Review For the Final

May 12, 2020
Turing Machines and DTIME

May 12, 2020
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

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

$x \notin A \rightarrow M(x) = N$
Def Let $T(n)$ be a computable function (think increasing). $A$ is in $\text{DTIME}(T(n))$ 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 too much on machine model.

- Prove theorems about $\text{DTIME}(T(n))$ where the model does not matter. (Time hierarchy theorem).
- Define time classes that are model-independent (P, NP stuff)
The Time Hierarchy Thm

**Thm** (The Time Hierarchy Thm) For all computable increasing $T(n)$ there exists a decidable set $A$ such that $A \notin \text{DTIME}(T(n))$.

**Proof** Let $M_1, M_2, \ldots$, represent all of $\text{DTIME}(T(n))$ (obtain by listing out all Turing Machines and putting a time bound on them). Here is our algorithm for $A$. It will be a subset of $0^*$.

1. Input $0^i$.
2. Run $M_i(0^i)$. If the results is 1 then output 0. If the results is 0 then output 1.

For all $i$, $M_i$ and $A$ DIFFER on $0^i$. Hence $A$ is not decided by any $M_i$. So $A \notin \text{DTIME}(T(n))$.

**End of Proof**
P, NP, Reductions

May 12, 2020
P and EXP

Def

1. P = DTIME($n^{O(1)}$).
2. EXP = DTIME($2^{n^{O(1)}}$).
3. PF is the set of functions that are computable in poly time.
**Def** A is in NP if there exists a set $B \in \mathbb{P}$ and a polynomial $p$ such that

$$A = \{ x \mid (\exists y)[|y| = p(|x|) \land (x, y) \in B] \}.$$
**Def** A is in \( \text{NP} \) if there exists a set \( B \in \text{P} \) and a polynomial \( p \) such that

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

Intuition. Let \( A \in \text{NP} \).

- 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.
**Def** A is in NP if there exists a set $B \in P$ and a polynomial $p$ such that

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

**Intuition.** Let $A \in \text{NP}$.

- 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 proof that $x \in A$. 
Examples of Sets in NP

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

\[ 3COL = \{ G : G \text{ is 3-colorable} \} \]

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

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

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

**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 are transitive. 

**Easy Lemma** If $X \leq Y$ and $Y \in P$ then $X \in P$. 

**Contrapositive** If $X \leq Y$ and $X \not\in P$ then $Y \not\in P$. 

**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 are transitive.

**Easy Lemma** If $X \leq Y$ and $Y \in \mathbb{P}$ then $X \in \mathbb{P}$. 
**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 $Y \in P$ then $X \in P$.

**Contrapositive** If $X \leq Y$ and $X \notin P$ then $Y \notin P$. 
Def of NP-Complete

Def A language $Y$ is NP-complete

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

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

Honesty
When I first saw the definition of NP-completeness I thought (1) there are no NP-complete sets or (2) there are no natural NP-complete sets. The condition: for EVERY $X \in \text{NP}$, $X \leq Y$? seemed very hard to meet.
Def A language $Y$ is **NP-complete**

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

**Easy Lemma** If $Y$ is NP-complete and $Y \in \text{P}$ then $\text{P} = \text{NP}$. 
Def of NP-Complete

Def A language $Y$ is **NP-complete**
- $Y \in \text{NP}$
- If $X \in \text{NP}$ then $X \leq Y$

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

Honesty When I first saw the definition of NP-completeness I thought (1) there are no NP-complete sets or (2) there are no natural NP-complete sets.
**Def of NP-Complete**

A language $Y$ is **NP-complete**
- $Y \in \text{NP}$
- If $X \in \text{NP}$ then $X \leq Y$.

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

**Honesty** When I first saw the definition of NP-completeness I thought (1) there are no NP-complete sets or (2) there are no natural NP-complete sets.

