Which game is harder?

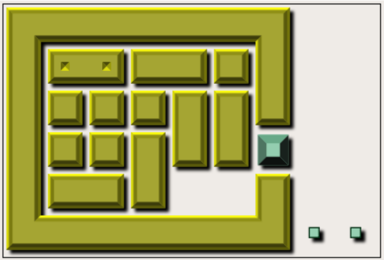
Ramprasad Saptharishi

Chennai Mathematical Institute
Third Year B.Sc Mathematics

September 6, 2006

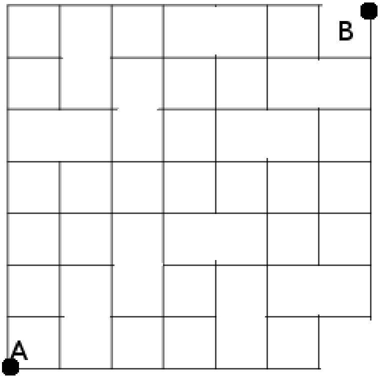
Outline

Rush Hour



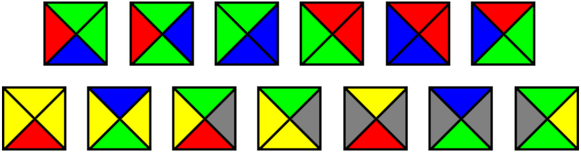
Can you get your car out of the mess...

Paths



How many paths are there from A to B?

The Wang-Tiling Problem



Can they tile the plane “nicely”?

Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Does there exists a consistant solution to this?

Outline

Computational Model

Computational Model

- What do we use to solve the problems?

Computational Model

- What do we use to solve the problems?
- How does one classify hardness?

Computational Model

- What do we use to solve the problems?
- How does one classify hardness?
- *Are there bounds on computation?*

Finite State Machines

- Has finitely many states in it, with a certain start state, and some final states.

Finite State Machines

- Has finitely many states in it, with a certain start state, and some final states.
- Read one letter of the input, look at your present state, move to some state, position head on next input

Finite State Machines

- Has finitely many states in it, with a certain start state, and some final states.
- Read one letter of the input, look at your present state, move to some state, position head on next input
- At the end of the input, if you are in a final state, accept that input. Otherwise reject.

What it can and can't

What it can and can't

Things it **can** do:

What it can and can't

Things it **can** do:

- Does the input end in 01001?

What it can and can't

Things it **can** do:

- Does the input end in 01001?
- Does the input have exactly $1 \mod 3$ zeros in it?

What it can and can't

Things it **can** do:

- Does the input end in 01001?
- Does the input have exactly $1 \mod 3$ zeros in it?
- Is the input devoid of the substring 000 and 111?

What it can and can't

Things it **can** do:

- Does the input end in 01001?
- Does the input have exactly $1 \mod 3$ zeros in it?
- Is the input devoid of the substring 000 and 111?
- *All regular expressions*

What it can and can't

Things it **can** do:

- Does the input end in 01001?
- Does the input have exactly $1 \mod 3$ zeros in it?
- Is the input devoid of the substring 000 and 111?
- *All regular expressions*

Things it **can't** do:

What it can and can't

Things it **can** do:

- Does the input end in 01001?
- Does the input have exactly $1 \mod 3$ zeros in it?
- Is the input devoid of the substring 000 and 111?
- *All regular expressions*

Things it **can't** do:

- Is the input a prime number in binary?

What it can and can't

Things it **can** do:

- Does the input end in 01001?
- Does the input have exactly $1 \mod 3$ zeros in it?
- Is the input devoid of the substring 000 and 111?
- *All regular expressions*

Things it **can't** do:

- Is the input a prime number in binary?
- Is the input a power of 3?

What it can and can't

Things it **can** do:

- Does the input end in 01001?
- Does the input have exactly $1 \mod 3$ zeros in it?
- Is the input devoid of the substring 000 and 111?
- *All regular expressions*

Things it **can't** do:

- Is the input a prime number in binary?
- Is the input a power of 3?
- Is the input of the form $0^n 1^n$?

What it can and can't

Things it **can** do:

- Does the input end in 01001?
- Does the input have exactly $1 \mod 3$ zeros in it?
- Is the input devoid of the substring 000 and 111?
- *All regular expressions*

Things it **can't** do:

- Is the input a prime number in binary?
- Is the input a power of 3?
- Is the input of the form 0^n1^n?
- *Remember*

Pushdown Automata

Pushdown Automata

- Finite state machine + stack

Pushdown Automata

- Finite state machine + stack
- Look at input, look at the stack top, look at your present state, move to new state, modify the top of the stack, move to next letter of the input.

