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!!!

Review for Final

Rules

1. **Begin** Final ON Gradescope: Tuesday May 13, 8:00PM-10:15PM. (IF this is a problem for you contact me ASAP!!)

Rules

1. **Begin** Final ON Gradescope: Tuesday May 13, 8:00PM-10:15PM. (IF this is a problem for you contact me ASAP!!)
2. **Resources** Final is open-everything. The web, my notes, my HW solutions, all fine to use. Cannot ask any other human. Can talk to your cat. Honor System. I trust and **respect** you (how much do I respect you–see next slide).

Rules

1. **Begin** Final ON Gradescope: Tuesday May 13, 8:00PM-10:15PM. (IF this is a problem for you contact me ASAP!!)
2. **Resources** Final is open-everything. The web, my notes, my HW solutions, all fine to use. Cannot ask any other human. Can talk to your cat. Honor System. I trust and **respect** you (how much do I respect you–see next slide).
3. **Caveat** You must hand in your own work and you must understand what you hand in.

Rules

1. **Begin** Final ON Gradescope: Tuesday May 13, 8:00PM-10:15PM. (IF this is a problem for you contact me ASAP!!)
2. **Resources** Final is open-everything. The web, my notes, my HW solutions, all fine to use. Cannot ask any other human. Can talk to your cat. Honor System. I trust and **respect** you (how much do I respect you–see next slide).
3. **Caveat** You must hand in your own work and you must understand what you hand in.
4. **Warning** Mindlessly copying does not work.

Rules

1. **Begin** Final ON Gradescope: Tuesday May 13, 8:00PM-10:15PM. (IF this is a problem for you contact me ASAP!!)
2. **Resources** Final is open-everything. The web, my notes, my HW solutions, all fine to use. Cannot ask any other human. Can talk to your cat. Honor System. I trust and **respect** you (how much do I respect you–see next slide).
3. **Caveat** You must hand in your own work and you must understand what you hand in.
4. **Warning** Mindlessly copying does not work.
5. **Neat** LaTex is best. Good handwriting okay. Draw Aut, or use LateX tool posted.

Rules

1. **Begin** Final ON Gradescope: Tuesday May 13, 8:00PM-10:15PM. (IF this is a problem for you contact me ASAP!!)
2. **Resources** Final is open-everything. The web, my notes, my HW solutions, all fine to use. Cannot ask any other human. Can talk to your cat. Honor System. I trust and **respect** you (how much do I respect you–see next slide).
3. **Caveat** You must hand in your own work and you must understand what you hand in.
4. **Warning** Mindlessly copying does not work.
5. **Neat** LaTex is best. Good handwriting okay. Draw Aut, or use LateX tool posted.
6. **Our Intent** This is exam I intended to give out originally. The extra time is meant for you to format and put in LaTeX.

Rules

1. **Begin** Final ON Gradescope: Tuesday May 13, 8:00PM-10:15PM. (IF this is a problem for you contact me ASAP!!)
2. **Resources** Final is open-everything. The web, my notes, my HW solutions, all fine to use. Cannot ask any other human. Can talk to your cat. Honor System. I trust and **respect** you (how much do I respect you–see next slide).
3. **Caveat** You must hand in your own work and you must understand what you hand in.
4. **Warning** Mindlessly copying does not work.
5. **Neat** LaTex is best. Good handwriting okay. Draw Aut, or use LateX tool posted.
6. **Our Intent** This is exam I intended to give out originally. The extra time is meant for you to format and put in LaTeX.
7. **Scope of the Exam**

   Short Answer HWs and lectures.

   Long Answer This Presentation.

RESPECT

I RESPECT you as much as

RESPECT

I RESPECT you as much as

we all RESPECT the hardness of proving lower bounds.

RESPECT

I RESPECT you as much as

we all RESPECT the hardness of proving lower bounds.

(Thats a lot!)

Turing Machines

1. For this review we omit definitions and conventions.
2. There is a JAVA program for function f iff there is a TM that computes f.
3. Everything computable can be done by a TM.

Decidable Sets

Def A set A is DECIDABLE if there is a Turing Machine M such that

$$x \in A \rightarrow M(x) = Y$$

$$x \notin A \rightarrow M(x) = N$$

Terrible Def of DTIME

Def Let $T(n)$ be a computable function (think increasing). A is in $\mathrm{DTIME}(T(n))$ if there is a TM M that decides A and also, for all x, $M(x)$ halts in time $\leq O(T(|x|))$.
Terrible Def since depends to much on machine model.

- Prove theorems about $\mathrm{DTIME}(T(n))$ where the model does not matter. (Time hierarchy theorem–We did not do this)).
- Define time classes that are model-independent (P, NP stuff)

P, NP, Reductions

P and EXP

Def

1. $\mathrm{P} = \mathrm{DTIME}(n^{O(1)})$.
2. $\mathrm{EXP} = \mathrm{DTIME}(2^{n^{O(1)}})$.
3. PF is the set of **functions** that are computable in poly time.

NP

Def A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x \mid (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

NP

Def A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x \mid (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

Intuition. Let $A \in \mathrm{NP}$.

NP

Def A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x \mid (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

Intuition. Let $A \in \mathrm{NP}$.

- If $x \in A$ then there is a SHORT (poly in $|x|$) proof of this fact, namely y, such that x can be VERIFIED in poly time. So if I wanted to convince you that $x \in L$, I could give you y. You can verify $(x, y) \in B$ easily and be convinced.

