15-453
FORMAL LANGUAGES,
AUTOMATA AND
COMPUTABILITY
THURSDAY APRIL 3

REVIEW for Midterm 2

TUESDAY April 8
Definition: A Turing Machine is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$
TURING MACHINE

FINITE STATE CONTROL

INPUT

INFINITE TAPE
CONFIGURATIONS

11010q₇001110

q₇

1 1 0 1 0 0 0 1 1 0
An accepting computation history is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

1. $C_1$ is the start configuration, $C_1 = q_0w$.
2. $C_k$ is an accepting configuration, $C_k = uq_{\text{accept}}v$.
3. Each $C_i$ follows from $C_{i-1}$ via the transition function $\delta$.

A rejecting computation history is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

1. $C_1$ is the start configuration.
2. $C_k$ is a rejecting configuration, $C_k = uq_{\text{reject}}v$.
3. Each $C_i$ follows from $C_{i-1}$.
M accepts $w$ if and only if there is an accepting computation history that starts with $C_1=q_0w$. 
We can encode a TM as a string of 0s and 1s

\[( (p,a), (q,b,L) ) = 0^p10^a10^q10^b10 \]
NB. We assume a given convention of describing TMs by strings in $\Sigma^*$.

We may assume that any string in $\Sigma^*$ describes some TM:

Either the string describes a TM by the convention,

or if the string is gibberish at some point then the “machine” just halts if/when a computation gets to that point.
A language is called \textbf{Turing-recognizable} or \textbf{semi-decidable} or \textbf{recursively enumerable} (\textbf{r.e.}) if some TM recognizes it.

A language is called \textbf{decidable} or \textbf{recursive} if some TM decides it.

Languages over \{0,1\} are classified into:

- \textbf{semi-decidable} (\textbf{r.e.}) languages
- \textbf{decidable} (\textbf{recursive}) languages
\[ A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \]

\textbf{\( A_{TM} \) is undecidable:} (proof by contradiction)

Assume machine \( H \) decides \( A_{TM} \)

\[
H((M,w)) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Reject} & \text{if } M \text{ does not accept } w 
\end{cases}
\]

Construct a new TM \( D \) as follows: on input \( M \), run \( H \) on \((M,M)\) and output the opposite of \( H \)

\[
D(M) = \begin{cases} 
\text{Reject} & \text{if } M \text{ accepts } M \\
\text{Accept} & \text{i.e. if } H(M,M) \text{ accepts} \\
\text{Accept} & \text{if } M \text{ does not accept } M \\
\text{Reject} & \text{i.e. if } H(M,M) \text{ rejects}
\end{cases}
\]
\[ \mathbf{A_{TM}} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \]

\[ \mathbf{A_{TM}} \text{ is undecidable: (proof by contradiction)} \]

Assume machine \( H \) decides \( \mathbf{A_{TM}} \)

\[ H( (M,w) ) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Reject} & \text{if } M \text{ does not accept } w 
\end{cases} \]

Construct a new TM \( D \) as follows: on input \( M \), run \( H \) on \( (M,M) \) and output the opposite of \( H \)

\[ D( D ) = \begin{cases} 
\text{Reject} & \text{if } D \text{ accepts } D \\
\text{Accept} & \text{if } D \text{ does not accept } D 
\end{cases} \]
\[ A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \]

**\( A_{TM} \) is undecidable:** (constructive proof & subtle)

Assume machine \( H \) semi-decides \( A_{TM} \)

\[
H( (M,w) ) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Rejects or Loops} & \text{otherwise}
\end{cases}
\]

Construct a new TM \( D_H \) as follows:
on input \( M \), run \( H \) on \( (M,M) \) and output the “opposite” of \( H \) whenever possible.
$D_H(D_H) = \begin{cases} 
\text{Reject if } D_H \text{ accepts } D_H \\
(i.e. \text{ if } H(D_H, D_H) = \text{Accept}) \\
\text{Accept if } D_H \text{ reject } D_H \\
(i.e. \text{ if } H(D_H, D_H) = \text{Reject}) \\
\text{Loops if } D_H \text{ loops or } D_H \\
(i.e. \text{ if } H(D_H, D_H) \text{ loops}) 
\end{cases}$

Note: There is no contradiction here!

$D_H$ loops on $D_H$

We can effectively construct an instance which does not belong to $A_{TM}$ (namely, $(D_H, D_H)$) but $H$ fails to tell us that.
Theorem: Let $T$ be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. Then there is a Turing machine $R$ that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every string $w$,

$$r(w) = t(<R>, w)$$
Theorem: Let $T$ be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. Then there is a Turing machine $R$ that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every string $w$, $r(w) = t(<R>, w)$.
Recursion Theorem says: A Turing machine can obtain its own description (code), and compute with it.

We can use the operation: "Obtain your own description" in pseudocode!

Given a computable t, we can get a computable r such that \( r(w) = t(<R>,w) \) where \(<R>\) is a description of r.

INSIGHT: T (or t) is really R (or r)
Theorem: $A_{TM}$ is undecidable

Proof (using the Recursion Theorem):

Assume $H$ decides $A_{TM}$  (Informal Proof)

Construct machine $R$ such that on input $w$:

1. Obtains its own description $<R>$
2. Runs $H$ on $(<R>, w)$ and flips the output

Running $R$ on input $w$ always does the opposite of what $H$ says it should!
**Theorem:** $A_{TM}$ is undecidable

**Proof (using the Recursion Theorem):**

Assume $H$ decides $A_{TM}$ (Formal Proof)

Let $T_H(x, w) =$

Reject if $H(x, w)$ accepts
Accept if $H(x, w)$ rejects

(Here $x$ is viewed as a code for a TM)

By the *Recursion Theorem*, there is a TM $R$ such that:

$R(w) =$

Reject if $H(<R>, w)$ accepts
Accept if $H(<R>, w)$ rejects

**Contradiction!**
Theorem: \( \text{MIN}_{\text{TM}} \) is not RE.

Proof (using the Recursion Theorem):

\[
\text{MIN}_{\text{TM}} = \{<M> | M \text{ is a minimal TM, wrt } |<M>|\}
\]
Theorem: \( \text{MIN}_{TM} \) is not RE.