Pushdown Automata

- Finite state machine + stack
- Look at input, look at the stack top, look at your present state, move to new state, modify the top of the stack, move to next letter of the input.
- If you are in a final state at the end, accept the string, else reject.

What it can and can't

What it can and can't

Things this **can** do:

What it can and can't

Things this **can** do:

- Is the input a palindrome?

What it can and can't

Things this **can** do:

- Is the input a palindrome?
- Is the input of the form $0^n 1^n$?

What it can and can't

Things this **can** do:

- Is the input a palindrome?
- Is the input of the form $0^n 1^n$?
- Balanced Paranthesis

What it can and can't

Things this **can** do:

- Is the input a palindrome?
- Is the input of the form 0^n1^n?
- Balanced Paranthesis
- *Context Free Grammars*

Things this **can't** do:

What it can and can't

Things this **can** do:

- Is the input a palindrome?
- Is the input of the form $0^n 1^n$?
- Balanced Paranthesis
- *Context Free Grammars*

Things this **can't** do:

- Is the input of the form $a^n b^n c^n$?

What it can and can't

Things this **can** do:

- Is the input a palindrome?
- Is the input of the form 0^n1^n?
- Balanced Paranthesis
- *Context Free Grammars*

Things this **can't** do:

- Is the input of the form $a^nb^nc^n$?
- Is the input of the form ww?

What it can and can't

Things this **can** do:

- Is the input a palindrome?
- Is the input of the form 0^n1^n?
- Balanced Paranthesis
- *Context Free Grammars*

Things this **can't** do:

- Is the input of the form $a^nb^nc^n$?
- Is the input of the form ww?
- Is the input a representation of a prime number?

Mimicking computation that we usually do

- The decision problem is fixed, and input provided on the board

Mimicking computation that we usually do

- The decision problem is fixed, and input provided on the board
- Pick out the rough book, and start solving.

Mimicking computation that we usually do

- The decision problem is fixed, and input provided on the board
- Pick out the rough book, and start solving.
- Do we work like a PDA? Or are we doing something more?

Mimicking computation that we usually do

- The decision problem is fixed, and input provided on the board
- Pick out the rough book, and start solving.
- Do we work like a PDA? Or are we doing something more?Oh yes... we can have multiple looks at the input, unlike a PDA!

The Turing Machine, finally

The Turing Machine, finally

- Has a finite state control, input tape, workspace.

The Turing Machine, finally

- Has a finite state control, input tape, workspace.
- Look at the head position on the input, look at the head position on the workspace, your present state, modify the workspace or input tape if you desire, move the head positions left or right.

The Turing Machine, finally

- Has a finite state control, input tape, workspace.
- Look at the head position on the input, look at the head position on the workspace, your present state, modify the workspace or input tape if you desire, move the head positions left or right.
- Accept or reject?

The Turing Machine, finally

- Has a finite state control, input tape, workspace.
- Look at the head position on the input, look at the head position on the workspace, your present state, modify the workspace or input tape if you desire, move the head positions left or right.
- Accept or reject? Label two special states as *ACCEPT* and *REJECT* and act accordingly once you reach these states.

The Turing Machine, finally

- Has a finite state control, input tape, workspace.
- Look at the head position on the input, look at the head position on the workspace, your present state, modify the workspace or input tape if you desire, move the head positions left or right.
- Accept or reject? Label two special states as *ACCEPT* and *REJECT* and act accordingly once you reach these states.
- What if you never reach them?

The Turing Machine, finally

- Has a finite state control, input tape, workspace.
- Look at the head position on the input, look at the head position on the workspace, your present state, modify the workspace or input tape if you desire, move the head positions left or right.
- Accept or reject? Label two special states as *ACCEPT* and *REJECT* and act accordingly once you reach these states.
- What if you never reach them? That's the programmer's headache...

The Turing Machine, finally

- Has a finite state control, input tape, workspace.
- Look at the head position on the input, look at the head position on the workspace, your present state, modify the workspace or input tape if you desire, move the head positions left or right.
- Accept or reject? Label two special states as *ACCEPT* and *REJECT* and act accordingly once you reach these states.
- What if you never reach them? That's the programmer's headache...

But what's the "finally" doing in the title?

“More powerful” Machines

“More powerful” Machines

- PDA with 2 stacks

“More powerful” Machines

- PDA with 2 stacks
- PDA with counters

"More powerful" Machines

- PDA with 2 stacks
- PDA with counters
- Turing machines with 171283 workspace tapes

“More powerful” Machines

- PDA with 2 stacks
- PDA with counters
- Turing machines with 171283 workspace tapes
- RAM model turing machines

"More powerful" Machines

- PDA with 2 stacks
- PDA with counters
- Turing machines with 171283 workspace tapes
- RAM model turing machines
- *P*4 machine with MS Windows Vista installed

"More powerful" Machines

- PDA with 2 stacks
- PDA with counters
- Turing machines with 171283 workspace tapes
- RAM model turing machines
- $P4$ machine with MS Windows Vista installed

None of them are more powerful than a Turing Machine.