NP

Def A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x \mid (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

Intuition. Let $A \in \mathrm{NP}$.

- If $x \in A$ then there is a SHORT (poly in $|x|$) proof of this fact, namely y, such that x can be VERIFIED in poly time. So if I wanted to convince you that $x \in L$, I could give you y. You can verify $(x, y) \in B$ easily and be convinced.
- If $x \notin A$ then there is NO short verifiable proof that $x \in A$.

Examples of Sets in NP

$$\mathrm{SAT} = \{\phi : (\exists \vec{y})[\phi(\vec{y}) = T]\}$$

$$\mathrm{3COL} = \{G : G \text{ is 3-colorable } \}$$

$$\mathrm{CLIQ} = \{(G,k) : G \text{ has a clique of size } k\}$$

$$HAM = \{G : G \text{ has a Hamiltonian Cycle}\}$$

$$EUL = \{G : G \text{ has an Eulerian Cycle}\}$$

Note These all ask if something EXISTS. To FIND the (say) 3-coloring one can make queries to (say) 3COL.
Note $EUL \in P$. The rest are NPC hence likely NOT in P.

Reductions

Def Let X, Y be languages. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

We express this by writing $X \leq Y$.

Reductions

Def Let X, Y be languages. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

We express this by writing $X \leq Y$.

Reductions are transitive.
Easy Lemma If $X \leq Y$ and $Y \in \mathrm{P}$ then $X \in \mathrm{P}$.

Reductions

Def Let X, Y be languages. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

We express this by writing $X \leq Y$.

Reductions are transitive.
Easy Lemma If $X \leq Y$ and $Y \in \mathrm{P}$ then $X \in \mathrm{P}$.
Contrapositive If $X \leq Y$ and $X \notin \mathrm{P}$ then $Y \notin \mathrm{P}$.

Def of NP-Complete

Def A language Y is NP**-complete**

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \leq Y$.

Def of NP-Complete

Def A language Y is NP**-complete**

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \leq Y$.

Easy Lemma If Y is NP-complete and $Y \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

Thoughts on this:

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. We did this proof. It involved looking at 1-tape, 1-head, Turing Machines. It had to since SAT was the the **first** NP-complete problem. You could not use some other problem.

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. We did this proof. It involved looking at 1-tape, 1-head, Turing Machines. It had to since SAT was the the **first** NP-complete problem. You could not use some other problem.
2. Once we have SAT is NP-complete we NEVER used TMs again. To show Y NP-complete: (1) $Y \in NP$, (2) $\mathrm{SAT} \le Y$. Might use some other known NPC problem rather than SAT.

SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. We did this proof. It involved looking at 1-tape, 1-head, Turing Machines. It had to since SAT was the the **first** NP-complete problem. You could not use some other problem.
2. Once we have SAT is NP-complete we NEVER used TMs again. To show Y NP-complete: (1) $Y \in NP$, (2) $\mathrm{SAT} \le Y$. Might use some other known NPC problem rather than SAT.
3. Thousands of problems are NP-complete. If any are in P then they are all in P.

The Cook-Levin Thm

What does the Proof Involve

Proof involved coding a TM into a Boolean Formula which had parts:

1. $z_{i,j,\sigma} = T$ iff the jth symbol in the ith configuration is σ.
2. First config: input x, start state, SOME y of the right length.
3. Last config: accepts
4. C_{i+1} follows from C_i.

Closure of P

Easy Closure Properties of P

Assume $L_1, L_2 \in P$.

1. $L_1 \cup L_2 \in P$. EASY. Uses polys closed under addition.
2. $L_1 \cap L_2 \in P$. EASY. Uses polys closed under addition.
3. $\overline{L_1} \in P$. EASY.
4. $L_1 L_2 \in P$. EASY. Uses $p(n)$ poly then $np(n)$ poly.

Closure of P Under *

Thm If $L \in \mathrm{P}$ then $L^* \in \mathrm{P}$.
Proof
First lets talk about what you **should not** do:
The technique of looking at **all** ways to break up x into pieces takes roughly 2^n steps, so we need to do something clever.

Dyn Prog

Dynamic Programming We solve a harder problem but get lots of information in the process.

Dyn Prog

Dynamic Programming We solve a harder problem but get lots of information in the process.

Original Problem Given $x = x_1 \cdots x_n$ want to know if $x \in L^*$

Dyn Prog

Dynamic Programming We solve a harder problem but get lots of information in the process.

Original Problem Given $x = x_1 \cdots x_n$ want to know if $x \in L^*$

New Problem Given $x = x_1 \cdots x_n$ want to know:

$e \in L^*$

$x_1 \in L^*$

$x_1 x_2 \in L^*$

$\vdots$

$x_1 x_2 \cdots x_n \in L^*$.

Intuition $x_1 \cdots x_i \in L^*$ IFF it can be broken into TWO pieces, the first one in L^*, and the second in L.

Final Algorithm

$A[i]$ stores if $x_1 \cdots x_i$ is in L^*. M is poly-time Alg for L, poly p.

Final Algorithm

$A[i]$ stores if $x_1 \cdots x_i$ is in L^*. M is poly-time Alg for L, poly p.

Input $x = x_1 \cdots x_n$
$A[1] = A[2] = ... = A[n] =$ FALSE
$A[0] =$ TRUE
for $i = 1$ to n do
    for $j = 0$ to $i - 1$ do
        if $A[j]$ AND $M(x_{j+1} \cdots x_i) = Y$ then $A[i] =$ TRUE
output $A[n]$

Final Algorithm

$A[i]$ stores if $x_1 \cdots x_i$ is in L^*. M is poly-time Alg for L, poly p.

