CS 154

Reductions and Undecidability

A Concrete Undecidable Problem

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

Theorem: A_{TM} is recognizable but **NOT** decidable

Corollary: $\neg A_{TM}$ is not recognizable

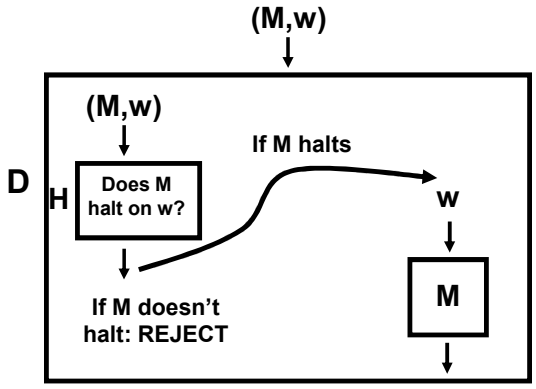
The Halting Problem

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

Theorem: $HALT_{TM}$ is undecidable

Proof: Assume, for a contradiction, that TM H decides $HALT_{TM}$

We use H to construct a TM D that decides A_{TM}
But A_{TM} is undecidable – contradiction.



One can often show that a language L is undecidable by showing that if L is decidable, then so is A_{TM}

We reduce A_{TM} to the language L

$$A_{TM} \leq L$$

We showed: $A_{TM} \leq HALT_{TM}$

Mapping Reductions

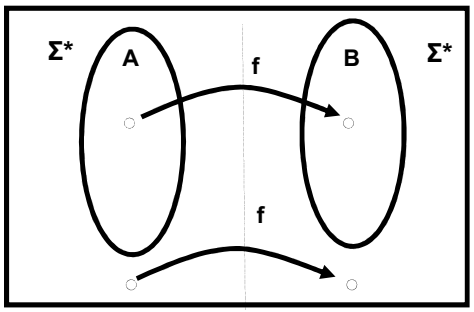
$f : \Sigma^* \rightarrow \Sigma^*$ is a computable function if there is a Turing machine M that halts with just f(w) written on its tape, for every input w.

A language A is *mapping reducible* to language B, written $A \leq_m B$, if there is a computable $f : \Sigma^* \rightarrow \Sigma^*$, such that for every w,

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

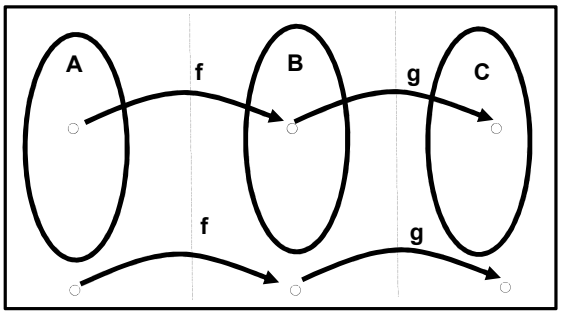
f is called a mapping reduction (or many-one reduction) from A to B

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: $A \leq_m B$

Theorem: If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$



Theorem: If $A \leq_m B$ and B is decidable, then A is decidable

Proof: Let M decide B.
Let f be a reduction from A to B

To decide A, we build a machine M'

M'(w):

1. Compute f(w)
2. Run M on f(w), output its answer

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

Theorem: If $A \leq_m B$ and B is recognizable, then A is recognizable

Proof: Let M decide B.
Let f be a reduction from A to B

To decide A, we build a machine M'

M'(w):

1. Compute f(w)
2. Run M on f(w), output its answer if you ever receive one

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

Theorem: If $A \leq_m B$ and B is decidable, then A is decidable

Corollary: If $A \leq_m B$ and A is undecidable, then B is undecidable

Theorem: If $A \leq_m B$ and B is recognizable, then A is recognizable

Corollary: If $A \leq_m B$ and A is unrecognizable, then B is unrecognizable

All undecidability proofs we've seen can be viewed as constructing an f that reduces A_{TM} to some language

Theorem: $A_{TM} \leq_m HALT_{TM}$

f(M, w) := (M', w)
where M'(w) = accept, if M(w) accepts
loops otherwise

So $(M, w) \in A_{TM} \Leftrightarrow (M', w) \in HALT_{TM}$

Is $HALT_{TM} \leq_m A_{TM}$?