Proof (using the Recursion Theorem):

Assume \( E \) enumerates \( \text{MIN}_{TM} \) (Informal Proof)

Construct machine \( R \) such that on input \( w \):

1. Obtains its own description \(<R>\)

2. Runs \( E \) until a machine \( D \) appears with a longer description than of \( R \)

3. Simulate \( D \) on \( w \)

Contradiction. Why?
MIN\_TM = \{\langle M \rangle | M \text{ is a minimal TM, wrt } |\langle M \rangle|\}

**Theorem:** MIN\_TM is not RE.

**Proof (using the Recursion Theorem):**

Assume E enumerates MIN\_TM  

Let T_E(x, w) = D(w) where \langle D \rangle is first in E’s enumeration s.t. |\langle D \rangle| > |x|

By the *Recursion Theorem*, there is a TM R such that:

R(w) = T_E(\langle R \rangle, w) = D(w)

where \langle D \rangle is first in E’s enumeration s.t. |\langle D \rangle| > |\langle R \rangle|

Contradiction. Why?
Theorem: Let $f : \Sigma^* \rightarrow \Sigma^*$ be a computable function. There is a TM $R$ such that $f(<R>)$ describes a TM that is equivalent to $R$.

Proof: Pseudocode for the TM $R$:

On input $w$:

1. Obtain the description $<R>$
2. Let $g = f(<R>)$ and interpret $g$ as a code for a TM $G$
3. Accept $w$ iff $G(w)$ accepts
THE FIXED-POINT THEOREM

Theorem: Let \( f : \Sigma^* \rightarrow \Sigma^* \) be a computable function. There is a TM \( R \) such that \( f(<R>) \) describes a TM that is equivalent to \( R \).

Proof: Let \( T_f(x, w) = G(w) \) where \( <G> = f(x) \) (Here \( f(x) \) is viewed as a code for a TM)

By the Recursion Theorem, there is a TM \( R \) such that:

\[ R(w) = T_f(<R>, w) = G(w) \text{ where } <G> = f(<R>) \]

Hence \( R \equiv G \) where \( <G> = f(<R>) \), ie \( <R> \equiv f(<R>) \)

So \( R \) is a fixed point of \( f \)!
THE FIXED-POINT THEOREM

**Theorem:** Let \( f : \Sigma^* \rightarrow \Sigma^* \) be a computable function. There is a TM \( R \) such that \( f(<R>) \) describes a TM that is equivalent to \( R \).

**Example:**

Suppose a virus flips the first bit of each word \( w \) in \( \Sigma^* \) (or in each TM). Then there is a TM \( R \) that “remains uninfected”.


Theorem: Let $T$ be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$.

Then there is a Turing machine $R$ that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every string $w$,

$$r(w) = t(<R>, w)$$

**Diagram:**

- $(a,b) \rightarrow T \rightarrow t(a,b)$
- $w \rightarrow R \rightarrow t(<R>,w)$
**The Recursion Theorem**

**Theorem:** Let $T$ be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. Then there is a Turing machine $R$ that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every string $w$,

$$r(w) = t(<R>, w)$$

So first, need to show how to construct a TM that computes its own description (ie code).
Lemma: There is a computable function $q : \Sigma^* \rightarrow \Sigma^*$, where for any string $w$, $q(w)$ is the *description (code)* of a TM $P_w$ that on any input, prints out $w$ and then accepts $w$. 

TM $Q$ computes $q$.
A TM SELF THAT PRINTS <SELF>

B (<M>) = < P<sub>M</sub> M > where P<sub>M</sub> M (w’) = M (<M>)

So, B (<B>) = < P<sub>B</sub> B > where P<sub>B</sub> B (w’) = B (<B>)

Now, P<sub>B</sub> B (w’) = B(<B>) = <P<sub>B</sub> B >

So, let SELF = P<sub>B</sub> B
A TM SELF THAT PRINTS <SELF>
A TM SELF THAT PRINTS <SELF>
A NOTE ON SELF REFERENCE

Suppose in general we want to design a program that prints its own description. How?

Print this sentence.

Print two copies of the following (the stuff inside quotes), and put the second copy in quotes:
“Print two copies of the following (the stuff inside quotes), and put the second copy in quotes:”

= B

= P_{<B>
Let \( f : \Sigma^* \rightarrow \Sigma^* \) be a computable function such that \( w \in A \iff f(w) \in B \).

Say: \( A \) is Mapping Reducible to \( B \)
Write: \( A \leq_m B \)
(also, \( \neg A \leq_m \neg B \) (why?))
Let $f : \Sigma^* \rightarrow \Sigma^*$ be a *computable function* such that $w \in A \iff f(w) \in B$

So, if $B$ is (semi) decidable, then so is $A$

(And if $\neg B$ is (semi) decidable, then so is $\neg A$)
$$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$$

$$HALT_{TM} = \{ (M,w) \mid M \text{ is a TM that halts on string } w \}$$

$$f: (M,w) \rightarrow (M', w)$$ where

- If $$M(s)$$ accepts, $$M'(s) = M(s)$$
- Loops otherwise

So, $$(M, w) \in A_{TM} \iff (M', w) \in HALT_{TM}$$
$A_{\text{TM}} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \}$

$E_{\text{TM}} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \}$

$f: (M, w) \rightarrow M_w$ where $M_w(s) = M(w)$ if $s = w$, Loops otherwise

$\text{So, } (M, w) \in A_{\text{TM}} \iff M_w \in \overline{E_{\text{TM}}}$
$$A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \}$$

$$REG_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \}$$

$f: (M, w) \rightarrow M'w$ where $M'_w(s) = \text{accept}$ if $s = 0^n1^n$, $M(w)$ otherwise

So, $(M, w) \in A_{TM} \iff M'_w \in REG_{TM}$
$E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \}$