"More powerful" Machines

- PDA with 2 stacks
- PDA with counters
- Turing machines with 171283 workspace tapes
- RAM model turing machines
- $P4$ machine with MS Windows Vista installed

None of them are more powerful than a Turing Machine. The first 4 are as powerful as a Turing machine, the 5th is probably worse than a finite state machine.

Outline

Computable Functions

Computable functions are those that can be computed by a halting Turing Machine

Computable Functions

Computable functions are those that can be computed by a halting Turing Machine

Are there functions that a Turing Machine **cannot** compute?

Existance of undecidable problems

Existance of undecidable problems

- Turing Machine descriptions are of finite length

Existance of undecidable problems

- Turing Machine descriptions are of finite length
- Countably many turing machines only

Existance of undecidable problems

- Turing Machine descriptions are of finite length
- Countably many turing machines only
- Only countably many computable functions!

Existance of undecidable problems

- Turing Machine descriptions are of finite length
- Countably many turing machines only
- Only countably many computable functions!
- Boolean functions correspond to subsets of strings, uncountable.

Existance of undecidable problems

- Turing Machine descriptions are of finite length
- Countably many turing machines only
- Only countably many computable functions!
- Boolean functions correspond to subsets of strings, uncountable.
- Clearly shows existance of undecidable problems

Existance of undecidable problems

- Turing Machine descriptions are of finite length
- Countably many turing machines only
- Only countably many computable functions!
- Boolean functions correspond to subsets of strings, uncountable.
- Clearly shows existance of undecidable problems

Doh...a mathematician's proof, is there a constructive proof?

The *Osama-Bush* problem

Problem: You are given as input the code of a turing machine M. Suppose M is run on its own code as input, its output can either end with "*Osama*" or not. If M's output ends with "*Osama*", you have to just print "*Bush*" and halt. Othewise, you have to print "*Osama*" and halt.

The *Osama-Bush* problem

Problem: You are given as input the code of a turing machine M. Suppose M is run on its own code as input, its output can either end with "*Osama*" or not. If M's output ends with "*Osama*", you have to just print "*Bush*" and halt. Othewise, you have to print "*Osama*" and halt.

Suppose this problem was computable by a Turing machine OB, run OB on its own code,

The *Osama-Bush* problem

Problem: You are given as input the code of a turing machine M. Suppose M is run on its own code as input, its output can either end with "*Osama*" or not. If M's output ends with "*Osama*", you have to just print "*Bush*" and halt. Othewise, you have to print "*Osama*" and halt.

Suppose this problem was computable by a Turing machine OB, run OB on its own code, . . . contradiction! □

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

Proof.

Suppose there exists a machine PF that can actually check if the input machine has the property $\mathcal{P}$.

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

Proof.

Suppose there exists a machine PF that can actually check if the input machine has the property $\mathcal{P}$.Since $\mathcal{P}$ is non-trivial, assume some $L(M_0) \in \mathcal{P}$ and ϕ in the complement.

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

Proof.

Suppose there exists a machine PF that can actually check if the input machine has the property $\mathcal{P}$.Since $\mathcal{P}$ is non-trivial, assume some $L(M_0) \in \mathcal{P}$ and ϕ in the complement.
Taking a machine M as input, create the following machine:

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

Proof.

Suppose there exists a machine PF that can actually check if the input machine has the property $\mathcal{P}$.Since $\mathcal{P}$ is non-trivial, assume some $L(M_0) \in \mathcal{P}$ and ϕ in the complement.
Taking a machine M as input, create the following machine:

- Takes as input a string w

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

Proof.

Suppose there exists a machine PF that can actually check if the input machine has the property $\mathcal{P}$.Since $\mathcal{P}$ is non-trivial, assume some $L(M_0) \in \mathcal{P}$ and ϕ in the complement.
Taking a machine M as input, create the following machine:

- Takes as input a string w
- Run M on its code, until it prints "osama" in the end

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

Proof.

Suppose there exists a machine PF that can actually check if the input machine has the property $\mathcal{P}$.Since $\mathcal{P}$ is non-trivial, assume some $L(M_0) \in \mathcal{P}$ and ϕ in the complement.
Taking a machine M as input, create the following machine:

- Takes as input a string w
- Run M on its code, until it prints "osama" in the end
- Run M_0 on w and accept/reject based on the outcome

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

Proof.

