BILL AND NATHAN RECORD LECTURE!!!!

BILL AND NATHAN RECORD LECTURE!!!

UN-TIMED PART OF FINAL IS TUESDAY May 11 11:00A. NO DEAD CAT

FINAL IS THURSDAY

May 13

8:00PM-10:15PM

FILL OUT COURSE EVALS for ALL YOUR COURSES!!!

Kolmogorov Complexity

Exposition by William Gasarch—U of MD

The Complexity of a Finite String

Which of these strings looks more random?
(This is NOT a trick question. Your intuitions will be correct. We will formalize them.)

The Complexity of a Finite String

Which of these strings looks more random?
(This is NOT a trick question. Your intuitions will be correct. We will formalize them.)

1. 000000000000000000000000000000000
2. 011010001100000011101010100011001

The Complexity of a Finite String

Which of these strings looks more random?
(This is NOT a trick question. Your intuitions will be correct. We will formalize them.)

1. 000000000000000000000000000000000
2. 011010001100000011101010100011001

You prob think the second string is **more random** then the first.

The Complexity of a Finite String

Which of these strings looks more random?
(This is NOT a trick question. Your intuitions will be correct. We will formalize them.)

1. 000000000000000000000000000000000
2. 011010001100000011101010100011001

You prob think the second string is **more random** then the first.
You are right!

The Complexity of a Finite String

Which of these strings looks more random?
(This is NOT a trick question. Your intuitions will be correct. We will formalize them.)

1. 000000000000000000000000000000000
2. 011010001100000011101010100011001

You prob think the second string is **more random** then the first.
You are right!
How do we pin this down?

The Complexity of a Finite String

Which of these strings looks more random?
(This is NOT a trick question. Your intuitions will be correct. We will formalize them.)

1. 00000000000000000000000000000000
2. 01101000110000001110101010001100 1

You prob think the second string is **more random** then the first.
You are right!
How do we pin this down? Discuss!

A Programs to Print Out $0 \cdots 0$

Here is a program to print out
000000000000000000000000000000000

$$\texttt{For } i = 1 \texttt{ to 33 print(0)}$$

A Programs to Print Out $0 \cdots 0$

Here is a program to print out
000000000000000000000000000000000

$$\texttt{For } i = 1 \texttt{ to 33 print(0)}$$

The string was of length 33 but the program is far shorter.

A Programs to Print Out $0 \cdots 0$

Here is a program to print out
000000000000000000000000000000000

`For` $i = 1$ `to 33 print(0)`

The string was of length 33 but the program is far shorter.

For the string 0^n the string is length n, the program is length $\lg(n) + O(1)$.

A Programs to Print Out the Second String

Here is a program to print out
0110100011000000111010101010001100.

A Programs to Print Out the Second String

Here is a program to print out 01101000110000001110101010001100.

```
print(01101000110000001110101010001100)
```

A Programs to Print Out the Second String

Here is a program to print out
011010001100000011101010100011 00.

```
print(011010001100000011101010100011 00)
```

The string is of length 33 and the program is of length 33.

Upshot The **less random string** required a much shorter program to print it out then the **more random string**.

Def of Randomness

Taking a cue from the above two examples, we will define the **Randomness of a string *x*** to be the size of the shortest Turing Machine (TM) that prints x.

Def

Def of Randomness

Taking a cue from the above two examples, we will define the **Randomness of a string** $\boldsymbol{x}$ to be the size of the shortest Turing Machine (TM) that prints x.

Def

1. If $x \in \{0,1\}^n$ then $\boldsymbol{C(x)}$ is the length of the shortest TM that, on input e, prints out x. Note that $C(x) \leq n + O(1)$.

Def of Randomness

Taking a cue from the above two examples, we will define the **Randomness of a string x** to be the size of the shortest Turing Machine (TM) that prints x.

Def

1. If $x \in \{0,1\}^n$ then $C(x)$ is the length of the shortest TM that, on input e, prints out x. Note that $C(x) \le n + O(1)$.
2. If $x \in \{0,1\}^n$ then $C(x|y)$ is the length of the shortest TM that, on input y, prints out x. Note that $C(x|y) \le n + O(1)$.

Def of Randomness

Taking a cue from the above two examples, we will define the **Randomness of a string x** to be the size of the shortest Turing Machine (TM) that prints x.