$EQ_{TM} = \{( M, N) \mid M, N \text{ are TMs and } L(M) = L(N)\}$

$f: M \rightarrow (M, M\emptyset)$ where $M\emptyset(s) = \text{Loops}$

So, $M \in E_{TM} \iff (M, M\emptyset) \in EQ_{TM}$
$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

$ALL_{PDA} = \{ P \mid P \text{ is a PDA and } L(P) = \Sigma^* \}$

$f: (M,w) \rightarrow \text{PDA } P_w \text{ where } s \in \Sigma^*$

$P_w (s) = \text{accept} \text{ iff } s \text{ is NOT an accepting computation of } M(w)$

So, $(M, w) \in A_{TM} \iff P_w \in \neg ALL_{PDA}$
\[ \mathcal{A}_{\text{TM}} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \]

\[ \mathcal{FPCP} = \{ P \mid P \text{ is a set of dominos with a match starting with the first domino} \} \]

Construct \( f: (M,w) \rightarrow P_{(M,w)} \) such that

\[ (M, w) \in \mathcal{A}_{\text{TM}} \iff P_{(M,w)} \in \mathcal{FPCP} \]
\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

\[ PCP = \{ P \mid P \text{ is a set of dominos with a match} \} \]

Construct \( f: (M, w) \rightarrow P_{(M, w)} \) such that

\[ (M, w) \in A_{TM} \iff P_{(M, w)} \in PCP \]
\[ A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \]

\[ \text{HALT}_{TM} = \{ (M,w) \mid M \text{ is a TM that halts on string } w \} \]

\[ E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \} \]

\[ \text{REG}_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \} \]

\[ \text{EQ}_{TM} = \{ (M, N) \mid M, N \text{ are TMs and } L(M) = L(N) \} \]

\[ \text{ALL}_{PDA} = \{ P \mid P \text{ is a PDA and } L(P) = \Sigma^* \} \]

\[ \text{PCP} = \{ P \mid P \text{ is a set of dominos with a match} \} \]

ALL UNDECIDABLE

Use Reductions to Prove
\[
A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \} \\
HALT_{TM} = \{ (M,w) \mid M \text{ is a TM that halts on string } w \} \\
E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \} \\
\neg E_{TM} \\
R E G_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \} \\
EQ_{TM} = \{ (M, N) \mid M, N \text{ are TMs and } L(M) = L(N) \} \\
\neg EQ_{TM} \\
A L L_{PDA} = \{ P \mid P \text{ is a PDA and } L(P) = \Sigma^* \} \\
\neg ALL_{PDA} \\
P C P = \{ P \mid P \text{ is a set of dominos with a match} \} \\
\]

ALL UNDECIDABLE

Use Reductions to Prove

Which are SEMI-DECIDABLE?
RICE’S THEOREM

Let $L$ be a language over Turing machines. Assume that $L$ satisfies the following properties:

1. For any TMs $M_1$ and $M_2$, where $L(M_1) = L(M_2)$, $M_1 \in L$ if and only if $M_2 \in L$

2. There are TMs $M_1$ and $M_2$, where $M_1 \in L$ and $M_2 \notin L$

Then $L$ is undecidable

EXTREMELY POWERFUL!
RICE'S THEOREM

Let $L$ be a language over Turing machines. Assume that $L$ satisfies the following properties:

1. For any TMs $M_1$ and $M_2$, where $L(M_1) = L(M_2)$, $M_1 \in L$ if and only if $M_2 \in L$
2. There are TMs $M_1$ and $M_2$, where $M_1 \in L$ and $M_2 \notin L$

Then $L$ is undecidable

$FIN_{TM} = \{ M \mid M$ is a TM and $L(M)$ is finite $\}$
$E_{TM} = \{ M \mid M$ is a TM and $L(M) = \emptyset \}$
$REG_{TM} = \{ M \mid M$ is a TM and $L(M)$ is regular $\}$
Proof: Show $L$ is undecidable.

Show: $A_{TM}$ is mapping reducible to $L$. 

\[ \Sigma^* \]

\[ A_{TM} \]

\[ (M,w) \]

\[ f \]

\[ L \]

\[ (M,w) \]

\[ \Sigma^* \]
Proof: Show $L$ is undecidable

Show: $A_{TM}$ is mapping reducible to $L$
RICE’S THEOREM

Proof:

Define $M_{\emptyset}$ to be a TM that never halts.

Assume, WLOG, that $M_{\emptyset} \notin L$ Why?

Let $M_1 \in L$ (such $M_1$ exists, by assumption).

Show $A_{TM}$ is mapping reducible to

$L :$

$\text{Map } (M, w) \rightarrow M_w$ where

$M_w(s) = \text{accepts if both } M(w) \text{ and } M_1(s) \text{ accept loops otherwise}$

What is the language of $M_w$ ?
$A_{TM}$ is mapping reducible to $L$
Corollary: The following languages are undecidable.

\[ E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \} \]
\[ REG_{TM} = \{ M \mid M \text{ is TM and } L(M) \text{ is regular} \} \]
\[ FIN_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is finite} \} \]
\[ DEC_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is decidable} \} \]
Finite State Control

Infinite Tape

Is (M, w) in $A_{TM}$?

$q_{YES}$

ORACLE TMs

Input

Infinite Tape
A Turing Reduces to B

We say A is decidable in B if there is an oracle TM M with oracle B that decides A

$A \leq_T B$

$\leq_T$ is transitive
Theorem: If $A \leq_m B$ then $A \leq_T B$
But in general, the converse doesn’t hold!