Input $x = x_1 \cdots x_n$
$A[1] = A[2] = \ldots = A[n] = \text{FALSE}$
$A[0] = \text{TRUE}$
for $i = 1$ to n do
    for $j = 0$ to $i - 1$ do
        if $A[j]$ AND $M(x_{j+1} \cdots x_i) = Y$ then $A[i] = \text{TRUE}$
output $A[n]$

$O(n^2)$ calls to M on inputs of length $\leq n$. Runtime $\leq O(n^2 p(n))$.

Final Algorithm

$A[i]$ stores if $x_1 \cdots x_i$ is in L^*. M is poly-time Alg for L, poly p.

Input $x = x_1 \cdots x_n$
$A[1] = A[2] = ... = A[n] = $ FALSE
$A[0] = $ TRUE
for $i = 1$ to n do
  for $j = 0$ to $i - 1$ do
    if $A[j]$ AND $M(x_{j+1} \cdots x_i) = Y$ then $A[i] = $ TRUE
output $A[n]$

$O(n^2)$ calls to M on inputs of length $\le n$. Runtime $\le O(n^2 p(n))$.
Note Key is that the set of polynomials is closed under mult by n^2.

Closure of NP

Closure of NP under Union

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 \cup L_2 \in \mathrm{NP}$.

Closure of NP under Union

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 \cup L_2 \in \mathrm{NP}$.
$L_1 = \{x : (\exists y_1)[|y_1| = p_1(|x|) \wedge (x, y_1) \in B_1]$
$L_2 = \{x : (\exists y_2)[|y_2| = p_2(|x|) \wedge (x, y_2) \in B_2]$

Closure of NP under Union

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 \cup L_2 \in \mathrm{NP}$.
$L_1 = \{x : (\exists y_1)[|y_1| = p_1(|x|) \wedge (x, y_1) \in B_1]$
$L_2 = \{x : (\exists y_2)[|y_2| = p_2(|x|) \wedge (x, y_2) \in B_2]$

The following defines $L_1 \cup L_2$ in an NP-way.

$L_1 \cup L_2 = \{x : (\exists y)$:

- $|y| = p_1(|x|) + p_2(|x|) + 1$. $y = y_1\$y_2$ where $|y_1| = p_1(|x|)$ and $|y_2| = p_2(|X|)$.
- $(x, y_1) \in B_1 \vee (x, y_2) \in B_2)$

Closure of NP under Union

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 \cup L_2 \in \mathrm{NP}$.
$L_1 = \{x : (\exists y_1)[|y_1| = p_1(|x|) \wedge (x, y_1) \in B_1]$
$L_2 = \{x : (\exists y_2)[|y_2| = p_2(|x|) \wedge (x, y_2) \in B_2]$

The following defines $L_1 \cup L_2$ in an NP-way.

$L_1 \cup L_2 = \{x : (\exists y)$:

- $|y| = p_1(|x|) + p_2(|x|) + 1$. $y = y_1 \$ y_2$ where $|y_1| = p_1(|x|)$ and $|y_2| = p_2(|X|)$.
- $(x, y_1) \in B_1 \vee (x, y_2) \in B_2)$

Witness: $|y| = p_1(|x|) + p_2(|x|) + 1$ is short.

Closure of NP under Union

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 \cup L_2 \in \mathrm{NP}$.
$L_1 = \{x : (\exists y_1)[|y_1| = p_1(|x|) \wedge (x, y_1) \in B_1]$
$L_2 = \{x : (\exists y_2)[|y_2| = p_2(|x|) \wedge (x, y_2) \in B_2]$

The following defines $L_1 \cup L_2$ in an NP-way.

$L_1 \cup L_2 = \{x : (\exists y)$:

- $|y| = p_1(|x|) + p_2(|x|) + 1$. $y = y_1\$y_2$ where $|y_1| = p_1(|x|)$ and $|y_2| = p_2(|X|)$.
- $(x, y_1) \in B_1 \vee (x, y_2) \in B_2)$

Witness: $|y| = p_1(|x|) + p_2(|x|) + 1$ is short.
Verification: $(x, y_1) \in B_1 \vee (x, y_2) \in B_2)$, is quick.

Closure of NP under Intersection

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 \cap L_2 \in \mathrm{NP}$.

Proof is similar to closure under $\cup$.

Closure of NP under Concatenation

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 L_2 \in \mathrm{NP}$.

Closure of NP under Concatenation

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 L_2 \in \mathrm{NP}$.
$L_1 = \{x : (\exists y_1)[|y_1| = p_1(|x|) \wedge (x, y_1) \in B_1]$
$L_2 = \{x : (\exists y_2)[|y_2| = p_2(|x|) \wedge (x, y_2) \in B_2]$

Closure of NP under Concatenation

Thm If $L_1 \in \mathrm{NP}$ and $L_2 \in \mathrm{NP}$ then $L_1 L_2 \in \mathrm{NP}$.
$L_1 = \{x : (\exists y_1)[|y_1| = p_1(|x|) \wedge (x, y_1) \in B_1]$
$L_2 = \{x : (\exists y_2)[|y_2| = p_2(|x|) \wedge (x, y_2) \in B_2]$

The following defines $L_1 L_2$ in an NP-way.

$$\{x : (\exists x_1, x_2, y_1, y_2)$$

- $x = x_1 x_2$
- $|y_1| = p_1(|x_1|)$
- $|y_2| = p_2(|x_2|)$
- $(x_1, y_1) \in B_1$
- $(x_2, y_2) \in B_2$

Closure of NP under *

Thm If $L \in \mathrm{NP}$ then $L^* \in \mathrm{NP}$.