Def

1. If $x \in \{0,1\}^n$ then **$C(x)$** is the length of the shortest TM that, on input e, prints out x. Note that $C(x) \le n + O(1)$.
2. If $x \in \{0,1\}^n$ then **$C(x|y)$** is the length of the shortest TM that, on input y, prints out x. Note that $C(x|y) \le n + O(1)$.
3. A string is **Kolmogorov random** if $C(x) \ge n$. A string is **Kolmogorov random relative to y** if $C(x|y) \ge n$.

Def of Randomness

Taking a cue from the above two examples, we will define the **Randomness of a string x** to be the size of the shortest Turing Machine (TM) that prints x.

Def

1. If $x \in \{0,1\}^n$ then $C(x)$ is the length of the shortest TM that, on input e, prints out x. Note that $C(x) \le n + O(1)$.
2. If $x \in \{0,1\}^n$ then $C(x|y)$ is the length of the shortest TM that, on input y, prints out x. Note that $C(x|y) \le n + O(1)$.
3. A string is **Kolmogorov random** if $C(x) \ge n$. A string is **Kolmogorov random relative to y** if $C(x|y) \ge n$.

Do you like these definitions?

Like-Dislike-Like

Like The definition works in that a string with some sort of pattern will have low randomness.

Like-Dislike-Like

Like The definition works in that a string with some sort of pattern will have low randomness.

Dislike Java-Random, Python-Random, 1-tape-TM-Random will all give different values. Want a def that is **model-independent**.

Like-Dislike-Like

Like The definition works in that a string with some sort of pattern will have low randomness.

Dislike Java-Random, Python-Random, 1-tape-TM-Random will all give different values. Want a def that is **model-independent**.

Like Translating a program from Java to Python to $\cdots$ is only CONSTANT overhead. If $C(x) = n$ in Java then $C(x)$ will be $n \pm O(1)$ in Python. This will be good enough for our purposes.

Like-Dislike-Like

Like The definition works in that a string with some sort of pattern will have low randomness.

Dislike Java-Random, Python-Random, 1-tape-TM-Random will all give different values. Want a def that is **model-independent**.

Like Translating a program from Java to Python to $\cdots$ is only CONSTANT overhead. If $C(x) = n$ in Java then $C(x)$ will be $n \pm O(1)$ in Python. This will be good enough for our purposes.

Convention We pick one model, TMs, and note that our results are up to an $O(1)$.

Do Random Strings Exist?

Is there a string of length n that has $C(x) \geq n$?

Do Random Strings Exist?

Is there a string of length n that has $C(x) \geq n$?

Breakout Rooms

Do Random Strings Exist? (cont)

Thm For all $n \in \mathbb{N}$ there is a string of length n that has $C(x) \geq n$.
How many strings are there of length n? 2^n.

Do Random Strings Exist? (cont)

Thm For all $n \in \mathbb{N}$ there is a string of length n that has $C(x) \geq n$.
How many strings are there of length n? 2^n.

How many TMs are there of length $\leq n-1$?
$2^0 + \cdots + 2^{n-1} = 2^n - 1$.

Do Random Strings Exist? (cont)

Thm For all $n \in \mathbb{N}$ there is a string of length n that has $C(x) \geq n$.
How many strings are there of length n? 2^n.

How many TMs are there of length $\leq n-1$?
$2^0 + \cdots + 2^{n-1} = 2^n - 1$.

Map all elements of $\{0,1\}^n$ to the shortest program that prints it out. Since there are 2^n strings and only $2^n - 1$ programs of length $\leq n-1$ some string maps to a program of length $\geq n$.

Application of Kolmogorov Complexity to Proving Languages Not Regular

Exposition by William Gasarch—U of MD

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let n be a number such that $C(n)$ is large (we say how large later).

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let n be a number such that $C(n)$ is large (we say how large later).

We describe a short machine that prints out n.

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let n be a number such that $C(n)$ is large (we say how large later).

We describe a short machine that prints out n.

This step is preprocessing. Feed a^n into M. It ends in state r.

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let n be a number such that $C(n)$ is large (we say how large later).

We describe a short machine that prints out n.

This step is preprocessing. Feed a^n into M. It ends in state r.

Key b^n is the **only** string x such that $\delta(r, x) \in F$.

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let n be a number such that $C(n)$ is large (we say how large later).

We describe a short machine that prints out n.

This step is preprocessing. Feed a^n into M. It ends in state r.

Key b^n is the **only** string x such that $\delta(r, x) \in F$.

