15-453

FORMAL LANGUAGES, AUTOMATA AND COMPUTABILITY

THURSDAY APRIL 3

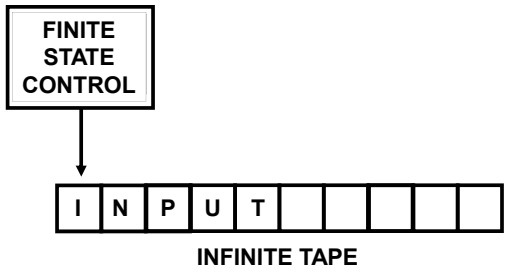
REVIEW for Midterm 2

TUESDAY April 8

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

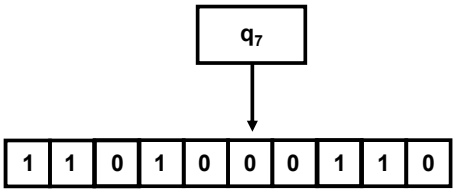
- **Q is a finite set of states**
- **Σ is the input alphabet, where $\square \notin \Sigma$**
- **Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$**
- **$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$**
- **$q_0 \in Q$ is the start state**
- **$q_{accept}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

TURING MACHINE



CONFIGURATIONS

$11010q_700110$



COMPUTATION HISTORIES

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

1. **C_1 is the start configuration, $C_1 = q_0 w$**
2. **C_k is an accepting configuration, $C_k = u q_{accept} v$**
3. **Each C_i follows from C_{i-1}** via the transition function δ

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

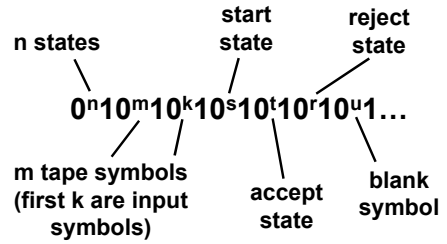
1. **C_1 is the start configuration,**
2. **C_k is a rejecting configuration, $C_k = u q_{reject} v$**
3. **Each C_i follows from C_{i-1}**

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 Σ^*.

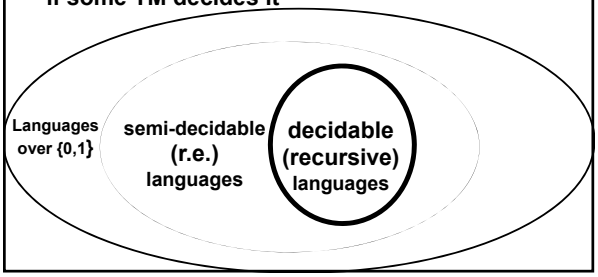
We may assume that any string in Σ^* describes some TM:

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 Turing-recognizable or semi-decidable or recursively enumerable (r.e.) if some TM recognizes it

A language is called decidable or recursive if some TM decides it



A_{TM} = { (M,w) | M is a TM that accepts string w }

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

Assume machine H decides A_{TM}

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

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

$$D(M) = \begin{cases} \text{Reject} & \text{if M accepts M} \\ & \text{i.e. if H(M,M) accepts} \\ \text{Accept} & \text{if M does not accept M} \\ & \text{i.e. if H(M,M) rejects} \end{cases}$$

A_{TM} = { (M,w) | M is a TM that accepts string w }

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

Assume machine H decides A_{TM}

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

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

$$D(D) = \begin{cases} \text{Reject} & \text{if D accepts D} \\ \text{Accept} & \text{if D does not accept D} \end{cases}$$

A_{TM} = { (M,w) | M is a TM that accepts string w }

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

Assume machine H SEMI-DECIDES A_{TM}

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

Construct a new TM D_H as follows:
on input M, run H on (M,M) and output the "opposite" of H *whenever possible*.

$$D_H(D_H) = \begin{cases} \text{Reject if } D_H \text{ accepts } D_H \\ \text{(i.e. if H}(D_H, D_H) = \text{Accept)} \\ \text{Accept if } D_H \text{ reiect } D_H \text{ l} \\ \text{(i.e. if H}(D_H, D_H) = \text{Reject)} \\ \text{Loops i } D_H \text{ loops o } D_H \\ \text{(i.e. if H}(D_H, D_H) \text{ loops)} \end{cases}$$

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.

THE RECURSION THEOREM

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

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

Given T: (a,b) → [T] → t(a,b)

THEN ∃ R: w → [R] → t(R,w)

Recursion Theorem: A Turing machine can obtain its own description, 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)

A NOTE ON SELF REFERENCE

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

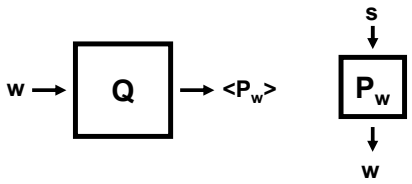
Print this sentence.