The condition:

for EVERY $X \in \text{NP}$, $X \leq Y$?

seemed very hard to meet.
SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed: **CNF-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. The proof is not hard, but it involves looking at actual TMs. We will prove it next lecture. SAT was 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. The proof is not hard, but it involves looking at actual TMs. We will prove it next lecture. SAT was the **first** NP-complete problem. You could not use some other problem.

2. Once we have SAT is NP-complete we will NEVER use TMs again. To show $Y$ NP-complete: (1) $Y \in NP$, (2) $SAT \leq Y$. 
SAT is NP-Complete

In 1971 Stephen Cook and Leonid Levin Independently showed: **CNF-SAT is NP-complete**

Thoughts on this:

1. The proof is not hard, but it involves looking at actual TMs. We will prove it next lecture. SAT was the **first** NP-complete problem. You could not use some other problem.

2. Once we have SAT is NP-complete we will NEVER use TMs again. To show $Y$ NP-complete: (1) $Y \in NP$, (2) $SAT \leq Y$.

3. Thousands of problems are NP-complete. If any are in P then they are all in P.
The Cook-Levin Thm

May 12, 2020
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 $j$th symbol in the $i$th configuration is $\sigma$.
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

May 12, 2020
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. $L_1L_2 \in P$. EASY. Uses $p(n)$ poly then $np(n)$ poly.
Closure of P Under *

**Thm** If \( L \in P \) then \( L^* \in P \).

**Proof**
First let's 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.
Dynamic Programming We solve a harder problem but get lots of information in the process.
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 $x \in L^*$
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 $x = x_1 \cdots x_n$


$A[0] = \text{TRUE}$

for $i = 1$ to $n$ do

    for $j = 0$ to $i - 1$ do

        if $A[j] \text{ AND } M(x_{j+1} \cdots x_i) = \text{Y}$ then $A[i] = \text{TRUE}$

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


$A[0] = \text{TRUE}$

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 $\leq O(n^2 p(n))$.

Note Key is that the set of polynomials is closed under mult by $n^2$. 
Closure of NP

May 12, 2020
Closure of NP under Union

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

$L_1 = \{ x : (\exists y_1)[|y_1| = p_1(|x|) \land (x, y_1) \in B_1] \}$

$L_2 = \{ x : (\exists y_2)[|y_2| = p_2(|x|) \land (x, y_2) \in B_2] \}$
**Thm** If $L_1 \in \text{NP}$ and $L_2 \in \text{NP}$ then $L_1 \cup L_2 \in \text{NP}$.

$L_1 = \{ x : (\exists y_1)[|y_1| = p_1(|x|) \land (x, y_1) \in B_1] \}$

$L_2 = \{ x : (\exists y_2)[|y_2| = p_2(|x|) \land (x, y_2) \in B_2] \}$

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 $|y_2| = p_2(|X|)$.
- $(x, y_1) \in B_1 \lor (x, y_2) \in B_2$)
Closure of NP under Union

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

The following defines $L_1 \cup L_2$ in an NP-way.
$L_1 \cup L_2 = \{x : (\exists y):$

$\quad |y| = p_1(|x|) + p_2(|x|) + 1. \ y = y_1 \$ y_2$ where $|y_1| = p_1(|x|)$ and $|y_2| = p_2(|X|)$.

$\quad (x, y_1) \in B_1 \lor (x, y_2) \in B_2)$

Witness: $|y| = p_1(|x|) + p_2(|x|) + 1$ is short.
Closure of NP under Union

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

$L_1 = \{x : (\exists y_1)[|y_1| = p_1(|x|) \land (x, y_1) \in B_1]\}$

$L_2 = \{x : (\exists y_2)[|y_2| = p_2(|x|) \land (x, y_2) \in B_2]\}$

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 $|y_2| = p_2(|X|)).$

| $(x, y_1) \in B_1 \lor (x, y_2) \in B_2)$

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

Verification: $(x, y_1) \in B_1 \lor (x, y_2) \in B_2)$, is quick.
Closure of Concatenation