Suppose there exists a machine PF that can actually check if the input machine has the property $\mathcal{P}$.Since $\mathcal{P}$ is non-trivial, assume some $L(M_0) \in \mathcal{P}$ and ϕ in the complement.
Taking a machine M as input, create the following machine:

- Takes as input a string w
- Run M on its code, until it prints "osama" in the end
- Run M_0 on w and accept/reject based on the outcome

The language accepted by this machine is ϕ if M doesn't print "osama" in the end, and the language accepted by this is $L(M_0)$ if it does print "osama".

All Interesting Problems are Undecidable!

Theorem (Rice's Theorem)

For any non-trivial property $\mathcal{P}$ of languages, checking if the language accepted by the input (interpretted as a machine code) has the property $\mathcal{P}$ is undecidable!

Proof.

Suppose there exists a machine PF that can actually check if the input machine has the property $\mathcal{P}$.Since $\mathcal{P}$ is non-trivial, assume some $L(M_0) \in \mathcal{P}$ and ϕ in the complement.
Taking a machine M as input, create the following machine:

- Takes as input a string w
- Run M on its code, until it prints "osama" in the end
- Run M_0 on w and accept/reject based on the outcome

The language accepted by this machine is ϕ if M doesn't print "osama" in the end, and the language accepted by this is $L(M_0)$ if it does print "osama". Pass this through the PF, and it would solve the "osama-bush" problem! □

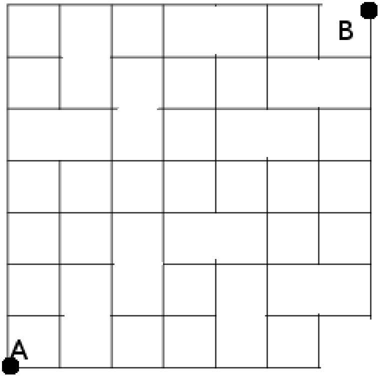
Outline

Back to the games

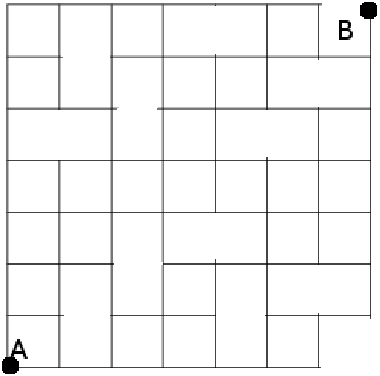
Non-decreasing order of difficulty

- Paths
- Sudoku
- Rush Hour
- The Wang-Tiling Problem

Paths

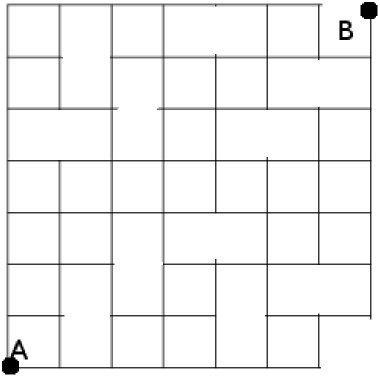


Paths



Dynamic programming solves it.

Paths



Dynamic programming solves it.

Definition

P is the class of problems that can be solved by a Turing Machine whose running time is bounded by some fixed polynomial in the size of the input

Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Solutions are "short" and checking the correctness of a claimed solution is easy.

Sudoku

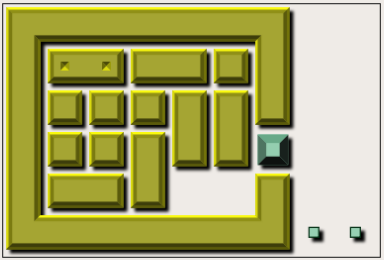
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Solutions are "short" and checking the correctness of a claimed solution is easy.

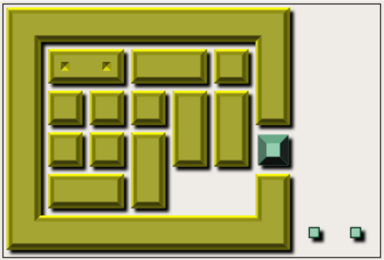
Definition

NP is the class of problems that have a short witness. Formally, if $L \in NP$ for all $x \in L$, thre exists a "short" y such that checking that y is a solution for x is checkable in P.

Rush Hour

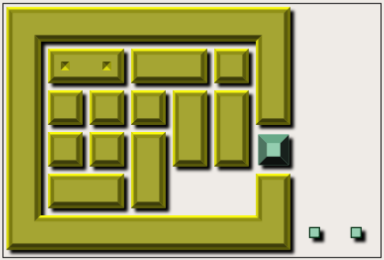


Rush Hour



...given a black board of sufficient size, I can explain the solution.