Proof:
If $A \leq_m B$ then there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B$$

We can thus use an oracle for $B$ to decide $A$

Theorem: $\neg\text{HALT}_{TM} \leq_T \text{HALT}_{TM}$

Theorem: $\neg\text{HALT}_{TM} \leq_m \text{HALT}_{TM}$

WHY?
THE ARITHMETIC HIERARCHY

\[ \Delta^0_{n+1} = \{ \text{decidable sets} \} \quad \text{(sets = languages)} \]

\[ \Sigma^0_{n+1} = \{ \text{semi-decidable sets} \} \]

\[ \Sigma^0_{n} = \{ \text{sets semi-decidable in some } B \in \sum^0_n \} \]

\[ \Delta^0_{n+1} = \{ \text{sets decidable in some } B \in \sum^0_n \} \]

\[ \Pi^0_n = \{ \text{complements of sets in } \sum^0_n \} \]
Semi-decidable Languages
\[ \Sigma^0_1 \]

Decidable Languages
\[ \Delta^0_1 \cap \Pi^0_1 \]

Co-semi-decidable Languages
\[ \Sigma^0_3 \]
\[ \Pi^0_3 \]

\[ \Sigma^0_2 \]
\[ \Pi^0_2 \]
\[ \Sigma^0_3 \]
\[ \Pi^0_3 \]
Definition: A decidable predicate \( R(x,y) \) is some proposition about \( x \) and \( y \), where there is a TM \( M \) such that

for all \( x, y \), \( R(x,y) \) is TRUE \( \Rightarrow \) \( M(x,y) \) accepts
\( R(x,y) \) is FALSE \( \Rightarrow \) \( M(x,y) \) rejects

We say \( M \) “decides” the predicate \( R \).

**EXAMPLES:**
- \( R(x,y) = \text{“} x + y \text{ is less than 100”} \)
- \( R(<N>,y) = \text{“} N \text{ halts on } y \text{ in at most 100 steps”} \)
- Kleene’s T predicate, \( T(<M>, x, y) \): \( M \) accepts \( x \) in \( y \) steps.

1. \( x, y \) are positive integers or elements of \( \Sigma^* \)
Theorem: A language \( A \) is semi-decidable if and only if there is a decidable predicate \( R(x, y) \) such that \( A = \{ x \mid \exists y \ R(x,y) \} \)

Proof:

(1) If \( A = \{ x \mid \exists y \ R(x,y) \} \) then \( A \) is semi-decidable
Because we can enumerate over all \( y \)'s

(2) If \( A \) is semi-decidable, then \( A = \{ x \mid \exists y \ R(x,y) \} \)
Let \( M \) semi-decide \( A \) and
Let \( R_{<M>}(x,y) \) be the Kleene T- predicate: \( T(<M>, x, y): \) TM \( M \) accepts \( x \) in \( y \) steps (\( y \) interpreted as an integer)
\( R_{<M>} \) is a decidable predicate (why?)
So \( x \in A \) if and only if \( \exists y \ R_{<M>} (x,y) \) is true.
Theorem

\( \Sigma^0_1 = \{ \text{semi-decidable sets} \} \)
\( = \text{languages of the form } \{ x \mid \exists y \ R(x,y) \} \)

\( \Pi^0_1 = \{ \text{complements of semi-decidable sets} \} \)
\( = \text{languages of the form } \{ x \mid \forall y \ R(x,y) \} \)

\( \Delta^0_1 = \{ \text{decidable sets} \} \)
\( = \Sigma^0_1 \cap \Pi^0_1 \)

Where \( R \) is a decidable predicate
Theorem

\[ \Sigma^0_2 = \{ \text{sets semi-decidable in some semi-dec. B} \} \]
= languages of the form \( \{ x \mid \exists y_1 \forall y_2 \ R(x,y_1,y_2) \} \)

\[ \Pi^0_2 = \{ \text{complements of } \Sigma^0_2 \text{ sets} \} \]
= languages of the form \( \{ x \mid \forall y_1 \exists y_2 \ R(x,y_1,y_2) \} \)

\[ \Delta^0_2 = \Sigma^0_2 \cap \Pi^0_2 \]

Where \( R \) is a decidable predicate
Theorem

$$\sum^0_n = \text{languages } \{ x \mid \exists y_1 \forall y_2 \exists y_3 \ldots Q y_n \ R(x,y_1,\ldots,y_n) \}$$

$$\Pi^0_n = \text{languages } \{ x \mid \forall y_1 \exists y_2 \forall y_3 \ldots Q y_n \ R(x,y_1,\ldots,y_n) \}$$

$$\Delta^0_n = \sum^0_n \cap \Pi^0_n$$

Where $R$ is a decidable predicate
Example

\[ \sum_1^0 = \text{languages of the form } \{ x \mid \exists y \ R(x,y) \} \]

We know that \( A_{\text{TM}} \) is in \( \sum_1^0 \)

Why?

Show it can be described in this form:

\[ A_{\text{TM}} = \{ <(M,w)> \mid \exists t \ [M \text{ accepts } w \text{ in } t \text{ steps} ] \} \]

Decidable predicate

\[ A_{\text{TM}} = \{ <(M,w)> \mid \exists t \ T (<M>, w, t ) \} \]

Decidable predicate

\[ A_{\text{TM}} = \{ <(M,w)> \mid \exists v \ (v \text{ is an accepting computation history of } M \text{ on } w) \} \]
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \text{ATM} \]

\[ \Delta_0^0 \]

\[ \Delta_2^0 \]

\[ \Delta_3^0 \]

\[ \Sigma_1^0 \]

\[ \Sigma_2^0 \]

\[ \Sigma_3^0 \]

\[ \Pi_1^0 \]

\[ \Pi_2^0 \]

\[ \Pi_3^0 \]

\[ \sum_2^0 \cap \pi_2^0 \]
Example

$\Pi^0_1 = \text{languages of the form } \{ x \mid \forall y \ R(x,y) \} \}$

Show that $\text{EMPTY (ie, } E_{\text{TM}}) = \{ M \mid L(M) = \emptyset \}$ is in $\Pi^0_1$

$\text{EMPTY} = \{ M \mid \forall w \forall t [M \text{ doesn’t accept } w \text{ in } t \text{ steps}] \}$

two quantifiers??

decidable predicate
Example

$\Pi^0_1 = \text{languages of the form } \{ x \mid \forall y \ R(x, y) \}$

Show that $\text{EMPTY (ie, } E_{TM}) = \{ M \mid L(M) = \emptyset \}$ is in $\Pi^0_1$