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

"Print two copies of the following (the stuff inside quotes), and put the second copy in quotes:" = $P_{<B>}$

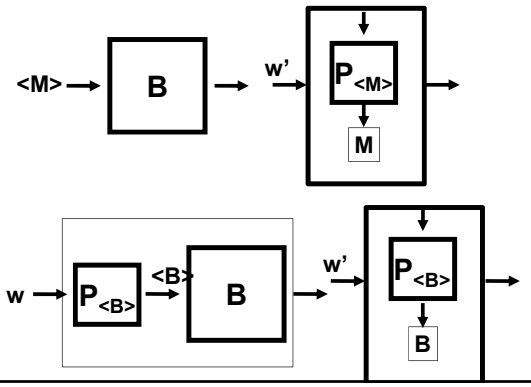
A TM THAT PRINTS <itSELF>

Theorem: There is a computable function $q : \Sigma^* \to \Sigma^*$, where for any string w, q(w) is the *description* of a TM P_w that on any input, prints out w and then accepts



TM Q computes q

A TM THAT PRINTS <itSELF>



Theorem: A_{TM} is undecidable

Proof (using the recursion theorem): Informally

Assume to the contrary, D decides A_{TM}

Construct machine R such that
On input w:

1. Obtains, via the recursion theorem, its own description R

2. Runs D on (R,w) and flips the output

Running R on input w does the opposite of what D says it should!

Theorem: A_{TM} is undecidable

Proof (using the recursion theorem) Formally

1. Assume to the contrary that D decides A_{TM}

2. Let T(M, w) = opposite of D(M, w)

3. Obtain by the Recursion Theorem, machine R so that R(w) = T(R, w)

Then,

R(w) accepts ⇔ T(R, w) accepts ⇔ D(R, w) rejects
by 3 by 2
⇔ R(w) does not accept
by 1

MIN_{TM} = {<M>| M is a minimal TM, wrt |<M>|}

Theorem: MIN_{TM} is not RE.

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

Assume E enumerates MIN_{TM}
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} = {<M>| M is a minimal TM, wrt |<M>|}

Theorem: MIN_{TM} is not RE.

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

Assume E enumerates MIN_{TM}

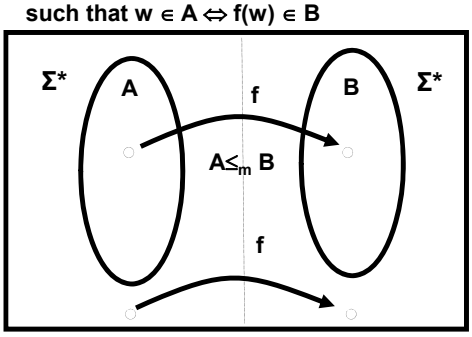
Let $T_E(x, w)$ = D(w) where <D> is first in E's enumeration s.t. |<D>| > |x|

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

R(w) = T_E(<R>, w) = D(w) where <D> is first in E's enumeration s.t. |<D>| > |<R>

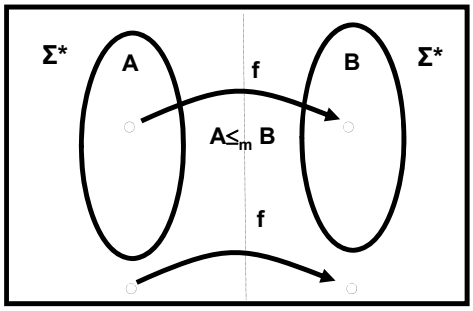
Contradiction. Why?

Let f : $\Sigma^* \to \Sigma^*$ be a computable function such that $w \in A \Leftrightarrow f(w) \in B$



Say: A is Mapping Reducible to B
Write: $A \leq_m B$ **(also, $\neg A \leq_m \neg B$ (why?))**

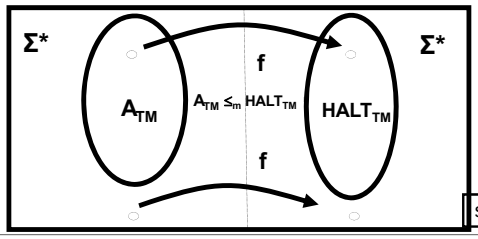
Let $f : \Sigma^* \to \Sigma^*$ be a computable function such that $w \in A \Leftrightarrow f(w) \in B$



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

$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

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

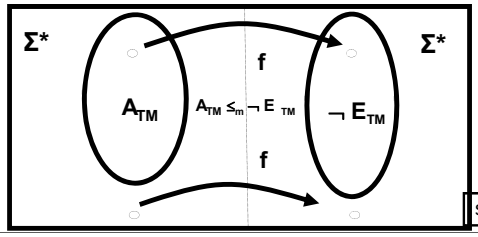


f: (M,w) → (M', w) where **M'(s) = M(s)** if **M(s)** accepts,
Loops otherwise

So, (M, w)∈ A_{TM} ⇔ (M', w) ∈ $HALT_{TM}$

$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

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

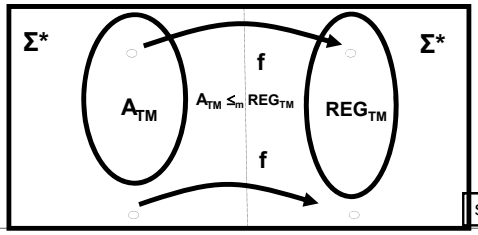


f: (M,w) → M_w where M_w (s) = M(w) if **s = w,**
Loops otherwise

So, (M, w)∈ A_{TM} ⇔ $M_w \in \neg E_{TM}$

$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

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

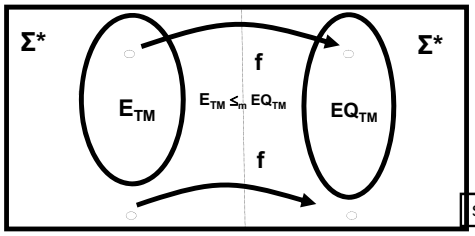


f: (M,w) → M'$_w$ where M'$_w$ (s) = **accept** if **s = 0^n1^n,**
M(w) otherwise