Rush Hour

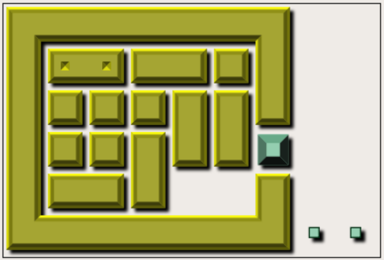


...given a black board of sufficient size, I can explain the solution.

Definition

PSPACE is the class of problems that can be solved by a Turing Machine whose space is bounded by some fixed polynomial in the size of the input

Rush Hour



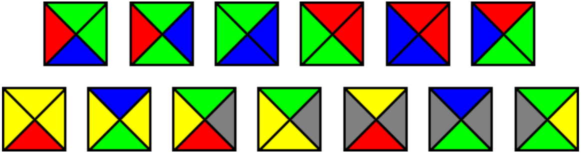
...given a black board of sufficient size, I can explain the solution.

Definition

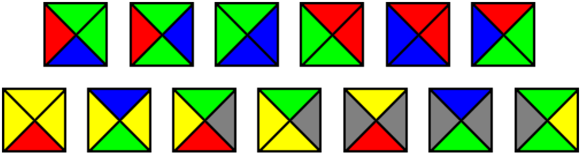
PSPACE is the class of problems that can be solved by a Turing Machine whose space is bounded by some fixed polynomial in the size of the input

This class also corresponds to *interactive proofs*

The Wang-Tiling Problem

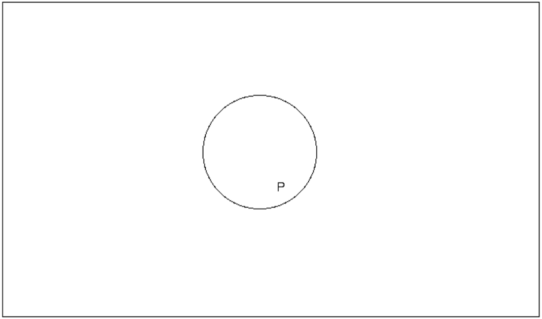


The Wang-Tiling Problem

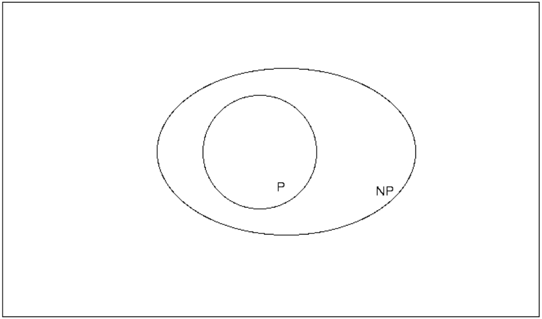


… no hope. Cannot be solved by any Turing machine that halts on all inputs.

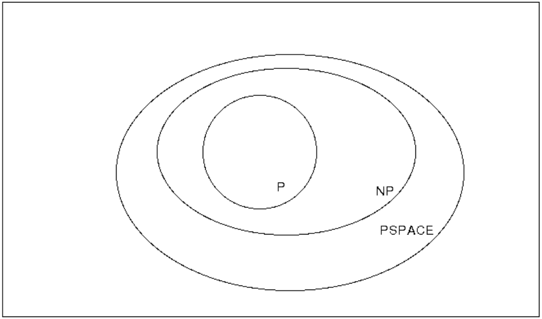
Containment of the classes



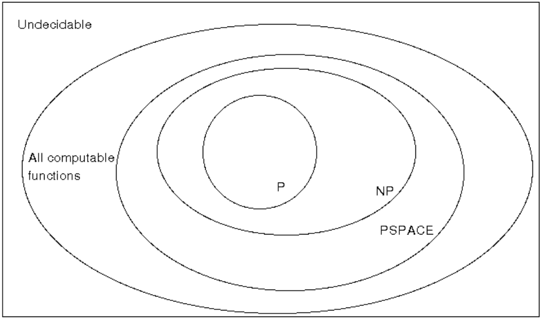
Containment of the classes



Containment of the classes



Containment of the classes



Are the containments strict?

Why do we believe that the containments are strict?

Are the containments strict?

Why do we believe that the containments are strict?

- It is a mathematical possibility

Are the containments strict?

Why do we believe that the containments are strict?

- It is a mathematical possibility
- None of us know at the moment

Are the containments strict?

Why do we believe that the containments are strict?

- It is a mathematical possibility
- None of us know at the moment
- Intuitively we believe that checking solutions is much easier than actually finding one. ($P \neq NP$)

Are the containments strict?

Why do we believe that the containments are strict?

- It is a mathematical possibility
- None of us know at the moment
- Intuitively we believe that checking solutions is much easier than actually finding one. ($P \neq NP$)
- Convincing someone with a witness is intuitively much easier than convincing someone with an interactive session. ($NP \neq PSPACE$)

Outline

Billy Madison Revisited

Billy Madison Revisited

- Bush and Osama decide to battle it out with a memory contest

Billy Madison Revisited

- Bush and Osama decide to battle it out with a memory contest
- They are given an infinite string to memorize, and they have to exhibit their talent.