$\text{EMPTY} = \{ M \mid \forall w \forall t [ \neg T(<M>, w, t) ] \}$

two quantifiers??
decidable predicate
Theorem. There is a 1-1 and onto computable function \(<\ ,\ >: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*\) and computable functions \(\pi_1\) and \(\pi_2\) : \(\Sigma^* \rightarrow \Sigma^*\) such that

\[z = \langle w, t \rangle \Rightarrow \pi_1(z) = w \text{ and } \pi_2(z) = t\]

\[\text{EMPTY} = \{ M \mid \forall w \forall t [M \text{ doesn’t accept } w \text{ in } t \text{ steps}] \}\]

\[\text{EMPTY} = \{ M \mid \forall z [M \text{ doesn’t accept } \pi_1(z) \text{ in } \pi_2(z) \text{ steps}] \}\]

\[\text{EMPTY} = \{ M \mid \forall z [\neg T(\langle M \rangle, \pi_1(z), \pi_2(z))] \}\]
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \sum_0^0 \]
\[ \sum_2^0 \]
\[ \sum_3^0 \]

\[ \Delta_0^0 \]
\[ \Delta_2^0 \]
\[ \Delta_3^0 \]

\[ \sum_2^0 \cap \pi_2^0 \]
= \[ \text{ATM EMPTY} \]

\[ \pi_1^0 \]
\[ \pi_2^0 \]
\[ \pi_3^0 \]
Example

\( \Pi_2^0 = \text{languages of the form } \{ x | \forall y \exists z \ R(x,y,z) \} \)

Show that TOTAL = \{ M | M \text{ halts on all inputs} \} is in \( \Pi_2^0 \)

TOTAL = \{ M | \forall w \exists t [M \text{ halts on } w \text{ in } t \text{ steps}] \}
Example

\[ \Pi^0_2 = \text{languages of the form } \{ x | \forall y \exists z \ R(x,y,z) \} \]

Show that TOTAL = \{ M | M \text{ halts on all inputs} \} is in \( \Pi^0_2 \)

TOTAL = \{ M | \forall w \exists t [ T(<M>, w, t) ] \}
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \sum^0_1 \]

\[ \Delta^0_2 \]

\[ \sum^0_2 \cap \Pi^0_2 \]

\[ \Delta^0_3 \]

\[ \Pi^0_3 \]

\[ \Pi^0_2 \]

\[ \Pi^0_1 \]

Total

Empty
Example

\[ \sum^0_2 = \text{languages of the form } \{ x \mid \exists y \forall z \ R(x,y,z) \} \]

Show that \( \text{FIN} = \{ M \mid \text{L(M) is finite} \} \) is in \( \sum^0_2 \)

\( \text{FIN} = \{ M \mid \exists n \forall w \forall t [\text{Either } |w| < n, \text{ or } M \text{ doesn’t accept } w \text{ in } t \text{ steps}] \} \)

\( \text{FIN} = \{ M \mid \exists n \forall w \forall t ( |w| < n \lor \neg T(<M>,w,t) ) \} \)

decidable predicate
Decidable languages

Semi-decidable languages

\[ \Delta^0_1 \subseteq \Sigma^0_2 \cap \Pi^0_2 \]

\[ \Delta^0_2 \] = \[ \sum^0_2 \cap \pi^0_2 \]

\[ \sum^0_3 \]

\[ \Delta^0_3 \]

\[ \sum^0_0 \]

\[ \pi^0_0 \]

\[ \pi^0_1 \]

\[ \pi^0_2 \]

\[ \pi^0_3 \]
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \sum_0 \]

\[ \Delta_1 \]

\[ \Delta_2 \]

\[ \Delta_3 \]

\[ \Pi_0 \]

\[ \Pi_1 \]

\[ \Pi_2 \]

\[ \Pi_3 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]

\[ \Sigma_0 \]

\[ \Pi_0 \]
Decidable languages

Semi-decidable languages

\[ \Sigma^0_3 \]

\[ \Sigma^0_2 \]

\[ \Sigma^0_1 \]

\[ \Delta^0_3 \]

\[ \Delta^0_2 \]

\[ \Delta^0_1 \]

\[ \sum^0_2 \cap \Pi^0_2 \]

\[ \Sigma^0_2 \cap \Pi^0_2 \]

\[ \Delta^0_2 \]

\[ \Pi^0_3 \]

\[ \Pi^0_2 \]

\[ \Pi^0_1 \]

\[ \text{ATM} \]

\[ \text{TOTAL} \]

\[ \text{FIN} \]

\[ \text{EMPTY} \]

\[ \text{REG} \]

Co-semi-decidable languages
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \Sigma^0_3 \]

\[ \Sigma^0_2 \]

\[ \Sigma^0_1 \]

\[ \Delta^0_3 \]

\[ \Delta^0_2 \]

\[ \Delta^0_1 \]

\[ \Pi^0_3 \]

\[ \Pi^0_2 \]

\[ \Pi^0_1 \]

\[ \sum^0_2 \cap \Pi^0_2 \]

\[ = \]

\[ \emptyset \]

\[ \mathsf{ATM} \]

\[ \mathsf{TOTAL} \]

\[ \mathsf{FIN} \]

\[ \mathsf{DEC} \]

\[ \mathsf{EMPTY} \]
Decidable languages

Semi-decidable languages

Co-semi-decidable languages

\[ \Sigma_3^0 \]

\[ \Sigma_2^0 \]

\[ \Sigma_1^0 \]

\[ \Delta_3^0 \]

\[ \Delta_2^0 \]

\[ \Delta_1^0 \]

\[ \Pi_3^0 \]

\[ \Pi_2^0 \]

\[ \Pi_1^0 \]

CFL

FIN

\[ \text{TOTAL} \]

\[ \text{EMPTY} \]

\[ \text{ATM} \]

\[ \text{FIN} \]

\[ \text{CFL} \]

\[ \Sigma_2^0 \cap \Pi_2^0 \]
Each is \( m \)-complete for its level in hierarchy and cannot go lower (by next Theorem, which shows the hierarchy does not collapse).
ORACLES not all powerful

The following problem cannot be decided, even by a TM with an oracle for the Halting Problem:

\[
\text{SUPERHALT} = \{ (M,x) | M, \text{ with an oracle for the Halting Problem, halts on } x \}
\]

Can use diagonalization here!