So, (M, w)∈ A_{TM} ⇔ M'$_w$ ∈ REG_{TM}

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

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

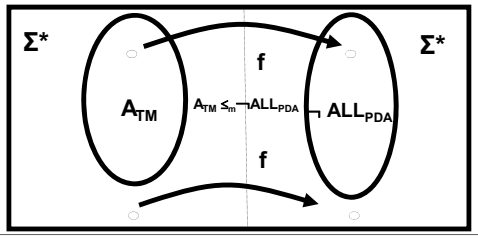


f: M → (M, $M_\varnothing$) where $M_\varnothing$ (s) = **Loops**

So, M∈ E_{TM} ⇔ (M, $M_\varnothing$) ∈ EQ_{TM}

$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

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



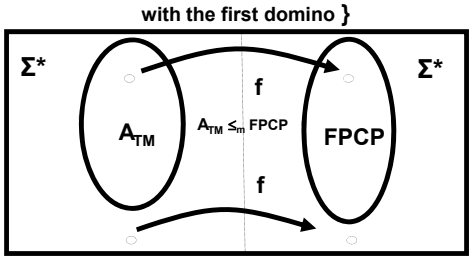
f: (M,w) → PDA P_w where s ∈ Σ*

P_w (s) = **accept** iff **s** is **NOT** an accepting computation of **M(w)**

So, (M, w)∈ A_{TM} ⇔ $P_w \in \neg ALL_{PDA}$

A_{TM} = { (M,w) | M is a TM that accepts string w }

FPCP = { P | P is a set of dominos with a match starting with the first domino }

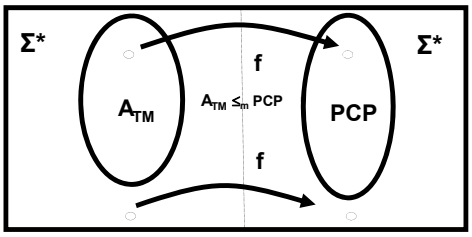


Construct f: (M,w) → $P_{(M,w)}$ such that

(M, w) ∈ A_{TM} ⇔ $P_{(M,w)}$ ∈ FPCP

A_{TM} = { (M,w) | M is a TM that accepts string w }

PCP = { P | P is a set of dominos with a match }



Construct f: (M,w) → $P_{(M,w)}$ such that

(M, w) ∈ A_{TM} ⇔ $P_{(M,w)}$ ∈ PCP

A_{TM} = { (M,w) | M is a TM that accepts string w }

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

E_{TM} = { M | M is a TM and L(M) = ∅ }

REG_{TM} = { M | M is a TM and L(M) is regular}

EQ_{TM} = {(M, N) | M, N are TMs and L(M) =L(N)}

ALL_{PDA} = { P | P is a PDA and L(P) = Σ* }

PCP = { P | P is a set of dominos with a match }

ALL UNDECIDABLE

Use Reductions to Prove

A_{TM} = { (M,w) | M is a TM that accepts string w }

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

E_{TM} = { M | M is a TM and L(M) = ∅ } $\neg E_{TM}$

REG_{TM} = { M | M is a TM and L(M) is regular}

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

ALL_{PDA} = { P | P is a PDA and L(P) = Σ* } $\neg ALL_{PDA}$

PCP = { P | P 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

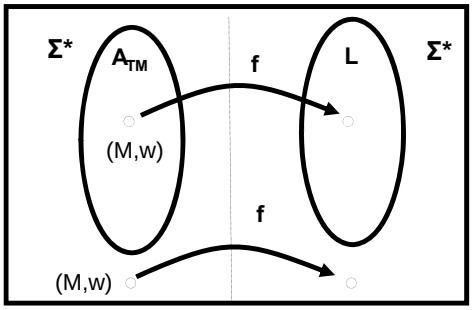
FIN_{TM} = { M | M is a TM and L(M) is finite}

E_{TM} = { M | M is a TM and L(M) = ∅ }

REG_{TM} = { M | M is a TM and L(M) is regular}

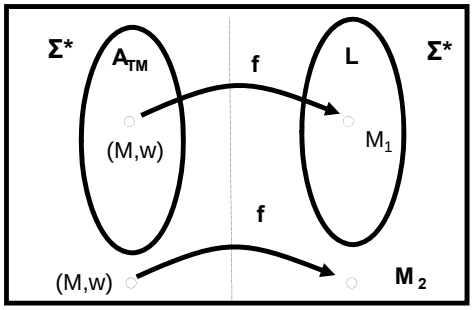
Proof: Show L is undecidable

Show: A_{TM} is mapping reducible to L



Proof: Show L is undecidable

Show: A_{TM} is mapping reducible to L



RICE'S THEOREM

Proof:

Define $M_\varnothing$ to be a TM that never halts

Assume, WLOG, that $M_\varnothing \notin L$ Why?