The following program prints out n.
Compute $\delta(r, b)$, $\delta(r, bb)$, $\cdots$ *until find an m such that* $\delta(r, b^m) \in F$. *Print out m.*

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let n be a number such that $C(n)$ is large (we say how large later).

We describe a short machine that prints out n.

This step is preprocessing. Feed a^n into M. It ends in state r.

Key b^n is the **only** string x such that $\delta(r, x) \in F$.

The following program prints out n.
Compute $\delta(r, b)$, $\delta(r, bb)$, $\cdots$ *until find an m such that* $\delta(r, b^m) \in F$. *Print out m.*

Since the **only** extension of a^n that is in L_1 is $a^n b^n$, $m = n$. Hence the program prints out n.

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let n be a number such that $C(n)$ is large (we say how large later).

We describe a short machine that prints out n.

This step is preprocessing. Feed a^n into M. It ends in state r.

Key b^n is the **only** string x such that $\delta(r, x) \in F$.

The following program prints out n.
Compute $\delta(r, b)$, $\delta(r, bb)$, $\cdots$ *until find an* m *such that* $\delta(r, b^m) \in F$. *Print out* m.

Since the **only** extension of a^n that is in L_1 is $a^n b^n$, $m = n$. Hence the program prints out n.

What is the length of the program? To describe the program all you need is M (size $O(1)$) and some $O(1)$ code. The program is of size $O(1)$, say A.

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ is Not Regular

Assume L_1 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let n be a number such that $C(n)$ is large (we say how large later).

We describe a short machine that prints out n.

This step is preprocessing. Feed a^n into M. It ends in state r.

Key b^n is the **only** string x such that $\delta(r, x) \in F$.

The following program prints out n.
Compute $\delta(r, b)$, $\delta(r, bb)$, $\cdots$ *until find an* m *such that* $\delta(r, b^m) \in F$. *Print out* m.

Since the **only** extension of a^n that is in L_1 is $a^n b^n$, $m = n$. Hence the program prints out n.

What is the length of the program? To describe the program all you need is M (size $O(1)$) and some $O(1)$ code. The program is of size $O(1)$, say A.

Pick n such that $C(n) > A$. Then you have a program of size $A < C(n)$ printing out n, which is a contradiction.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.
And Now Back to Our Proof

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.
And Now Back to Our Proof

Let i be a number such that $C(p_{i+1} - p_i)$ is large (we say how large later).

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.
And Now Back to Our Proof

Let i be a number such that $C(p_{i+1} - p_i)$ is large (we say how large later).

We describe a short machine that prints out $p_{i+1} - p_i$.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.
And Now Back to Our Proof

Let i be a number such that $C(p_{i+1} - p_i)$ is large (we say how large later).

We describe a short machine that prints out $p_{i+1} - p_i$.

This step is preprocessing. Feed a^{p_i} into M. It ends in state r.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.
And Now Back to Our Proof

Let i be a number such that $C(p_{i+1} - p_i)$ is large (we say how large later).

We describe a short machine that prints out $p_{i+1} - p_i$.

This step is preprocessing. Feed a^{p_i} into M. It ends in state r.

Key 0 $a^{p_i} a^{p_{i+1} - p_i} \in L_2$.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.
And Now Back to Our Proof

Let i be a number such that $C(p_{i+1} - p_i)$ is large (we say how large later).

We describe a short machine that prints out $p_{i+1} - p_i$.

This step is preprocessing. Feed a^{p_i} into M. It ends in state r.

Key 0 $a^{p_i} a^{p_{i+1} - p_i} \in L_2$.

Key 1 $a^{p_i} a^{p_{i+1} - p_i - 1} \notin L_2$.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.
And Now Back to Our Proof

Let i be a number such that $C(p_{i+1} - p_i)$ is large (we say how large later).

We describe a short machine that prints out $p_{i+1} - p_i$.

This step is preprocessing. Feed a^{p_i} into M. It ends in state r.

Key 0 $a^{p_i} a^{p_{i+1} - p_i} \in L_2$.

Key 1 $a^{p_i} a^{p_{i+1} - p_i - 1} \notin L_2$.

Key 2 $a^{p_i} a^{p_{i+1} - p_i - 2} \notin L_2$.

$L_2 = \{a^p : p \text{ is prime}\}$ is Not Regular

Assume L_2 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.
Let p_i be the ith prime.
Lemma For all N there exists i such that $p_{i+1} - p_i \geq N$.
Pf There are no primes between $(N+1)! + 2$ and $(N+1)! + N + 1$.
And Now Back to Our Proof

Let i be a number such that $C(p_{i+1} - p_i)$ is large (we say how large later).