Suppose \( H \) decides \( \text{SUPERHALT} \) (with oracle)
Define \( D(X) = \text{“if } H(X,X) \text{ accepts (with oracle) then LOOP, else ACCEPT.”} \)

\( D(D) \) halts \( \iff \) \( H(D,D) \) accepts \( \iff \) \( D(D) \) loops...
**Theorem:** The arithmetic hierarchy is strict. That is, the nth level contains a language that isn’t in any of the levels below n.

**Proof IDEA:** Same idea as the previous slide.

\[ \text{SUPERHALT}^0 = \text{HALT} = \{ (M,x) \mid M \text{ halts on } x \}. \]

\[ \text{SUPERHALT}^1 = \{ (M,x) \mid M, \text{ with an oracle for the Halting Problem, halts on } x \} \]

\[ \text{SUPERHALT}^n = \{ (M,x) \mid M, \text{ with an oracle for } \text{SUPERHALT}^{n-1}, \text{ halts on } x \} \]
Theorem:

1. The hierarchy is strict
2. Each of the languages is \textit{m-complete} for its class.

Proof Idea.

1. Let $A_{TM,1} = A_{TM}$

   $A_{TM, n+1} = \{(M, x) | M \text{ is an oracle machine with oracle } A_{TM} \text{ and } M \text{ accepts } x\}$

Then $A_{TM, n} \in \sum_0^n - \prod_0^n$
Theorem:

1. The hierarchy is strict

2. Each of the languages is \( m \)-complete for its class.

Proof.

2. Eg to show \( \text{FIN} \) is \( m \)-complete for \( \sum_2^0 \)

Need to show

a) \( \text{FIN} \in \sum_2^0 \)

\( \text{FIN} = \{ M \mid \exists n \ \forall x \ \forall t (|x| < n \text{ or } M \text{ does not accept } x \text{ in } t \text{ steps}) \} \)

b) For \( A \in \sum_2^0 \) then \( A \leq_m \text{FIN} \)
For $A \in \Sigma_2^0$, $A = \{ x \mid \exists y \forall z \ R(x,y,z) \}$

$\text{FIN} = \{ M \mid \text{L(M) is finite} \}$

Given input $w$:
For each $y$ of length $|w|$ or less, look for $z$ such that $\neg R(x,y,z)$ . If found for all such $y$, Accept. Otherwise keep on running.
For $A \in \Sigma^0_2$, $A = \{ x \mid \exists y \forall z \ R(x, y, z) \}$

$\text{FIN} = \{ M \mid L(M) \text{ is finite} \}$

- If $x \in A$, then $\exists y \forall z \ R(x, y, z)$, so when $|w| > |y|$, $M_x$ keeps on running, so $M_x \in \text{FIN}$.
- If $x \notin A$, then $\forall y \exists z \neg R(x, y, z)$, so $M_x$ recognizes $\Sigma^*$.
CAN WE QUANTIFY HOW MUCH INFORMATION IS IN A STRING?

A = 01010101010101010101010101010101

B = 110010011101110101101001011001011

Idea: The more we can “compress” a string, the less “information” it contains....
Definition: Let \( x \) in \( \{0,1\}^* \). The **shortest description** of \( x \), denoted as \( d(x) \), is the lexicographically shortest string \( <M,w> \) s.t. \( M(w) \) halts with \( x \) on tape.

Use pairing function to code \( <M,w> \)

Definition: The **Kolmogorov complexity** of \( x \), denoted as \( K(x) \), is \( |d(x)| \).
**KOLMOGOROV COMPLEXITY**

**Theorem:** There is a fixed $c$ so that for all $x$ in $\{0,1\}^*$, 
$$K(x) \leq |x| + c$$

“The amount of information in $x$ isn’t much more than $|x|$”

**Proof:** Define $M = “On w, halt.”$

On any string $x$, $M(x)$ halts with $x$ on its tape!

This implies
$$K(x) \leq |<M,x>| \leq 2|M| + |x| + 1 \leq c + |x|$$

(Note: $M$ is fixed for all $x$. So $|M|$ is constant)
**Theorem:** There is a fixed $c$ so that for all $x$ in $\{0,1\}^*$,

\[
K(xx) \leq K(x) + c
\]

“The information in $xx$ isn’t much more than that in $x$”

**Proof:** Let $N = “On <M,w>, let $s$=M(w). Print ss.””

Let $<M,w’>$ be the shortest description of $x$.

Then $<N,<M,w’>>$ is a description of $xx$

Therefore

\[
K(xx) \leq |<N,<M,w’>>| \leq 2|N| + K(x) + 1 \leq c + K(x)
\]
**Corollary:** There is a fixed $c$ so that for all $n$, and all $x \in \{0,1\}^*$,

$$K(x^n) \leq K(x) + c \log_2 n$$

“The information in $x^n$ isn’t much more than that in $x$”

**Proof:**

An intuitive way to see this:

Define $M$: “On $<x, n>$, print $x$ for $n$ times”.

Now take $<M, <x, n>>$ as a description of $x^n$.