**Thm** If $L_1 \in \text{NP}$ and $L_2 \in \text{NP}$ then $L_1 L_2 \in \text{NP}$.
Closure of Concatenation

**Thm** If $L_1 \in \text{NP}$ and $L_2 \in \text{NP}$ then $L_1L_2 \in \text{NP}$.

$L_1 = \{ x : (\exists y_1)[|y_1| = p_1(|x|) \land (x, y_1) \in B_1] \}$

$L_2 = \{ x : (\exists y_2)[|y_2| = p_2(|x|) \land (x, y_2) \in B_2] \}$
**Closure of Concatenation**

**Thm** If $L_1 \in \text{NP}$ and $L_2 \in \text{NP}$ then $L_1L_2 \in \text{NP}$.

$L_1 = \{x : (\exists y_1)[|y_1| = p_1(|x|) \land (x, y_1) \in B_1]\}$

$L_2 = \{x : (\exists y_2)[|y_2| = p_2(|x|) \land (x, y_2) \in B_2]\}$

The following defines $L_1L_2$ in an NP-way.

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

- $x = x_1x_2$
- $|y_1| = p_1(|x_1|)$
- $|y_2| = p_2(|x_2|)$
- $(x_1, y_1) \in B_1$
- $(x_2, y_2) \in B_2$
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.
CLIQ \leq SAT

Does $G$ have a clique of size $k$?
Does $G$ have a clique of size $k$?

We rephrase that:
Does $G$ have a clique of size $k$?
We rephrase that:
Let $G = (V, E)$. 
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 $\{1, \ldots, k\} \to V$ such that for all $1 \leq a, b \leq k$, $(f(a), f(b)) \in E$. 

Given $G$ and $k$ We want to know:

There is a 1-1 function $\{1, \ldots, k\} \rightarrow V$ such that for all $1 \leq a, b \leq k$, $(f(a), f(b)) \in E$. 
**CLIQ \leq SAT**

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

We formulate this as a Boolean Formula:

1. For $1 \leq i \leq k$, $1 \leq j \leq n$, have Boolean Vars $x_{ij}$. Intent:
   
   $$x_{ij} = \begin{cases} 
   T & \text{if vertex } i \text{ maps to vertex } j \\
   F & \text{if vertex } i \text{ does not maps to vertex } j 
   \end{cases}$$  \hspace{1cm} (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

May 12, 2020
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.
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 $M_1, M_2, \ldots$. 

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.
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:

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

Computable sets are also called decidable or solvable. A machine such as $M$ above is said to decide $A$. Notation $\text{DEC}$ is the set of Decidable Sets.
Computable Sets

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

$$M(x) = \begin{cases} 
Y & \text{if } x \in A \\
N & \text{if } x \notin A 
\end{cases}$$  \hspace{1cm} (2)$$

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

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

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

(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

$Me(s(d))$ is the result of running $Me(d)$ for $s$ steps.

$Me(d)\downarrow$ means $Me(d)$ halts.

$Me(d)\uparrow$ means $Me(d)$ does not halt.

$Me(s(d))\downarrow$ means $Me(d)$ halts within $s$ steps.

$Me(s(d))\downarrow = z$ means $Me(d)$ halts within $s$ steps and outputs $z$.

$Me(s(d))\uparrow$ means $Me(d)$ has not halted within $s$ steps.
**Notation** \( M_{e,s}(d) \) is the result of running \( M_e(d) \) for \( s \) steps.
**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 halt.
**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 halt.
- $M_{e,s}(d) \downarrow$ 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 halt.

$M_{e,s}(d) \downarrow$ 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 $M_{e,s}(d)$ 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 halt.