We describe a short machine that prints out $p_{i+1} - p_i$.

This step is preprocessing. Feed a^{p_i} into M. It ends in state r.

Key 0 $a^{p_i} a^{p_{i+1} - p_i} \in L_2$.

Key 1 $a^{p_i} a^{p_{i+1} - p_i - 1} \notin L_2$.

Key 2 $a^{p_i} a^{p_{i+1} - p_i - 2} \notin L_2$.

Real Key $a^{p_{i+1} - p_i}$ is the **shortest** string x such that $a^{p_i} x \in L_2$.

$L_2 = \{a^p : p \text{ is prime }\}$ is Not Regular (cont.)

$L_2 = \{a^p : p$ is prime $\}$ is Not Regular (cont.)

The following program prints out $p_{i+1} - p_i$.
Compute $\delta(r, a)$, $\delta(r, aa)$, $\cdots$ *until find FIRST* m *such that* $\delta(r, a^m) \in F$. *Print out* m.

$L_2 = \{a^p : p$ is prime $\}$ is Not Regular (cont.)

The following program prints out $p_{i+1} - p_i$.
Compute $\delta(r, a)$, $\delta(r, aa)$, $\cdots$ *until find FIRST* m *such that* $\delta(r, a^m) \in F$. *Print out* m.

Since the smallest m such that $a^{p_i+m} \in L_2$ is $p_{i+1} - p_i$, this program will print out

$$p_{i+1} - p_i$$

$L_2 = \{a^p : p$ is prime $\}$ is Not Regular (cont.)

The following program prints out $p_{i+1} - p_i$.
Compute $\delta(r, a)$, $\delta(r, aa)$, $\cdots$ *until find FIRST* m *such that* $\delta(r, a^m) \in F$. *Print out* m.

Since the smallest m such that $a^{p_i + m} \in L_2$ is $p_{i+1} - p_i$, this program will print out

$$p_{i+1} - p_i$$

What is the length of the program? To describe the program all you need is M and some $O(1)$ code. The program is of size $O(1)$, say A.

$L_2 = \{a^p : p \text{ is prime }\}$ is Not Regular (cont.)

The following program prints out $p_{i+1} - p_i$.
Compute $\delta(r, a), \delta(r, aa), \cdots$ *until find FIRST* m *such that* $\delta(r, a^m) \in F$. *Print out* m.

Since the smallest m such that $a^{p_i+m} \in L_2$ is $p_{i+1} - p_i$, this program will print out

$$p_{i+1} - p_i$$

What is the length of the program? To describe the program all you need is M and some $O(1)$ code. The program is of size $O(1)$, say A.

Pick i such that $C(p_{i+1} - p_i) \geq A$. Then you have a program of size $A < C(p_{i+1} - p_i)$ printing out $p_{i+1} - p_i$ which is a contradiction.

$L_3 = \{a^i b^j :$ gcd of i, j is 1 $\}$ is Not Regular

Assume L_3 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

$L_3 = \{a^i b^j :$ gcd of i, j is 1 $\}$ is Not Regular

Assume L_3 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let p be a large prime (we say how large later).

$L_3 = \{a^i b^j : \text{ gcd of } i, j \text{ is } 1\ \}$ is Not Regular

Assume L_3 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let p be a large prime (we say how large later).

We describe a short machine that prints out p.

$L_3 = \{a^i b^j : \text{ gcd of } i, j \text{ is } 1 \}$ is Not Regular

Assume L_3 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let p be a large prime (we say how large later).

We describe a short machine that prints out p.

This step is preprocessing. Feed $a^{(p-1)!}$ into M. It ends in state r.

$L_3 = \{a^i b^j :$ gcd of i, j is 1 $\}$ is Not Regular

Assume L_3 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let p be a large prime (we say how large later).

We describe a short machine that prints out p.

This step is preprocessing. Feed $a^{(p-1)!}$ into M. It ends in state r.

Key 1 $a^{(p-1)!}b \in L_3$ AND $a^{(p-1)!}b^p \in L_3$.

$L_3 = \{a^i b^j : \text{ gcd of } i, j \text{ is } 1\ \}$ is Not Regular

Assume L_3 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let p be a large prime (we say how large later).

We describe a short machine that prints out p.

This step is preprocessing. Feed $a^{(p-1)!}$ into M. It ends in state r.

Key 1 $a^{(p-1)!}b \in L_3$ AND $a^{(p-1)!}b^p \in L_3$.