In binary, $n$ takes $O(\log n)$ bits to write down, so we have $K(x) + O(\log n)$ as an upper bound on $K(x^n)$.
Recall: 
\[ A = 01010101010101010101010101010101 \]

Corollary: There is a fixed \( c \) so that for all \( n \), and all \( x \in \{0,1\}^* \),
\[ K(x^n) \leq K(x) + c \log_2 n \]

“The information in \( x^n \) isn’t much more than that in \( x \)”

Recall:
\[ A = 01010101010101010101010101010101 \]

For \( w = (01)^n \), \( K(w) \leq K(01) + c \log_2 n \)
**Theorem:** There is a fixed $c$ so that for all $x, y$ in $\{0,1\}^*$,

$$K(xy) \leq 2K(x) + K(y) + c$$

**Better:**

$$K(xy) \leq 2 \log K(x) + K(x) + K(y) + c$$
INCOMPRESSIBLE STRINGS

**Theorem:** For all \( n \), there is an \( x \in \{0,1\}^n \) such that \( K(x) \geq n \)

“There are incompressible strings of every length”

**Proof:**

\[
\text{(Number of binary strings of length } n) = 2^n
\]

\[
\leq \text{(Number of descriptions of length } < n) \leq \text{(Number of binary strings of length } < n) = 2^n - 1.
\]

Therefore: there’s at least one \( n \)-bit string that doesn’t have a description of length \( < n \).
INCOMPRESSIBLE STRINGS

**Theorem:** For all $n$ and $c$,

$$\Pr_{x \in \{0,1\}^n}[K(x) \geq n-c] \geq 1 - 1/2^c$$

“Most strings are fairly incompressible”

**Proof:**

(Number of binary strings of length $n$) = $2^n$

(Number of descriptions of length < $n-c$)

\leq (Number of binary strings of length < $n-c$)

= $2^{n-c} - 1$.

So the probability that a random $x$ has $K(x) < n-c$ is at most $(2^{n-c} - 1)/2^n < 1/2^c$. 
DETERMINING COMpressibility

Can an algorithm help us compress strings?
Can an algorithm tell us when a string is compressible?

COMPRESS = {(x,c) | K(x) ≤ c}

Theorem: COMPRESS is undecidable!

Berry Paradox: “The first string whose shortest description cannot be written in less than fifteen words.”
DETERMINING COMPRESSIBILITY

COMPRESS = \{ (x,n) \mid K(x) \leq n \}

**Theorem:** COMPRESS is undecidable!

**Proof:**

\[ M = \text{"On input } x \in \{0,1\}^\ast, \]
\[ \text{Interpret } x \text{ as integer } n. (|x| \leq \log n) \]
\[ \text{Find first } y \in \{0,1\}^\ast \text{ in lexicographical order,} \]
\[ \text{s.t. } (y,n) \not\in \text{ COMPRESS, then print } y \text{ and} \]
\[ \text{halt."} \]

\[ M(x) \text{ prints the first string } y^\ast \text{ with } K(y^\ast) > n. \]

Thus \(<M,x>\) describes \(y^\ast\), and \(|<M,x>| \leq c + \log n\)

So \(n < K(y^\ast) \leq c + \log n\). **CONTRADICTION!**
**Theorem:** K is not computable

**Proof:**

M = “On input \( x \in \{0,1\}^* \),

Interpret \( x \) as integer \( n \). (\( |x| \leq \log n \))

Find first \( y \in \{0,1\}^* \) in lexicographical order, s. t. \( K(y) > n \), then print \( y \) and halt.”

\( M(x) \) prints the first string \( y^* \) with \( K(y^*) > n \).

Thus \( <M,x> \) describes \( y^* \), and \( |<M,x>| \leq c + \log n \)

So \( n < K(y^*) \leq c + \log n \). CONTRADICTION!
What about other measures of compressibility?

For example:

- the smallest DFA that recognizes \{x\}
- the shortest grammar in Chomsky normal form that generates the language \{x\}
SO WHAT CAN YOU DO WITH THIS?

Many results in mathematics can be proved very simply using incompressibility.

**Theorem:** There are infinitely many primes.

**IDEA:** Finitely many primes ⇒ can compress everything!

**Proof:** Suppose not. Let $p_1, \ldots, p_k$ be the primes. Let $x$ be incompressible. Think of $n = x$ as integer. Then there are $e_i$ s.t.

$$n = p_1^{e_1} \cdots p_k^{e_k}$$

For all $i$, $e_i \leq \log n$, so $|e_i| \leq \log \log n$

Can describe $n$ (and $x$) with $k \log \log n + c$ bits! But $x$ was incompressible… **CONTRADICTION!**
Definition: Let M be a TM that halts on all inputs. The running time or time complexity of M is a function \( f : \mathbb{N} \rightarrow \mathbb{N} \), where \( f(n) \) is the maximum number of steps that M uses on any input of length \( n \).

Definition: \( \text{TIME}(t(n)) = \{ L \mid L \text{ is a language decided by a } O(t(n)) \text{ time Turing Machine} \} \)

\[
P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)
\]
Definition: A Non-Deterministic TM is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\emptyset \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\emptyset \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L,R\})}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$
**Definition:** $\text{NTIME}(t(n)) = \{ L \mid L \text{ is decided by a } O(t(n))-\text{time non-deterministic Turing machine} \}$

$\text{TIME}(t(n)) \subseteq \text{NTIME}(t(n))$

$\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$
Theorem: $L \in NP \iff$ if there exists a poly-time Turing machine $V$ with

$L = \{ x \mid \exists y \ [ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} ] \}$

Proof:

(1) If $L = \{ x \mid \exists y \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$ then $L \in NP$

Non-deterministically guess $y$ and then run $V(x,y)$

(2) If $L \in NP$ then $L = \{ x \mid \exists y \ |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$

Let $N$ be a non-deterministic poly-time TM that decides $L$, define $V(x,y)$ to accept iff $y$ is an accepting computation history of $N$ on $x$
A language is in NP if and only if there exist “polynomial-length proofs” for membership to the language

P = the problems that can be efficiently solved
NP = the problems where proposed solutions can be efficiently verified

P = NP?

Can Problem Solving Be Automated?

$$$

A Clay Institute Millennium Problem
**POLY-TIME REDUCIBILITY**

$f : \Sigma^* \rightarrow \Sigma^*$ is a **polynomial time computable function** if some poly-time Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.