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

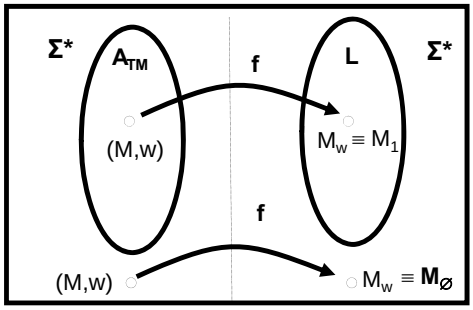
Show A_{TM} is mapping reducible to L:

Map (M, w) → M_w where

M_w (s) = accepts if both M(w) and M_1(s) accept
loops otherwise

What is the language of M_w ?

A_{TM} is mapping reducible to L



QED

Corollary: The following languages are undecidable.

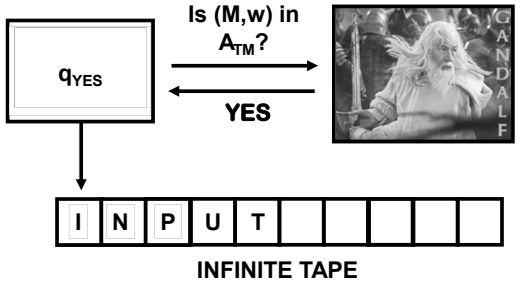
E_{TM} = { M | M is a TM and L(M) = ∅ }

REG_{TM} = {M | M is TM and L(M) is regular}

FIN_{TM} = {M | M is a TM and L(M) is finite}

DEC_{TM} = {M | M is a TM and L(M) is decidable}

ORACLE TMs



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

$\leq_T$ VERSUS $\leq_m$

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 $f : \Sigma^* \to \Sigma^*$, where for every w,

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

We can thus use an oracle for B to decide A

Theorem: $\neg HALT_{TM} \leq_T HALT_{TM}$

Theorem: $\neg HALT_{TM} \not\leq_m HALT_{TM}$ **WHY?**

THE ARITHMETIC HIERARCHY

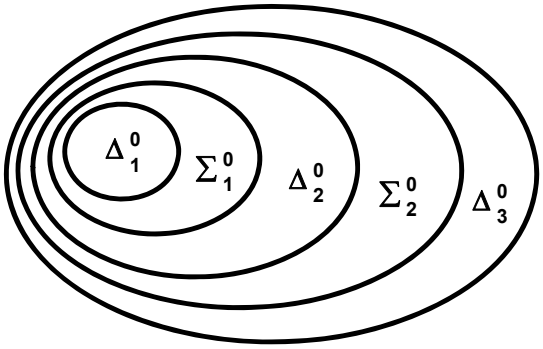
Δ^0_1 = { decidable sets } (sets = languages)

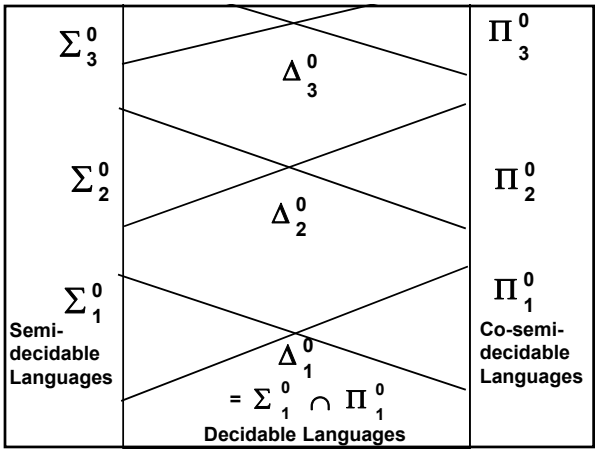
Σ^0_1 = { semi-decidable sets }

Σ^0_{n+1} = { sets semi-decidable in some $B \in \Sigma^0_n$ }

Δ^0_{n+1} = { sets decidable in some $B \in \Sigma^0_n$ }

Π^0_n = { complements of sets in Σ^0_n }





Definition: A decidable predicate R(x,y) is some proposition about x and y[1], 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) = "x + y is less than 100"

R(<N>,y) = "N halts on y 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 Σ^*

Theorem: A language A is semi-decidable if and only if there is a decidable predicate R(x, y) such that A = { x | ∃y R(x,y) }

Proof:

(1) If A = { x | ∃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 | ∃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

Σ^0_1 **= { semi-decidable sets }**

= languages of the form { x | ∃y R(x,y) }

Π^0_1 **= { complements of semi-decidable sets }**

= languages of the form { x | ∀y R(x,y) }

Δ^0_1 **= { decidable sets }**

$= \Sigma^0_1 \cap \Pi^0_1$

Where R is a decidable predicate

Theorem

Σ^0_2 **= { sets semi-decidable in some semi-dec. B }**

= languages of the form $\{ x \mid \exists y_1 \forall y_2\, R(x,y_1,y_2) \}$

Π^0_2 **= { complements of** Σ^0_2 **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

Σ^0_n **= languages** $\{ x \mid \exists y_1 \forall y_2 \exists y_3 \ldots Qy_n\, R(x,y_1,\ldots,y_n) \}$

Π^0_n **= languages** $\{ x \mid \forall y_1 \exists y_2 \forall y_3 \ldots Qy_n\, R(x,y_1,\ldots,y_n) \}$

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

Where R is a decidable predicate

Example

Decidable predicate

Σ^0_1 **= languages of the form** $\{ x \mid \exists y\, R(x,y) \}$

We know that A_{TM} **is in** Σ^0_1 Why?