$M_{e,s}(d) \downarrow$ 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 $M_{e,s}(d)$ 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 halt.
- $M_{e,s}(d) \downarrow$ 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:
Using that HALT is undecidable we can prove the following undecidable:

\{ e : M_e \text{ halts on } \text{at least} \ 12 \ \text{numbers} \} \ (\text{at } \text{most, exactly})
Other Undecidable Problems

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

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

\{ e : M_e \text{ halts on an infinite number of numbers} \}
Other Undecidable Problems

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

\{ e : M_e \text{ halts on at least 12 numbers} \} \ (\text{at most, exactly})
\{ e : M_e \text{ halts on an infinite number of numbers}\}
\{ e : M_e \text{ halts on a finite number of numbers}\}
Other Undecidable Problems

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

\{ e : M_e \text{ halts on at least 12 numbers} \} (at most, exactly)
\{ e : M_e \text{ halts on an infinite number of numbers} \}
\{ e : M_e \text{ halts on a finite number of numbers} \}
\{ e : M_e \text{ 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 \text{ halts on at least 12 numbers} \} \ (at \ most, exactly)
\{ e : M_e \text{ halts on an infinite number of numbers}\}
\{ e : M_e \text{ halts on a finite number of numbers}\}
\{ e : M_e \text{ does the Hokey Pokey and turns itself around } \}

TOT = \{ e : M_e \text{ halts on all inputs}\}
Other Undecidable Problems

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

\{ e : M_e \text{ halts on at least 12 numbers} \} \ (\text{at most, exactly})
\{ e : M_e \text{ halts on an infinite number of numbers} \}
\{ e : M_e \text{ halts on a finite number of numbers} \}
\{ e : M_e \text{ does the Hokey Pokey and turns itself around} \}

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

Proofs by reductions. Similar to NPC. We \text{will not} do that.
Σ₁ Sets

HALT is undecidable.
HALT is undecidable. How undecidable?
Σ₁ Sets

HALT is undecidable. How undecidable? Measure with quants:
Σ₁ Sets

HALT is undecidable. How undecidable? Measure with quants:

\[ \text{HALT} = \{(e, d) : (\exists s)[M_{e,s}(d) \downarrow]\} \]
HALT is undecidable. How undecidable? Measure with quants:

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

Let

\[ B = \{(e, d, s) : M_{e,s}(d) \downarrow\} \]
HALT is undecidable. How undecidable? Measure with quants:

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

Let

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

\( B \) is decidable and

\[ \text{HALT} = \{ (e, d) : (\exists s)[(e, d, s) \in B] \} \]
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.
Σ₁ 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 ∈ Σ₁ if there exists decidable \( B \) such that

\[ A = \{x : (\exists y)[(x, y) \in B]\} \]
Σ₁ 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?
Σ₁ 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 \( A \in \Sigma₁ \) 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 \( \Sigma_1 \)

\( A \in \text{NP} \) if there exists \( B \in \text{P} \) and poly \( p \) such that

\[
A = \{ x : (\exists y, |y| \leq p(|x|))(x, y) \in B \}
\]
Compare NP to $\Sigma_1$

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

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

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

$$A = \{x : (\exists y)[(x, y) \in B]\}$$
Compare NP to $\Sigma_1$

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

2. For NP easy means P and the quantifier is over an exp size set.

2.1 For $\Sigma_1$ easy means DEC and the quantifier is over $\mathbb{N}$.

3. $\Sigma_1$ 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.
Compare NP to $\Sigma_1$

1. Both use a quantifier and then something easy. So the sets are difficult because of the quantifier.
Compare NP to $\Sigma_1$

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.
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 $\Sigma_1$ easy means DEC and the quantifier is over $\mathbb{N}$. 
Compare NP to $\Sigma_1$

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 $\Sigma_1$ easy means DEC and the quantifier is over $\mathbb{N}$.
3. $\Sigma_1$ came first by several decades. Complexity theory borrowed ideas from Computability theory for the basic definitions.
Compare NP to $\Sigma_1$

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 $\Sigma_1$ easy means $\text{DEC}$ and the quantifier is over $\mathbb{N}$.