Similar to the Closure of NP under CONCAT.

Much easier than the proof that P is closed under *.

Is NP closed under Complementation?

Unknown to Science!

But the common opinion is NO.

Is NP closed under Complementation?

Unknown to Science!
But the common opinion is NO.
Unlikely that there is a short poly-verifiable witness to G NOT being 3-colorable.

IND SET is NP-Complete, 3COL is NP-Complete

GOTO the slides from Stanford

CLIQ ≤ SAT

SAT-solvers are so good that people now use reductions to map problem TO SAT.

We do a reduction of CLIQ to SAT.

Does G have a clique of size k?

CLIQ $\le$ SAT

SAT-solvers are so good that people now use reductions to map problem TO SAT.

We do a reduction of CLIQ to SAT.

Does G have a clique of size k?

We rephrase that:

CLIQ $\le$ SAT

SAT-solvers are so good that people now use reductions to map problem TO SAT.

We do a reduction of CLIQ to SAT.

Does G have a clique of size k?

We rephrase that:

Let $G = (V, E)$.

CLIQ $\le$ SAT

SAT-solvers are so good that people now use reductions to map problem TO SAT.

We do a reduction of CLIQ to SAT.

Does G have a clique of size k?

We rephrase that:

Let $G = (V, E)$.

G has a clique of size k is EQUIVALENT TO:
There is a 1-1 function $\{1, \ldots, k\} \rightarrow V$ *such that for all* $1 \le a, b \le k$, $(f(a), f(b)) \in E$.

CLIQ $\le$ SAT

Given G and k We want to know:
There is a 1-1 function $\{1, \ldots, k\} \rightarrow V$ *such that for all* $1 \le a, b \le k$, $(f(a), f(b)) \in E$.

CLIQ $\le$ SAT

Given G and k We want to know:
There is a 1-1 function $\{1, \ldots, k\} \to V$ *such that for all* $1 \le a, b \le k$, $(f(a), f(b)) \in E$.

We formulate this as a Boolean Formula:

1. For $1 \le i \le k$, $1 \le j \le n$, have Boolean Vars x_{ij}. Intent:

$$x_{ij} = \begin{cases} T & \text{if vertex } i \text{ maps to vertex } j \\ F & \text{if vertex } i \text{ does not maps to vertex } j \end{cases} \tag{1}$$

2. Part of formula says x_{ij} is a bijection.
3. Part of formula says that the k points map to a clique.

Decidability and Undecidability

Recall Turing Machines

Recall Turing Machines

1. TM's are Java Programs.

Recall Turing Machines

1. TM's are Java Programs.
2. We have a listing of them $M_1, M_2, \ldots$.

Recall Turing Machines

1. TM's are Java Programs.
2. We have a listing of them $M_1, M_2, \ldots$.
3. If you run $M_e(d)$ it might not halt.

Recall Turing Machines

1. TM's are Java Programs.
2. We have a listing of them $M_1, M_2, \ldots$.
3. If you run $M_e(d)$ it might not halt.
4. Everything computable is computable by some TM.

Recall Turing Machines

1. TM's are Java Programs.
2. We have a listing of them $M_1, M_2, \ldots$.
3. If you run $M_e(d)$ it might not halt.
4. Everything computable is computable by some TM.
5. A TM that halts on all inputs is called **total** .

Computable Sets

Def A set A is *computable* if there exists a Turing Machine M that behaves as follows:

$$M(x) = \begin{cases} Y & \text{if } x \in A \\ N & \text{if } x \notin A \end{cases} \quad (2)$$

Computable Sets

Def A set A is *computable* if there exists a Turing Machine M that behaves as follows:

$$M(x) = \begin{cases} Y & \text{if } x \in A \\ N & \text{if } x \notin A \end{cases} \tag{2}$$

Computable sets are also called decidable or solvable. A machine such as M above is said to **decide** A.

Computable Sets

Def A set A is *computable* if there exists a Turing Machine M that behaves as follows:

$$M(x) = \begin{cases} Y & \text{if } x \in A \\ N & \text{if } x \notin A \end{cases} \tag{2}$$

Computable sets are also called decidable or solvable. A machine such as M above is said to **decide** A.

Notation DEC is the set of Decidable Sets.

Notation

Notation

Notation $M_{e,s}(d)$ is the result of running $M_e(d)$ for s steps.

Notation

Notation $M_{e,s}(d)$ is the result of running $M_e(d)$ for s steps.
$M_e(d)\downarrow$ means $M_e(d)$ halts.

Notation

Notation $M_{e,s}(d)$ is the result of running $M_e(d)$ for s steps.
$M_e(d)\downarrow$ means $M_e(d)$ halts.
$M_e(d)\uparrow$ means $M_e(d)$ does not halts.

Notation

Notation $M_{e,s}(d)$ is the result of running $M_e(d)$ for s steps.
$M_e(d) \downarrow$ means $M_e(d)$ halts.
$M_e(d) \uparrow$ means $M_e(d)$ does not halts.
$M_{e,s}(d) \downarrow$ means $M_e(d)$ halts within s steps.