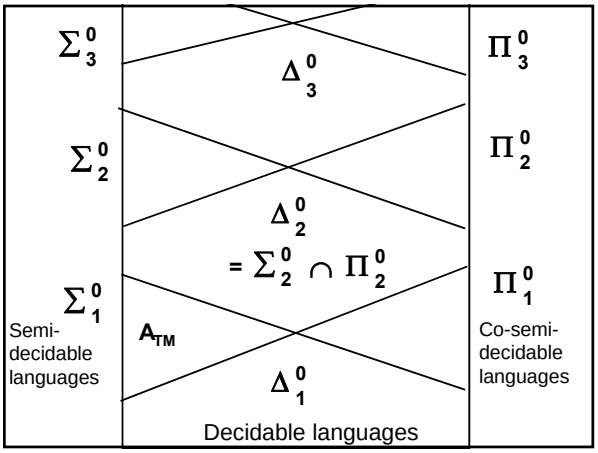
Show it can be described in this form:

A_{TM} **= { <(M,w)> | ∃t [M accepts w in t steps] }**

decidable predicate

A_{TM} **= { <(M,w)> | ∃t T (<M>, w, t)}**

A_{TM} **= { <(M,w)> | ∃v (v is an accepting computation history of M on w}**



Example

Π^0_1 = languages of the form { x | ∀y R(x,y) }

Show that EMPTY (ie, E_{TM}) = { M | L(M) = ∅ } is in Π^0_1

EMPTY = { M | ∀w∀t [M doesn't accept w in t steps] }

two quantifiers?? decidable predicate

Example

Π^0_1 = languages of the form { x | ∀y R(x,y) }

Show that EMPTY (ie, E_{TM}) = { M | L(M) = ∅ } is in Π^0_1

EMPTY = { M | ∀w∀t [¬T(<M>, w, t)] }

two quantifiers?? decidable predicate

THE PAIRING FUNCTION

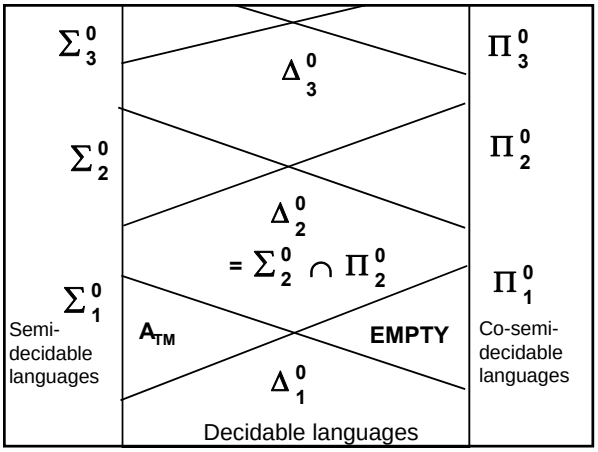
Theorem. There is a 1-1 and onto computable function < , >: $\Sigma^* \times \Sigma^* \to \Sigma^*$ and computable functions π_1 and $\pi_2 : \Sigma^* \to \Sigma^*$ such that