Billy Madison Revisited

- Bush and Osama decide to battle it out with a memory contest
- They are given an infinite string to memorize, and they have to exhibit their talent.
- Bush might cheat by fixing the string that is easy to memorize for him. Our job is to make the game fair.

Billy Madison Revisited

- Bush and Osama decide to battle it out with a memory contest
- They are given an infinite string to memorize, and they have to exhibit their talent.
- Bush might cheat by fixing the string that is easy to memorize for him. Our job is to make the game fair.

How does one create a string that is hard to remember?

Randomness of Strings

Randomness of Strings

Strings are "hard" to remember if they are random

Randomness of Strings

Strings are "hard" to remember if they are random

What does it mean to say a certain string is "random"?

Possible Definitions

Attempt 1:
For every N, the first N bits of the string should contain roughly equal number of 0s and 1s.

Possible Definitions

Attempt 1:
For every N, the first N bits of the string should contain roughly equal number of 0s and 1s. Is 01010101010... a random string???

Possible Definitions

Attempt 1:
For every N, the first N bits of the string should contain roughly equal number of 0s and 1s. Is 01010101010... a random string???
Ah, it didn't contain substrings like 11, maybe that was the problem.

Attempt 2:
All possible sequences of length N must occur in the string with roughly the same probability.

Possible Definitions

Attempt 1:
For every N, the first N bits of the string should contain roughly equal number of 0s and 1s. Is 01010101010... a random string???
Ah, it didn't contain substrings like 11, maybe that was the problem.

Attempt 2:
All possible sequences of length N must occur in the string with roughly the same probability.
Champ's Sequence: 012345678910111213... has that property. This of course can't be called a "random" sequence!

What sort of definition could it be?

What sort of definition could it be?

- Random $\Leftrightarrow$ Hard to remember

What sort of definition could it be?

- Random $\Leftrightarrow$ Hard to remember
- Patterns in strings $\Rightarrow$ easy to remember

What sort of definition could it be?

- Random $\Leftrightarrow$ Hard to remember
- Patterns in strings $\Rightarrow$ easy to remember
- $\therefore$ Random strings are “void” of patterns

What sort of definition could it be?

- Random $\Leftrightarrow$ Hard to remember
- Patterns in strings $\Rightarrow$ easy to remember
- $\therefore$ Random strings are “void” of patterns
- There shouldn’t be an easier way to remember the string other than actually memorizing it bit by bit

What sort of definition could it be?

- Random $\Leftrightarrow$ Hard to remember
- Patterns in strings $\Rightarrow$ easy to remember
- $\therefore$ Random strings are "void" of patterns
- There shouldn't be an easier way to remember the string other than actually memorizing it bit by bit

How does one formalize this notion?

Chaitin-Kolmogorov Randomness

Definition

A string x is said to be CK-random if the any halting turing machine that outputs x has description length atleast x. The length of the description shortest halting turing machine that outputs x is called the Kolmogorov Complexity of x.

Chaitin-Kolmogorov Randomness

Definition

A string x is said to be CK-random if the any halting turing machine that outputs x has description length atleast x. The length of the description shortest halting turing machine that outputs x is called the Kolmogorov Complexity of x.

Any program that outputs x should more or less have a line print(x) in it, this making the program very long.

Strings that are **not** CK-random

- 0101010...: trivial short program

Strings that are **not** CK-random

- 0101010...: trivial short program
- Champ's Sequence: again, small program.

Strings that are **not** CK-random

- 0101010...: trivial short program
- Champ's Sequence: again, small program.
- Binary representation of e: Taylor series

Strings that are **not** CK-random

- 0101010...: trivial short program
- Champ's Sequence: again, small program.
- Binary representation of e: Taylor series
- Binary representation of π: Fourier series

Do there exist CK-random strings at all?

Do there exist CK-random strings at all?

Yes there are...

Do there exist CK-random strings at all?

Yes there are...
There are atmost $1 + 2 + 2^2 + \ldots 2^{n-1} = 2^n - 1$ programs of length less than n hence there is atleast one sting of length n of Kolmogorov complexity atleast n.

Do there exist CK-random strings at all?

Yes there are...
There are atmost $1 + 2 + 2^2 + \dots 2^{n-1} = 2^n - 1$ programs of length less than n hence there is atleast one sting of length n of Kolmogorov complexity atleast n.

Doh... mathematician's proof again, is there a more constructive proof?

Do there exist CK-random strings at all?