Notation

Notation $M_{e,s}(d)$ is the result of running $M_e(d)$ for s steps.
$M_e(d)\downarrow$ means $M_e(d)$ halts.
$M_e(d)\uparrow$ means $M_e(d)$ does not halts.
$M_{e,s}(d)\downarrow$ means $M_e(d)$ halts within s steps.
$M_{e,s}(d)\downarrow= z$ means $M_e(d)$ halts within s steps and outputs z.

Notation

Notation $M_{e,s}(d)$ is the result of running $M_e(d)$ for s steps.
$M_e(d) \downarrow$ means $M_e(d)$ halts.
$M_e(d) \uparrow$ means $M_e(d)$ does not halts.
$M_{e,s}(d) \downarrow$ means $M_e(d)$ halts within s steps.
$M_{e,s}(d) \downarrow = z$ means $M_e(d)$ halts within s steps and outputs z.
$M_{e,s}(d) \uparrow$ means $M_e(d)$ has not halted within s steps.

Notation

Notation $M_{e,s}(d)$ is the result of running $M_e(d)$ for s steps.
$M_e(d)\downarrow$ means $M_e(d)$ halts.
$M_e(d)\uparrow$ means $M_e(d)$ does not halts.
$M_{e,s}(d)\downarrow$ means $M_e(d)$ halts within s steps.
$M_{e,s}(d)\downarrow= z$ means $M_e(d)$ halts within s steps and outputs z.
$M_{e,s}(d)\uparrow$ means $M_e(d)$ has not halted within s steps.

Noncomputable Sets

Are there any noncomputable sets?

1. Yes—ALL SETS: uncountable. DEC Sets: countable, hence there exists an uncountable number of noncomputable sets.
2. YES—HALT is undecidable, and once you have that you have many other sets undec.
3. YES—the problem of telling if a $p \in \mathbb{Z}[x_1, \ldots, x_n]$ has an int solution is undecidable.

The HALTING Problem

Def The HALTING set is the set

$$HALT = \{(e, d) \mid M_e(d) \text{ halts } \}.$$

HALT is Undecidable

Thm HALT is not computable.
Proof Assume HALT computable via TM M.

$$M(e,d) = \begin{cases} Y & \text{if } M_e(d)\downarrow \\ N & \text{if } M_e(d)\uparrow \end{cases} \quad (3)$$

We use M to create the following machine which is M_e.

1. Input d
2. Run $M(d,d)$
3. If $M(d,d) = Y$ then RUN FOREVER.
4. If $M(d,d) = N$ then HALT.

$M_e(e)\downarrow \implies M(e,e) = Y \implies M_e(e)\uparrow$
$M_e(e)\uparrow \implies M(e,e) = N \implies M_e(e)\downarrow$
We now have that $M_e(e)$ cannot $\downarrow$ and cannot $\uparrow$. **Contradiction.**

Other Undecidable Problems

Using that HALT is undecidable we can prove the following undecidable:

Other Undecidable Problems

Using that HALT is undecidable we can prove the following undecidable:

$\{e : M_e$ halts on **at least** 12 numbers $\}$ (**at most** ,**exactly**)

Other Undecidable Problems

Using that HALT is undecidable we can prove the following undecidable:

$\{e : M_e$ halts on **at least** 12 numbers $\}$ (**at most** ,**exactly**)

$\{e : M_e$ halts on an **infinite** number of numbers$\}$

Other Undecidable Problems

Using that HALT is undecidable we can prove the following undecidable:

$\{e : M_e$ halts on **at least** 12 numbers $\}$ (**at most** ,**exactly**)

$\{e : M_e$ halts on an **infinite** number of numbers$\}$

$\{e : M_e$ halts on a **finite** number of numbers$\}$

Other Undecidable Problems

Using that HALT is undecidable we can prove the following undecidable:

$\{e : M_e$ halts on **at least** 12 numbers $\}$ (**at most** ,**exactly**)

$\{e : M_e$ halts on an **infinite** number of numbers$\}$

$\{e : M_e$ halts on a **finite** number of numbers$\}$

$\{e : M_e$ does the Hokey Pokey and turns itself around $\}$

Other Undecidable Problems

Using that HALT is undecidable we can prove the following undecidable:

$\{e : M_e$ halts on **at least** 12 numbers $\}$ (**at most** ,**exactly**)

$\{e : M_e$ halts on an **infinite** number of numbers$\}$

$\{e : M_e$ halts on a **finite** number of numbers$\}$

$\{e : M_e$ does the Hokey Pokey and turns itself around $\}$

$TOT = \{e : M_e$ halts on all inputs$\}$

Other Undecidable Problems

Using that HALT is undecidable we can prove the following undecidable:

$\{e : M_e$ halts on **at least** 12 numbers $\}$ (**at most** ,**exactly**)

$\{e : M_e$ halts on an **infinite** number of numbers$\}$

$\{e : M_e$ halts on a **finite** number of numbers$\}$

$\{e : M_e$ does the Hokey Pokey and turns itself around $\}$

$TOT = \{e : M_e$ halts on all inputs$\}$

Proofs by reductions. Similar to NPC. We **will not** do that.

Σ_1 Sets

HALT is undecidable.

Σ_1 Sets

HALT is undecidable. How undecidable?

