no computable bound on the number of biased input bits needed to guarantee the output of n unbiased bits for any effective conversion procedure.

Thank you!