Yes there are...
There are atmost $1 + 2 + 2^2 + \ldots 2^{n-1} = 2^n - 1$ programs of length less than n hence there is atleast one sting of length n of Kolmogorov complexity atleast n.

Doh... mathematician's proof again, is there a more constructive proof?
Seems unlikely since it is possible to describe the string in a short manner, then it's Kolmogorov Complexity would be less.

Do there exist CK-random strings at all?

Yes there are...
There are atmost $1 + 2 + 2^2 + \ldots 2^{n-1} = 2^n - 1$ programs of length less than n hence there is atleast one sting of length n of Kolmogorov complexity atleast n.

Doh... mathematician's proof again, is there a more constructive proof?
Seems unlikely since it is possible to describe the string in a short manner, then it's Kolmogorov Complexity would be less... but we can still get around it.

Some Changes

- Instead of the Kolmogorov complexity being atleast n, let us weaken this to $n - c$ for some constant c.

Some Changes

- Instead of the Kolmogorov complexity being atleast n, let us weaken this to $n - c$ for some constant c.
- And assume that turing machine descriptions are prefix-free: if x is avalid turing machine description, then $x.y$ cannot be a valid description.

Some Changes

- Instead of the Kolmogorov complexity being atleast n, let us weaken this to $n - c$ for some constant c.
- And assume that turing machine descriptions are prefix-free: if x is avalid turing machine description, then $x.y$ cannot be a valid description. This is like putting a *STOP* command at the end of the program.

The Waiting Time Constant

The Waiting Time Constant

Let

$$WTC = \sum_{P} 2^{-|P|}$$

summed over all halting turing machines P that have "osama" in the end of the output. Of course, the value largely depends on the machine encoding.

The Waiting Time Constant

Let

$$WTC = \sum_P 2^{-|P|}$$

summed over all halting turing machines P that have "osama" in the end of the output. Of course, the value largely depends on the machine encoding.
Is this even a sequence?

The Waiting Time Constant

Let

$$WTC = \sum_P 2^{-|P|}$$

summed over all halting turing machines P that have "osama" in the end of the output. Of course, the value largely depends on the machine encoding.
Is this even a sequence? By Kraft's inequality we have $0 < WTC < 1$

No short program?

What must be done to construct this?

- Enumerate all programs

No short program?

What must be done to construct this?

- Enumerate all programs
- Check if it prints "osama" in the end.

No short program?

What must be done to construct this?

- Enumerate all programs
- Check if it prints “osama” in the end.
- If it does, then add $2^{-|P|}$ to the sum

No short program?

What must be done to construct this?

- Enumerate all programs
- Check if it prints "osama" in the end.
- If it does, then add $2^{-|P|}$ to the sum

Isn't this a short program?

No short program?

What must be done to construct this?

- Enumerate all programs
- Check if it prints "osama" in the end.
- If it does, then add $2^{-|P|}$ to the sum

Isn't this a short program?

Ah... The *Osama-Bush* problem is not computable...

What makes the *WTC* so powerful?

Claim:
WTC is (weak) CK-random

What makes the *WTC* so powerful?

Claim:
WTC is (weak) CK-random
With the first n bits of the *WTC*, you can figure out which Turing machines of description less than n halt and which do not. Infact one can write a program to do it.

Proof of Randomness

Suppose $WTCGen(n)$ is a program generating the first n bits of WTC.

Proof of Randomness

Suppose $WTCGen(n)$ is a program generating the first n bits of WTC.

- Get the first n bits of WTC using $WTCGen$ as a subroutine

Proof of Randomness

Suppose $WTCGen(n)$ is a program generating the first n bits of WTC.

- Get the first n bits of WTC using $WTCGen$ as a subroutine
- Using the bits, find out all Turing machines of length atmost n that halt.

Proof of Randomness

Suppose *WTCGen*(n) is a program generating the first n bits of *WTC*.

- Get the first n bits of *WTC* using *WTCGen* as a subroutine
- Using the bits, find out all Turing machines of length atmost n that halt.
- Simulate the halting turing machines, and find the lexicographically least string of length n that is not outputted by any of them.

Proof of Randomness

Suppose $WTCGen(n)$ is a program generating the first n bits of WTC.

- Get the first n bits of WTC using $WTCGen$ as a subroutine
- Using the bits, find out all Turing machines of length atmost n that halt.
- Simulate the halting turing machines, and find the lexicographically least string of length n that is not outputted by any of them.
- Print this string

Proof of Randomness

Suppose $WTCGen(n)$ is a program generating the first n bits of WTC.

- Get the first n bits of WTC using $WTCGen$ as a subroutine
- Using the bits, find out all Turing machines of length atmost n that halt.
- Simulate the halting turing machines, and find the lexicographically least string of length n that is not outputted by any of them.
- Print this string

Clearly, the output has Kolmogorov Complexity $\geq n$

Proof of Randomness

Suppose $WTCGen(n)$ is a program generating the first n bits of WTC.