Σ_1 Sets

HALT is undecidable. How undecidable? Measure with quants:

Σ_1 Sets

HALT is undecidable. How undecidable? Measure with quants:

$$HALT = \{(e,d) : (\exists s)[M_{e,s}(d)\downarrow]\}$$

Σ_1 Sets

HALT is undecidable. How undecidable? Measure with quants:

$$HALT = \{(e,d) : (\exists s)[M_{e,s}(d)\downarrow]\}$$

Let

$$B = \{(e,d,s) : M_{e,s}(d)\downarrow\}$$

Σ_1 Sets

HALT is undecidable. How undecidable? Measure with quants:

$$HALT = \{(e,d) : (\exists s)[M_{e,s}(d)\downarrow]\}$$

Let

$$B = \{(e,d,s) : M_{e,s}(d)\downarrow\}$$

B is decidable and

$$HALT = \{(e,d) : (\exists s)[(e,d,s) \in B]\}$$

Σ_1 Sets

HALT is undecidable. How undecidable? Measure with quants:

$$HALT = \{(e,d) : (\exists s)[M_{e,s}(d)\downarrow]\}$$

Let

$$B = \{(e,d,s) : M_{e,s}(d)\downarrow\}$$

B is decidable and

$$HALT = \{(e,d) : (\exists s)[(e,d,s) \in B]\}$$

B is decidable. This inspires the following definition.

Σ_1 Sets

HALT is undecidable. How undecidable? Measure with quants:

$$HALT = \{(e,d) : (\exists s)[M_{e,s}(d)\downarrow]\}$$

Let

$$B = \{(e,d,s) : M_{e,s}(d)\downarrow\}$$

B is decidable and

$$HALT = \{(e,d) : (\exists s)[(e,d,s) \in B]\}$$

B is decidable. This inspires the following definition.

Def $A \in \Sigma_1$ if there exists decidable B such that

$$A = \{x : (\exists y)[(x,y) \in B]\}$$

Σ_1 Sets

HALT is undecidable. How undecidable? Measure with quants:

$$HALT = \{(e,d) : (\exists s)[M_{e,s}(d)\downarrow]\}$$

Let

$$B = \{(e,d,s) : M_{e,s}(d)\downarrow\}$$

B is decidable and

$$HALT = \{(e,d) : (\exists s)[(e,d,s) \in B]\}$$

B is decidable. This inspires the following definition.

Def $A \in \Sigma_1$ if there exists decidable B such that

$$A = \{x : (\exists y)[(x,y) \in B]\}$$

Does this definition remind you of something?

Σ_1 Sets

HALT is undecidable. How undecidable? Measure with quants:

$$HALT = \{(e,d) : (\exists s)[M_{e,s}(d)\downarrow]\}$$

Let

$$B = \{(e,d,s) : M_{e,s}(d)\downarrow\}$$

B is decidable and

$$HALT = \{(e,d) : (\exists s)[(e,d,s) \in B]\}$$

B is decidable. This inspires the following definition.

Def $A \in \Sigma_1$ if there exists decidable B such that

$$A = \{x : (\exists y)[(x,y) \in B]\}$$

Does this definition remind you of something? YES- NP.

Compare NP to Σ_1

$A \in \mathrm{NP}$ if there exists $B \in \mathrm{P}$ and poly p such that

$$A = \{x : (\exists y, |y| \leq p(|x|))[(x, y) \in B]\}$$

Compare NP to Σ_1

$A \in \mathrm{NP}$ if there exists $B \in \mathrm{P}$ and poly p such that

$$A = \{x : (\exists y, |y| \leq p(|x|))[(x, y) \in B]\}$$

$A \in \Sigma_1$ if there exists $B \in \mathrm{DEC}$ such that

$$A = \{x : (\exists y)[(x, y) \in B]\}$$

Compare NP to Σ_1

Compare NP to Σ_1

1. Both use a quantifier and then something easy. So the sets are difficult because of the quantifier.

Compare NP to Σ_1

1. Both use a quantifier and then something easy. So the sets are difficult because of the quantifier.
2. 2.1 For NP **easy** means P and the quantifier is over an exp size set.

Compare NP to Σ_1

1. Both use a quantifier and then something easy. So the sets are difficult because of the quantifier.
2. 2.1 For NP **easy** means P and the quantifier is over an exp size set.
   2.2 For Σ_1 **easy** means DEC and the quantifier is over $\mathbb{N}$.

Compare NP to Σ_1

1. Both use a quantifier and then something easy. So the sets are difficult because of the quantifier.
2.
   2.1 For NP **easy** means P and the quantifier is over an exp size set.
   2.2 For Σ_1 **easy** means DEC and the quantifier is over $\mathbb{N}$.
3. Σ_1 came first by several decades. Complexity theory borrowed ideas from Computability theory for the basic definitions.

Compare NP to Σ_1

1. Both use a quantifier and then something easy. So the sets are difficult because of the quantifier.
2. 2.1 For NP **easy** means P and the quantifier is over an exp size set.
   2.2 For Σ_1 **easy** means DEC and the quantifier is over $\mathbb{N}$.
3. Σ_1 came first by several decades. Complexity theory borrowed ideas from Computability theory for the basic definitions.
4. Are ideas from Computability theory useful in complexity theory?

Compare NP to Σ_1

1. Both use a quantifier and then something easy. So the sets are difficult because of the quantifier.
2. 2.1 For NP **easy** means P and the quantifier is over an exp size set.
   2.2 For Σ_1 **easy** means DEC and the quantifier is over $\mathbb{N}$.
3. Σ_1 came first by several decades. Complexity theory borrowed ideas from Computability theory for the basic definitions.
4. Are ideas from Computability theory useful in complexity theory? Yes, to a limited extent.