Language $A$ is polynomial time reducible to language $B$, written $A \leq_P B$, if there is a poly-time computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that:

\[ w \in A \iff f(w) \in B \]

$f$ is called a polynomial time reduction of $A$ to $B$.

**Theorem:** If $A \leq_P B$ and $B \in P$, then $A \in P$.
SAT = \{ \phi \mid (\exists y)[y \text{ is a satisfying assignment to } \phi \\
and \phi \text{ is a boolean formula }] \} \\

3SAT = \{ \phi \mid (\exists y)[y \text{ is a satisfying assignment to } \phi \\
and \phi \text{ is in 3cnf }] \}
Theorem (Cook-Levin): SAT and 3-SAT are NP-complete

1. SAT ∈ NP:
   A satisfying assignment is a “proof” that a formula is satisfiable!

2. SAT is NP-hard:
   Every language in NP can be polytime reduced to SAT (complex formula)

Corollary: SAT ∈ P if and only if P = NP
Assume a reasonable encoding of graphs (example: the adjacency matrix is reasonable)

CLIQUE = \{ (G,k) \mid G \text{ is an undirected graph with a } k\text{-clique} \} 

**Theorem:** CLIQUE is NP-Complete

(1) CLIQUE ∈ NP

(2) 3SAT ≤_P CLIQUE
\( (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2) \)

**#nodes = 3(# clauses)**

**k = #clauses**
VERTEX-COVER = \{ (G,k) \mid G \text{ is an undirected graph with a } k\text{-node vertex cover} \}

**Theorem:** VERTEX-COVER is NP-Complete

(1) VERTEX-COVER $\in$ NP

(2) 3SAT $\leq_{P}$ VERTEX-COVER
\[(x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)\]

Variables and negations of variables

\[k = 2(\text{#clauses}) + (\text{#variables})\]
HAMPATH = \{ (G,s,t) \mid G \text{ is an directed graph with a Hamilton path from } s \text{ to } t \}\}

**Theorem:** HAMPATH is NP-Complete

1. HAMPATH ∈ NP
2. 3SAT ≤ₚ HAMPATH

Proof is in Sipser, Chapter 7.5
$\exists$ SAT $\leq_p$ HAM PATH

$\varphi = c_1 \land c_2 \land \ldots \land c_j \land \ldots \land c_k$

$c_j$, CLAUSE

$x_1, \ldots, x_k$, VARIABLES

$G$: S

If $x_i \in C_j$ (ARROWS REVERSED IF $\overline{x_i} \in C_j$)

$3k + 1$ NODES

$\varphi$ SATISFIABLE WITH SOME TRUTH ASSIGNMENT. ZIG ZAG if $x_i$ TRUE, ZAC - ZIG if $\overline{x_i}$ TRUE. DETOUR ON CLAUSES NOT ALREADY COVERED.
UHAMPATH = \{ (G,s,t) \mid G \text{ is an undirected graph with a Hamilton path from } s \text{ to } t \}\]

**Theorem:** UHAMPATH is NP-Complete

1. UHAMPATH ∈ NP
2. HAMPATH \leq_p UHAMPATH
HAMPATH $\leq_p$ UHAMPATH

Rule: $u \rightarrow v$ then $u_{out}$

Example:

Why do we need mid?
SUBSETSUM = \{ (S, t) \mid S \text{ is multiset of integers and for some } Y \subseteq S, \text{ we have } \sum_{y \in Y} y = t \}

**Theorem:** SUBSETSUM is NP-Complete

1. SUBSETSUM ∈ NP
2. 3SAT ≤ₚ SUBSETSUM
\[ \phi = \bigwedge_{j=1}^{C} \bigvee_{i=1}^{k} \neg x_{j} \]  

\text{VARIABLES: } x_{1}, \ldots, x_{k} 

\((S, t)\)  
\[ S = \left\{ y_{i}, z_{i}, g_{i}, h_{i} \mid i = 1, \ldots, k \right\} \]  
\[ t = \left\{ \frac{1}{2}, \frac{3}{2}, \frac{3}{k} \right\} \]

\[ 1, 2, \ldots, 2, 4, 2, 3, \ldots, k \]

\[ \begin{array}{cccc}
\neg x_{1} & y_{1} & 1 & 0 \ldots 0 \vspace{0.2cm} \\
\neg x_{1} & z_{1} & 1 & 0 \ldots 0 \vspace{0.2cm} \\
x_{2} & \neg y_{2} & 0 & 1 \ldots 0 \vspace{0.2cm} \\
x_{2} & \neg z_{2} & 0 & 1 \ldots 0 \vspace{0.2cm} \\
x_{k} & \neg y_{k} & 0 & 1 \ldots 0 \vspace{0.2cm} \\
x_{k} & \neg z_{k} & 0 & 1 \ldots 0 \vspace{0.2cm} \\
\end{array} \]

\[ \neg x_{1} \subset 0 \text{ (other)} \]

\[ \neg x_{1} \subset 0 \text{ (other)} \]

\[ \{ g_{1} \mid h_{1} \} \subset 10 \ldots 0 \]

\[ \{ g_{2} \mid h_{2} \} \subset 10 \ldots 0 \]

\[ \vdots \]

\[ \{ g_{k} \mid h_{k} \} \subset 10 \ldots 0 \]

\[ t = \left\{ \frac{1}{1}, \frac{3}{1}, \frac{3}{k} \right\} \]

\( \phi \) satisfiable with some truth assignment. For subset choose rows with literals true & \( g_{j}'s \) & \( h_{j}'s \) as necessary to add up.
Let $G$ denote a graph, and $s$ and $t$ denote nodes.

**SHORTEST PATH**

$$= \{(G, s, t, k) \mid G \text{ has a simple path of length } < k \text{ from } s \text{ to } t \}$$

**LONGEST PATH**

$$= \{(G, s, t, k) \mid G \text{ has a simple path of length } > k \text{ from } s \text{ to } t \}$$

**WHICH IS EASY? WHICH IS HARD?** Justify (see Sipser 7.21)
Good Luck on Midterm 2!