- Get the first n bits of WTC using $WTCGen$ as a subroutine
- Using the bits, find out all Turing machines of length atmost n that halt.
- Simulate the halting turing machines, and find the lexicographically least string of length n that is not outputted by any of them.
- Print this string

Clearly, the output has Kolmogorov Complexity $\geq n$, hence this program has to be of length $\geq n$.

Proof of Randomness

Suppose $WTCGen(n)$ is a program generating the first n bits of WTC.

- Get the first n bits of WTC using $WTCGen$ as a subroutine
- Using the bits, find out all Turing machines of length atmost n that halt.
- Simulate the halting turing machines, and find the lexicographically least string of length n that is not outputted by any of them.
- Print this string

Clearly, the output has Kolmogorov Complexity $\geq n$, hence this program has to be of length $\geq n$. If the main routine takes c bits, then the subroutine $WTCGen$ must be atleast $n - c$ bits long! □

Finding machines that halt

Finding machines that halt

- Pad every machine P with a print "osama" to get a machine P'

Finding machines that halt

- Pad every machine P with a print "osama" to get a machine P'
- Let WTC_n be the truncated version of WTC, and let WTC^s be the same thing summed over turing machines that halt and have "osama" in the end, within s steps.

Finding machines that halt

- Pad every machine P with a print "osama" to get a machine P'
- Let WTC_n be the truncated version of WTC, and let WTC^s be the same thing summed over turing machines that halt and have "osama" in the end, within s steps. WTC^s can be computed given s.

Finding machines that halt

- Pad every machine P with a print "osama" to get a machine P'
- Let WTC_n be the truncated version of WTC, and let WTC^s be the same thing summed over turing machines that halt and have "osama" in the end, within s steps. WTC^s can be computed given s.
- Find the smallest s such that $WTC^s > WTC_n$ where $|P'| = n$.

Finding machines that halt

- Pad every machine P with a print "osama" to get a machine P'
- Let WTC_n be the truncated version of WTC, and let WTC^s be the same thing summed over turing machines that halt and have "osama" in the end, within s steps. WTC^s can be computed given s.
- Find the smallest s such that $WTC^s > WTC_n$ where $|P'| = n$.

Claim:
A machine P halts if and only if the padded machine halts and prints "osama" within s steps.

Finding machines that halt

- Pad every machine P with a print "osama" to get a machine P'
- Let WTC_n be the truncated version of WTC, and let WTC^s be the same thing summed over turing machines that halt and have "osama" in the end, within s steps. WTC^s can be computed given s.
- Find the smallest s such that $WTC^s > WTC_n$ where $|P'| = n$.

Claim:
A machine P halts if and only if the padded machine halts and prints "osama" within s steps.

Proof.
We know that $WTC < WTC_n < WTC + \frac{1}{2^n}$.

Finding machines that halt

- Pad every machine P with a print "osama" to get a machine P'
- Let WTC_n be the truncated version of WTC, and let WTC^s be the same thing summed over turing machines that halt and have "osama" in the end, within s steps. WTC^s can be computed given s.
- Find the smallest s such that $WTC^s > WTC_n$ where $|P'| = n$.

Claim:
A machine P halts if and only if the padded machine halts and prints "osama" within s steps.

Proof.
We know that $WTC < WTC_n < WTC + \frac{1}{2^n}$. Suppose P was a halting machine but P' doesn't halt in less than s steps, then we would have
$WTC > WTC^s + \frac{1}{2^n} > WTC_n + \frac{1}{2^n}$, a contradiction. □

Why WTC?

Why WTC?

- Bush, Osama, WTC

Why WTC?

- Bush, Osama, WTC
- *WTC* is the probability that an infinitely long string halts and prints "osama" in the end.

Why WTC?

- Bush, Osama, WTC
- *WTC* is the probability that an infinitely long string halts and prints "osama" in the end.
- When the sum was just over halting turing machines, the constant is called the Chaitin's Omega, or the halting probability.

Why WTC?

- Bush, Osama, WTC
- *WTC* is the probability that an infinitely long string halts and prints "osama" in the end.
- When the sum was just over halting turing machines, the constant is called the Chaitin's Omega, or the halting probability. Of course, the value varies with different machine encodings.

Robustness of definition

Possible definition 2**:** Every substring of length n must occur with more or less equal probability in x.

Robustness of definition

Possible definition 2**:** Every substring of length n must occur with more or less equal probability in x.

For every non-computable function P, the associated constant satisfies this property.

Thank You

Slides and TeX files are available at
`~ramprasad/studenttalks/hardergame/hardergame.pdf`.
Looking forward for your comments and suggestions