Compare NP to Σ_1

1. Both use a quantifier and then something easy. So the sets are difficult because of the quantifier.
2. 2.1 For NP **easy** means P and the quantifier is over an exp size set.
   2.2 For Σ_1 **easy** means DEC and the quantifier is over $\mathbb{N}$.
3. Σ_1 came first by several decades. Complexity theory borrowed ideas from Computability theory for the basic definitions.
4. Are ideas from Computability theory useful in complexity theory? Yes, to a limited extent. My thesis was on showing some of those limits.

Beyond Σ_1

Def B is always a decidable set.

Beyond Σ_1

Def B is always a decidable set.

$A \in \Pi_1$ if $A = \{x : (\forall y)[(x, y) \in B]\}$.

Beyond Σ_1

Def B is always a decidable set.

$A \in \Pi_1$ if $A = \{x : (\forall y)[(x, y) \in B]\}$.

$A \in \Sigma_2$ if $A = \{x : (\exists y_1)(\forall y_2)[(x, y_1, y_2) \in B]\}$.

Beyond Σ_1

Def B is always a decidable set.

$A \in \Pi_1$ if $A = \{x : (\forall y)[(x, y) \in B]\}$.

$A \in \Sigma_2$ if $A = \{x : (\exists y_1)(\forall y_2)[(x, y_1, y_2) \in B]\}$.

$A \in \Pi_2$ if $A = \{x : (\forall y_1)(\exists y_2)[(x, y_1, y_2) \in B]\}$.

$\vdots$

Beyond Σ_1

Def B is always a decidable set.

$A \in \Pi_1$ if $A = \{x : (\forall y)[(x, y) \in B]\}$.

$A \in \Sigma_2$ if $A = \{x : (\exists y_1)(\forall y_2)[(x, y_1, y_2) \in B]\}$.

$A \in \Pi_2$ if $A = \{x : (\forall y_1)(\exists y_2)[(x, y_1, y_2) \in B]\}$.

$\vdots$

$TOT = \{x : (\forall y)(\exists s)[M_{x,s}(y) \downarrow]\} \in \Pi_2$.

Beyond Σ_1

Def B is always a decidable set.

$A \in \Pi_1$ if $A = \{x : (\forall y)[(x, y) \in B]\}$.

$A \in \Sigma_2$ if $A = \{x : (\exists y_1)(\forall y_2)[(x, y_1, y_2) \in B]\}$.

$A \in \Pi_2$ if $A = \{x : (\forall y_1)(\exists y_2)[(x, y_1, y_2) \in B]\}$.

$\vdots$

$TOT = \{x : (\forall y)(\exists s)[M_{x,s}(y) \downarrow]\} \in \Pi_2$.

Known: $TOT \notin \Sigma_1 \cup \Pi_1$.

Beyond Σ_1

Def B is always a decidable set.

$A \in \Pi_1$ if $A = \{x : (\forall y)[(x, y) \in B]\}$.

$A \in \Sigma_2$ if $A = \{x : (\exists y_1)(\forall y_2)[(x, y_1, y_2) \in B]\}$.

$A \in \Pi_2$ if $A = \{x : (\forall y_1)(\exists y_2)[(x, y_1, y_2) \in B]\}$.

$\vdots$

$TOT = \{x : (\forall y)(\exists s)[M_{x,s}(y) \downarrow]\} \in \Pi_2$.

Known: $TOT \notin \Sigma_1 \cup \Pi_1$.

Known:

$\Sigma_1 \subset \Sigma_2 \subset \Sigma_3 \cdots$

$\Pi_1 \subset \Pi_2 \subset \Pi_3 \cdots$

Beyond Σ_1

Def B is always a decidable set.

$A \in \Pi_1$ if $A = \{x : (\forall y)[(x, y) \in B]\}$.

$A \in \Sigma_2$ if $A = \{x : (\exists y_1)(\forall y_2)[(x, y_1, y_2) \in B]\}$.

$A \in \Pi_2$ if $A = \{x : (\forall y_1)(\exists y_2)[(x, y_1, y_2) \in B]\}$.

$\vdots$

$TOT = \{x : (\forall y)(\exists s)[M_{x,s}(y) \downarrow]\} \in \Pi_2$.

Known: $TOT \notin \Sigma_1 \cup \Pi_1$.

Known:

$\Sigma_1 \subset \Sigma_2 \subset \Sigma_3 \cdots$

$\Pi_1 \subset \Pi_2 \subset \Pi_3 \cdots$

TOT is **harder** than HALT.

WS1S Formulas and Sentences

1. Variables x, y, z range over $\mathbb{N}$, X, Y, Z range over finite subsets of $\mathbb{N}$.
2. Symbols: $<$, $\in$ (usual meaning), S (meaning $S(x) = x + 1$).
3. A *Formula* allows variables to not be quantified over. A Formula is neither true or false. Example: $(\exists x)[x + y = 7]$.
4. A *Sentence* has all variables quantified over. Example: $(\forall y)(\exists x)[x + y = 7]$. So a Sentence is either true or false IF domain is

WS1S: Weak Second order Theory of One Successor. Weak Second order means quantify over finite sets.