$z = \langle w, t \rangle \Rightarrow \pi_1(z) = w$ and $\pi_2(z) = t$

EMPTY = { M | ∀w∀t [M doesn't accept w in t steps] }

EMPTY = { M | ∀z[M doesn't accept π_1 (z) in π_2(z) steps]}

EMPTY = { M | ∀z[¬T(<M>, π_1 (z) , π_2(z))] }



Example

Π^0_2 = languages of the form { x | ∀y∃z R(x,y,z) }

Show that TOTAL = { M | M halts on all inputs } is in Π^0_2

TOTAL = { M | ∀w ∃t [M halts on w in t steps] }

decidable predicate

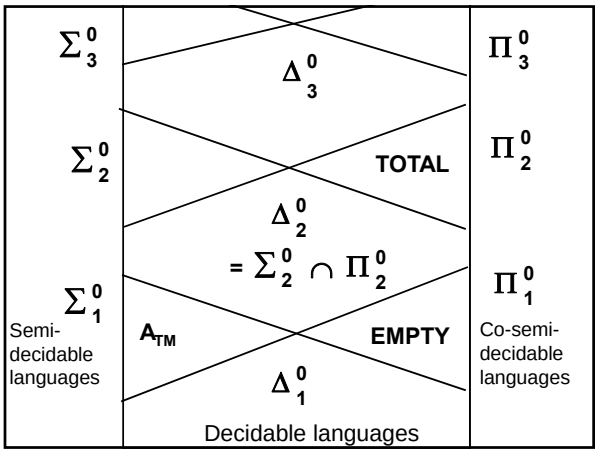
Example

Π^0_2 = languages of the form { x | ∀y∃z R(x,y,z) }

Show that TOTAL = { M | M halts on all inputs } is in Π^0_2

TOTAL = { M | ∀w ∃t [T(<M>, w, t)] }

decidable predicate



Example

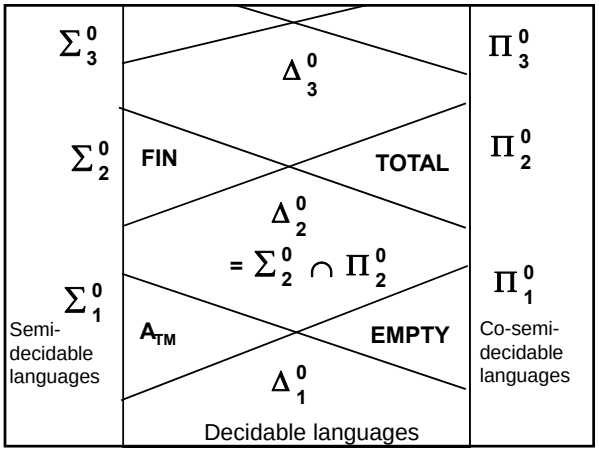
Σ^0_2 = languages of the form { x | ∃y∀z R(x,y,z) }

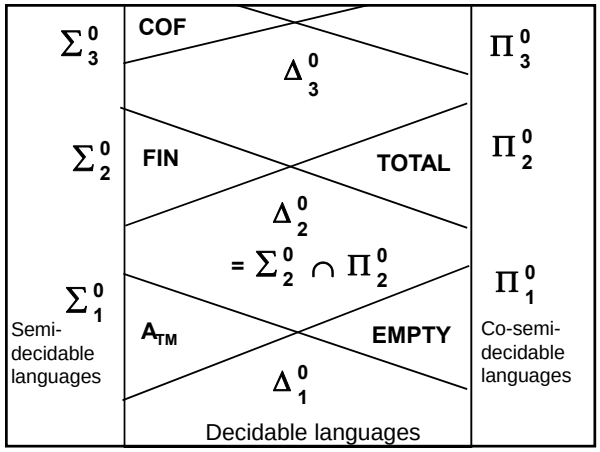
Show that FIN = { M | L(M) is finite } is in Σ^0_2

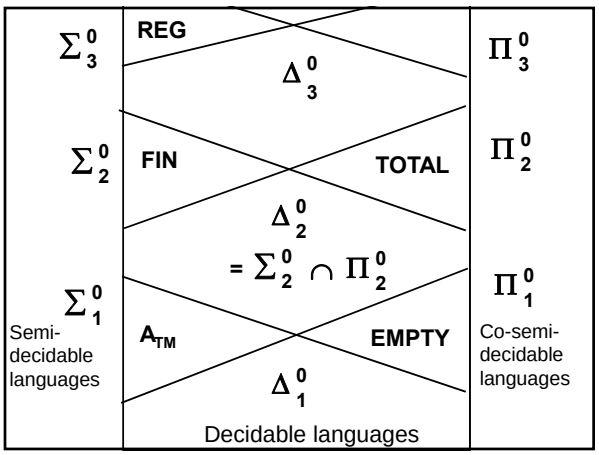
FIN = { M | ∃n∀w∀t [Either |w| < n, or
M doesn't accept w in t steps] }

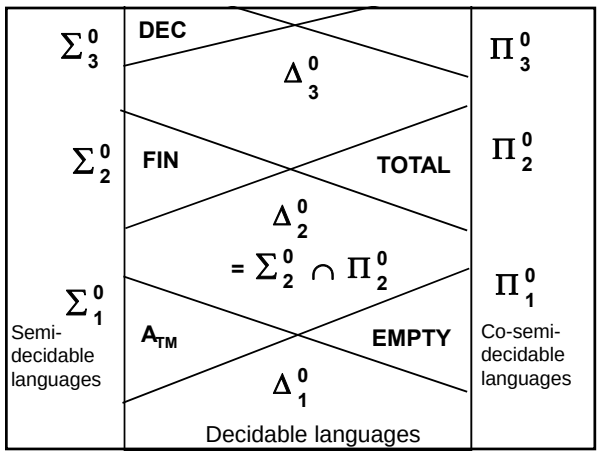
FIN = { M | ∃n∀w∀t (|w| < n ∨¬ T(<M>,w, t))}

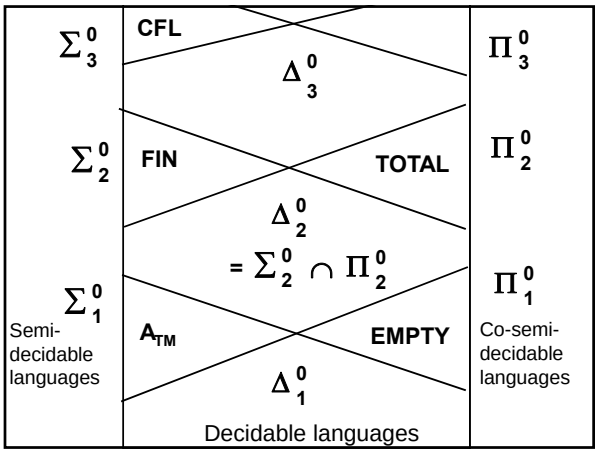
decidable predicate











Each is m-complete for its level in hierachy 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:**

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

Can use diagonalization here!

Suppose H decides SUPERHALT (with oracle)

Define **D(X) = "if H(X,X) accepts (with oracle) then LOOP, else ACCEPT."**

D(D) halts $\Leftrightarrow$ H(D,D) accepts $\Leftrightarrow$ D(D) loops…

ORACLES not all powerful

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.