Key 2 $a^{(p-1)!}b^2 \notin L_3$.

$L_3 = \{a^i b^j :$ gcd of i, j is 1 $\}$ is Not Regular

Assume L_3 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let p be a large prime (we say how large later).

We describe a short machine that prints out p.

This step is preprocessing. Feed $a^{(p-1)!}$ into M. It ends in state r.

Key 1 $a^{(p-1)!}b \in L_3$ AND $a^{(p-1)!}b^p \in L_3$.

Key 2 $a^{(p-1)!}b^2 \notin L_3$.

Key 3 $a^{(p-1)!}b^3 \notin L_3$.

$L_3 = \{a^i b^j : \text{ gcd of } i, j \text{ is } 1\ \}$ is Not Regular

Assume L_3 is regular via $M = (Q, \{a, b\}, \delta, s, F)$.

Let p be a large prime (we say how large later).

We describe a short machine that prints out p.

This step is preprocessing. Feed $a^{(p-1)!}$ into M. It ends in state r.

Key 1 $a^{(p-1)!}b \in L_3$ AND $a^{(p-1)!}b^p \in L_3$.

Key 2 $a^{(p-1)!}b^2 \notin L_3$.

Key 3 $a^{(p-1)!}b^3 \notin L_3$.

Real Key p is the smallest $m \geq 2$ such that $a^{(p-1)!}b^m \in L_3$.

$L_3 = \{a^i b^j : \text{ gcd of } i, j \text{ is } 1\}$ is Not Regular

$L_3 = \{a^i b^j :$ gcd of i, j is 1 $\}$ is Not Regular

The following program prints out p.
Compute $\delta(r, b^2)$, $\delta(r, b^3)$, $\cdots$ until find FIRST $m \geq 2$ such that $\delta(r, b^m) \in F$. Print out m.

$L_3 = \{a^i b^j :$ gcd of i, j is 1 $\}$ is Not Regular

The following program prints out p.
Compute $\delta(r, b^2)$, $\delta(r, b^3)$, $\cdots$ until find FIRST $m \geq 2$ such that $\delta(r, b^m) \in F$. Print out m.
From comments above $m = p$.

$L_3 = \{a^i b^j :$ gcd of i, j is 1 $\}$ is Not Regular

The following program prints out p.
Compute $\delta(r, b^2)$, $\delta(r, b^3)$, $\cdots$ until find FIRST $m \geq 2$ such that $\delta(r, b^m) \in F$. Print out m.
From comments above $m = p$.

What is the length of the program? To describe the program all you need is M and some $O(1)$ code. The program is of size $O(1)$, say A.

$L_3 = \{a^i b^j :$ gcd of i, j is 1 $\}$ is Not Regular

The following program prints out p.

Compute $\delta(r, b^2)$, $\delta(r, b^3)$, $\cdots$ until find FIRST $m \geq 2$ such that $\delta(r, b^m) \in F$. Print out m.

From comments above $m = p$.

What is the length of the program? To describe the program all you need is M and some $O(1)$ code. The program is of size $O(1)$, say A.

Pick prime p such that $C(p) \geq A$. Then you have a program of size $A < C(p)$ printing out p which is a contradiction.

Kolm Complexity Also Applies To

Kolm Complexity Also Applies To

1. Proves that other langs are not regular.

Kolm Complexity Also Applies To

1. Proves that other langs are not regular.
2. Proves that langs are not CFG.

Kolm Complexity Also Applies To

1. Proves that other langs are not regular.
2. Proves that langs are not CFG.
3. Can use it to show some langs require a large DFA, NFA, CFG, TM.

Kolm Complexity Also Applies To

1. Proves that other langs are not regular.
2. Proves that langs are not CFG.
3. Can use it to show some langs require a large DFA, NFA, CFG, TM.
4. Can use in proves of average case analysis. If an algorithm runs in time BLAH on a Kolg random input, then its average case is BLAH.

BILL AND NATHAN STOP RECORDING LECTURE!!!!

BILL AND NATHAN STOP RECORDING LECTURE!!!

UN-TIMED PART OF FINAL IS TUESDAY May 11 11:00A. NO DEAD CAT

Exposition by William Gasarch—U of MD

FINAL IS THURSDAY May 13 8:00PM-10:15PM

Exposition by William Gasarch—U of MD

FILL OUT COURSE EVALS for ALL YOUR COURSES!!!

Exposition by William Gasarch—U of MD