Atomic Formulas

An *Atomic Formula* is:

1. For any $c \in \mathbb{N}$, $x = y + c$ is an Atomic Formula.
2. For any $c \in \mathbb{N}$, $x < y + c$ is an Atomic Formula.
3. For any $c, d \in \mathbb{N}$, $x \equiv y + c \pmod{d}$ is an Atomic Formula.
4. For any $c \in \mathbb{N}$, $x + c \in X$ is an Atomic Formula.
5. For any $c \in \mathbb{N}$, $X = Y + c$ is an Atomic Formula.

WS1S Formulas

Build up formulas from atomic formulas using $\wedge$, $\vee$, $\neg$, $\exists$, $\forall$. Hence can define the set of formulas.

Can put formulas into **Prenex Normal Form** :

$$(Q_1 v_1)(Q_2 v_2) \cdots (Q_n v_m)[\phi(v_1, \ldots, v_n)]$$

Def If $\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ is a WS1S Formula then $TRUE(\phi)$ is the set

$$\{(a_1, \ldots, a_n, A_1, \ldots, A_m) \mid \phi(a_1, \ldots, a_n, A_1, \ldots, A_m) = T\}$$

This is the set of $(x_1, \ldots, x_n, X_1, \ldots, X_m)$ that make ϕ TRUE.

KEY THEOREM

Thm For all WS1S formulas ϕ the set $TRUE_{\phi}$ is regular.

KEY THEOREM

Thm For all WS1S formulas ϕ the set $TRUE_\phi$ is regular.

Need to clarify representation and the define stupid states to make all of this work.

KEY THEOREM

Thm For all WS1S formulas ϕ the set $TRUE_{\phi}$ is regular.

Need to clarify representation and the define stupid states to make all of this work.

We prove this by induction on the formation of a formula. If you prefer- induction on the LENGTH of a formula.

DECIDABILITY OF WS1S

Thm: WS1S is Decidable.
Proof:

1. Given a SENTENCE in WS1S put it into the form

   $$(Q_1 X_1) \cdots (Q_n X_n)(Q_{n+1} x_1) \cdots (Q_{n+m} x_m)[\phi(x_1, \ldots, x_m, X_1, \ldots, X_n)]$$

2. Assume $Q_1 = \exists$. (If not then negate and negate answer.)
3. View as $(\exists X)[\phi(X)]$, a FORMULA with ONE free var.
4. Construct DFA M for $\{X \mid \phi(X) \text{ is true}\}$.
5. Test if $L(M) = \emptyset$.
6. If $L(M) \neq \emptyset$ then $(\exists X)[\phi(X)]$ is TRUE.
   If $L(M) = \emptyset$ then $(\exists X)[\phi(X)]$ is FALSE.

Interesting Points about WS1S

Interesting Points about WS1S

1. Our algorithm has worst case $2^{2^{\cdots n}}$ time.

Interesting Points about WS1S

1. Our algorithm has worst case $2^{2^{\cdots n}}$ time.
2. There is a better algorithm: $2^{2^{n^3 \log n}}$ time. Provable that (roughly) can't do better than that.

Interesting Points about WS1S

1. Our algorithm has worst case $2^{2^{\cdots n}}$ time.
2. There is a better algorithm: $2^{2^{n^3 \log n}}$ time. Provable that (roughly) can't do better than that.
3. S1S is proven decidable using ω-regular. WS2S is proven decidable using Tree-Aut. S2S is proven decidable using ω-Tree-Automata. All the proofs use same structure as ours: the Truth sets are BLAH-regular and can test non-emptyness of BLAH-aut.

Interesting Points about WS1S

1. Our algorithm has worst case $2^{2^{\cdots n}}$ time.
2. There is a better algorithm: $2^{2^{n^3 \log n}}$ time. Provable that (roughly) can't do better than that.
3. S1S is proven decidable using ω-regular. WS2S is proven decidable using Tree-Aut. S2S is proven decidable using ω-Tree-Automata. All the proofs use same structure as ours: the Truth sets are BLAH-regular and can test non-emptyness of BLAH-aut.
4. For WS1S, WS2S, S1S, can state things about low-level verification of code (see MONA group). For S2S can state actual Mathematical Theorems of interest. However, the program would take to long to use this, and it would not offer mathematical insights anyway.

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 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 $\boldsymbol{C(x)}$ 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 $\boldsymbol{C(x|y)}$ 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 $\boldsymbol{C(x)}$ 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 $\boldsymbol{C(x|y)}$ 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 $C(x|y) \ge n$.

Note Java-Random, Python-Random, 1-tape-TM-Random will all give different values. But all within $O(1)$.

Do Random Strings Exist?

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

YES- there are more Strings of length n then TMs of length $\leq n-1$.

Application of Kolmogorov Complexity to Proving Languages Not Regular

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ 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 $\delta(r, a^n x) \in F$.

$L_1 = \{a^n b^n : n \in \mathbb{N}\}$ 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 $\delta(r, a^n x) \in F$.

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 $\delta(r, a^n x) \in F$.

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 $\delta(r, a^n x) \in F$.

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 $\delta(r, a^n x) \in F$.

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.

Kolm Complexity Also Applies To

Kolm Complexity Also Applies To

1. Can use it to show other langs not regular (I did that in class).

Kolm Complexity Also Applies To

1. Can use it to show other langs not regular (I did that in class).
2. Can use it to show some langs require a large DFA, NFA, CFG, TM.

Kolm Complexity Also Applies To

1. Can use it to show other langs not regular (I did that in class).
2. Can use it to show some langs require a large DFA, NFA, CFG, TM.
3. 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 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!!!