$SUPERHALT^0$ = HALT = { (M,x) | M halts on x}.

$SUPERHALT^1$ = { (M,x) | M, with an oracle for the Halting Problem, halts on x}

$SUPERHALT^n$ = { (M,x) | M, with an oracle for $SUPERHALT^{n-1}$, halts on x}

Theorem:

1. **The hierarchy is strict**
2. **Each of the languages is m-complete for its class.**

Proof Idea.

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

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

Then $A_{TM, n} \in \Sigma^0_n - \Pi^0_n$

Theorem:

1. **The hierarchy is strict**
2. **Each of the languages is m-complete for its class.**

Proof.

2. Eg to show FIN is m-complete for Σ^0_2

Need to show

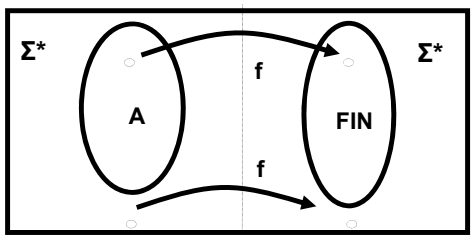
a) FIN $\in \Sigma^0_2$

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

b) For A $\in \Sigma^0_2$ then **$A \leq_m$ FIN**

For $A \in \Sigma_2^0$, $A=\{ x \mid \exists y \forall z\ R(x,y,z)\}$

FIN = { M | L(M) is finite }

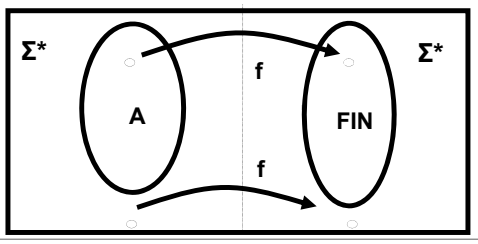


$f: x \rightarrow M_x$

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 $A \in \Sigma_2^0$, $A=\{ x \mid \exists y \forall z\ R(x,y,z)\}$

FIN = { M | L(M) 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 $M_x \in$ **FIN**.
•If $x \notin A$, **then $\forall y \exists z \neg$ R(x,y,z)}** , so M_x recognizes **Σ***

CAN WE QUANTIFY HOW MUCH INFORMATION **IS IN A STRING?**

A = 01010101010101010101010101010101

B = 11001001110111010110100101100101 1

Idea: The more we can "compress" a string, the less "information" it contains….

KOLMOGOROV COMPLEXITY

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)

REPETITIVE STRINGS

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
$K(xx) \leq |<N,<M,w'>>| \leq 2|N| + K(x) + 1 \leq c + K(x)$

REPETITIVE STRINGS

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 $K(x^n)$.

REPETITIVE STRINGS

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 $w = (01)^n$, $K(w) \leq K(01) + c \log_2 n$

CONCATENATION of STRINGS

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

(Number of descriptions of length < n)
$\leq$ **(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**,

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

"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 = $\{(x,c) \mid K(x) \leq c\}$

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) | K(x) ≤ n}

Theorem: COMPRESS is undecidable!

Proof:
M = "On input x ∈ {0,1}*,
Interpret x as integer n. (|x| ≤ log n)
Find first y ∈ {0,1}* in lexicographical order, s.t. (y,n) ∉ COMPRESS, then print y and halt."
M(x) prints the first string y* with K(y*) > n.
Thus <M,x> describes y*, and |<M,x>| ≤ c + log n
So n < K(y*) ≤ c + log n. CONTRADICTION!

DETERMINING COMPRESSIBILITY

Theorem: K is not computable

Proof:
M = "On input x ∈ {0,1}*,
Interpret x as integer n. (|x| ≤ log n)
Find first y ∈ {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>| ≤ c + log n
So n < K(y*) ≤ c + log n. CONTRADICTION!

DETERMINING COMPRESSIBILITY

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.
$$n = p_1^{e1} \ldots p_k^{ek}$$
For all i, $e_i \le \log n$, so $|e_i| \le \log \log n$
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 : N → N, where f(n) is the maximum number of steps that M uses on any input of length n.

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

$$P = \bigcup_{k \in N} TIME(n^k)$$

Definition: A Non-Deterministic TM is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

Q is a finite set of states

Σ is the input alphabet, where □ ∉ Σ

Γ is the tape alphabet, where □ ∈ Γ and Σ ⊆ Γ

$\delta : Q \times \Gamma \to 2^{(Q \times \Gamma \times \{L,R\})}$

$q_0 \in Q$ is the start state

$q_{accept} \in Q$ is the accept state

$q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$

Definition: NTIME(t(n)) = { L | L is decided by a O(t(n))-time non-deterministic Turing machine }

TIME(t(n)) $\subseteq$ NTIME(t(n))

$$NP = \bigcup_{k \in N} NTIME(n^k)$$

Theorem: L $\in$ NP $\Leftrightarrow$ if there exists a poly-time Turing machine V with

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

Proof:

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

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

(2) If L $\in$ NP then

L = { x | $\exists$y |y| = poly(|x|) and V(x,y) 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:

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

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 = { ϕ | ($\exists$y)[y is a satisfying assignment to ϕ and ϕ is a boolean formula] }