3. $\Sigma_1$ 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 $\Sigma_1$

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 $\Sigma_1$ easy means DEC and the quantifier is over $\mathbb{N}$.

3. $\Sigma_1$ 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 $\Sigma_1$

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 $\Sigma_1$ easy means DEC and the quantifier is over $\mathbb{N}$.

3. $\Sigma_1$ 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 $\Sigma_1$

**Def** $B$ is always a decidable set.
Beyond $\Sigma_1$

**Def** $B$ is always a decidable set.

$A \in \Pi_1$ if $A = \{x : (\forall y)(x, y) \in B\}$.
Beyond $\Sigma_1$

**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 $\Sigma_1$

**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 $\Sigma_1$

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

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

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

Known: $\Sigma_1 \subset \Sigma_2 \subset \Sigma_3 \cdots$.  

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

$TOT$ is harder than $HALT$.  

Beyond $\Sigma_1$

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

$\vdash$

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

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

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

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

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 $\Sigma_1$

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

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

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 $HALT$. 
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.
A *WS1S Formula* is:

1. Any Atomic Formula is a WS1S Formula.
2. If $\phi_1$, $\phi_2$ are WS1S Formulas then so are
   2.1 $\phi_1 \land \phi_2$,
   2.2 $\phi_1 \lor \phi_2$
   2.3 $\neg \phi_1$
3. If $\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ is a WS1S Formula then so are
   3.1 $(\exists x_i)[\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)]$
   3.2 $(\exists X_i)[\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)]$
**PRENEX NORMAL FORM**

A formulas is in **Prenex Normal Form** if it is of the form

\[(Q_1v_1)(Q_2v_2) \cdots (Q_nv_m)[\phi(v_1, \ldots, v_n)]\]

where the \(v_i\)'s are either number of finite-set variables, and \(\phi\) has no quantifiers. (There are \(m\) quantifiers and \(n \geq m\) variables since this is a formula- there could be variables that are not quantified over.)

Every formula can be put into this form using the following rules

1. \((\exists x)[\phi_1(x)] \lor (\exists y)[\phi_2(y)]\) is equiv to \((\exists x)[\phi_1(x) \lor \phi_2(x)]\).
2. \((\forall x)[\phi_1(x)] \land (\forall y)[\phi_2(y)]\) is equiv to \((\forall x)[\phi_1(x) \land \phi_2(x)]\).
3. \(\phi(x)\) is equivalent to \((\forall y)[\phi(x)]\) and \((\exists y)[\phi(x)]\).
**Def:** If $\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ is a WS1S Formula then $\text{TRUE}_\phi$ is the set

$$\{(x_1, \ldots, x_n, X_1, \ldots, X_m) \mid \phi(x_1, \ldots, x_n, X_1, \ldots, X_m) = T\}$$

This is the set of $(x_1, \ldots, x_n, X_1, \ldots, X_m)$ that make $\phi$ TRUE.
We want to say that *TRUE* is regular. Need to represent 
\((x_1, \ldots, x_n, X_1, \ldots, X_m)\).

We just look at \((x, y, X)\). Use the alphabet \(\{0, 1\}^3\).

**Below** Top line and the \(x, y, X\) are not there- Visual Aid.

The triple \((3, 4, \{0, 1, 2, 4, 7\})\) is represented by

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>(y)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>(X)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Note** After we see 0001 for \(x\) we DO NOT CARE what happens next. The *’s can be filled in with 0’s or 1’s and the string of symbols from \(\{0, 1\}^3\) above would still represent \((3, 4, \{0, 1, 2, 4, 7\})\).
**Thm** For all WS1S formulas $\phi$ the set $TRUE_\phi$ is regular.

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

\[(Q_1X_1) \cdots (Q_nX_n)(Q_{n+1}x_1) \cdots (Q_{n+m}x_m)[\phi(x_1, \ldots, x_m, X_1, \ldots, X_n)]\]

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.