3SAT = { ϕ | ($\exists$y)[y is a satisfying assignment to ϕ and ϕ is in 3cnf] }

Theorem (Cook-Levin):
SAT and 3-SAT are NP-complete

1. SAT $\in$ 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 $\in$ P if and only if P = NP

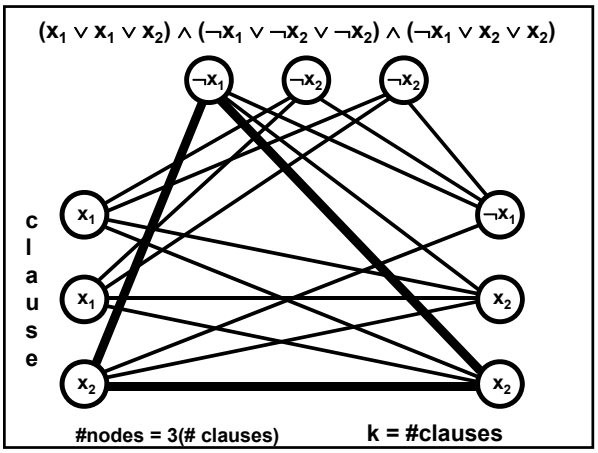
Assume a reasonable encoding of graphs
(example: the adjacency matrix is reasonable)

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

Theorem: CLIQUE is NP-Complete

(1) CLIQUE $\in$ NP

(2) 3SAT $\leq_p$ CLIQUE

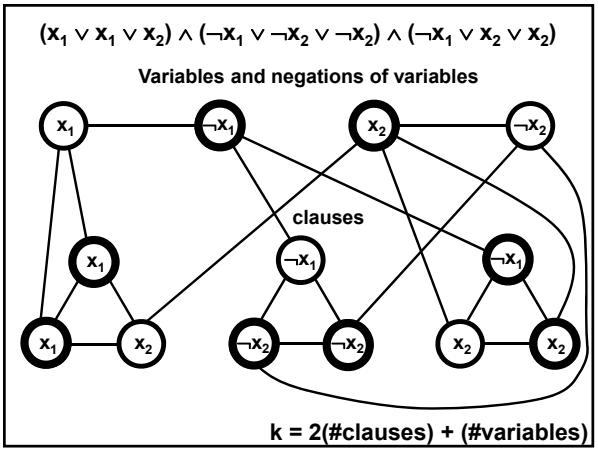


VERTEX-COVER = { (G,k) | G is an undirected graph with a k-node vertex cover }

Theorem: VERTEX-COVER is NP-Complete

(1) VERTEX-COVER $\in$ NP

(2) 3SAT $\leq_p$ VERTEX-COVER



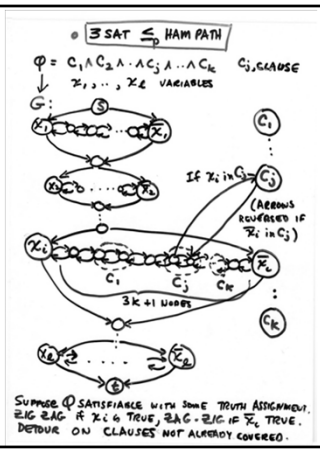
HAMPATH = { (G,s,t) | G is an directed graph with a Hamilton path from s to t}

Theorem: HAMPATH is NP-Complete

(1) HAMPATH $\in$ NP

(2) 3SAT $\leq_p$ HAMPATH

Proof is in Sipser, Chapter 7.5

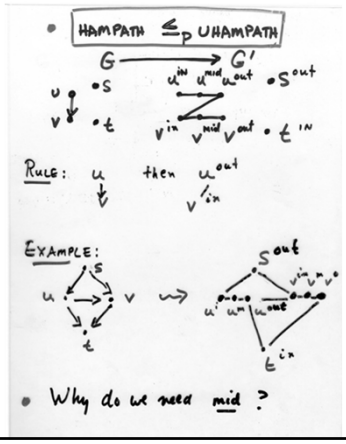


UHAMPATH = { (G,s,t) | G is an undirected graph with a Hamilton path from s to t}

Theorem: UHAMPATH is NP-Complete

(1) UHAMPATH ∈ NP

(2) HAMPATH $\leq_p$ UHAMPATH

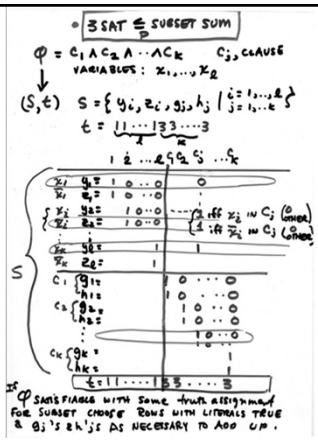


SUBSETSUM = { (S, t) | S is multiset of integers and for some Y ⊆ S, we have $\sum_{y \in Y} y = t$ }

Theorem: SUBSETSUM is NP-Complete

(1) SUBSETSUM ∈ NP

(2) 3SAT $\leq_p$ SUBSETSUM



HW

Let G denote a graph, and s and t denote nodes.

SHORTEST PATH
= {(G, s, t, k) |
G has a simple path of length < k from s to t }

LONGEST PATH
= {(G, s, t, k) |
G has a simple path of length > k from s to t }

WHICH IS EASY? WHICH IS HARD? Justify (see Sipser 7.21)

WWW.FLAC.WS

Good Luck on Midterm 2!