Yes.
f(M, w) := (M', w)
where M'(w) = accept, if M(w) ever halts
loops otherwise

$(M, w) \in HALT_{TM} \Leftrightarrow (M', w) \in A_{TM}$

The Emptiness Problem

$EMPTY_{TM}$ = { M | M is a TM such that L(M) = ∅}

Theorem: $EMPTY_{TM}$ is undecidable

Proof: Assume (for a contradiction) there is a TM E that decides $EMPTY_{TM}$
We'll use it to get a decider D for A_{TM}

D(M, w) := Build a TM M' with the behavior:
"M'(x) := if x ≠ w reject else run M(w)"
Run E(M').
If E accepts, *reject.* If rejects, *accept.*

The Emptiness Problem

$EMPTY_{TM}$ = { M | M is a TM such that L(M) = ∅}

Theorem: $EMPTY_{TM}$ is *unrecognizable*

Proof:
Show that $\neg A_{TM} \leq_m EMPTY_{TM}$

f(M, w) := Output a TM M' with the behavior:
"M'(x) := if x ≠ w reject else run M(w)"

$(M, w) \in A_{TM} \Leftrightarrow L(M') \neq \varnothing$
$\Leftrightarrow M' \notin EMPTY_{TM}$
$\Leftrightarrow f(M,w) \notin EMPTY_{TM}$

The Regularity Problem

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

Given a program, is it equivalent to some DFA?

Theorem: $REGULAR_{TM}$ is not recognizable

Proof: We show that $\neg A_{TM} \leq_m REGULAR_{TM}$

f(M,w) := Output a TM M' with the behavior:
"M'(x) := if ($x = 0^n1^n$) then sim M(w)
else reject"

$(M, w) \in A_{TM} \Rightarrow f(M,w) = M'$ accepts $\{0^n1^n\}$
$(M, w) \notin A_{TM} \Rightarrow f(M,w) = M'$ accepts nothing
$(M, w) \notin A_{TM} \Leftrightarrow f(M,w) \in REGULAR_{TM}$

The Equivalence Problem

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

Do two programs compute the same function?

Theorem: EQ_{TM} is unrecognizable

Proof: Reduce $EMPTY_{TM}$ to EQ_{TM}
Let $M_\varnothing$ be some TM with no path from start state to accept state
Define f(M) := $(M, M_\varnothing)$

$M \in EMPTY_{TM} \Leftrightarrow L(M) = L(M_\varnothing) = \varnothing$
$\Leftrightarrow (M', M_\varnothing) \in EQ_{TM}$

Post's Correspondence Problem

The PCP Game, or “Domino Solitaire”

ba	a	b	b
a	ab	bcb	a

aaa	a	a	c
a	c	aa	a

b	a	ca	abc
ca	ab	a	c

a c	ca	ac
b	a	ca

RULE #1

If every top string is longer than the corresponding bottom one, there can’t be a match

caa	acc	b	aab	c
a	a	b	aa	a

RULE #2

If there is a domino with the same string on the top and on the bottom, there is a match

Post's Correspondence Problem

Given a collection of domino types, can we build up a match?

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

Theorem: PCP is *undecidable*!

The FPCP Game

… just like the PCP game, except that a match has to start with the first domino type

Theorem: FPCP is *undecidable*
Use this to show PCP is undecidable

FPCP

aaa	a	a	c
a	c	aa	a

FPCP

ba	a	b	b
a	ab	bcb	a

Theorem: FPCP is undecidable

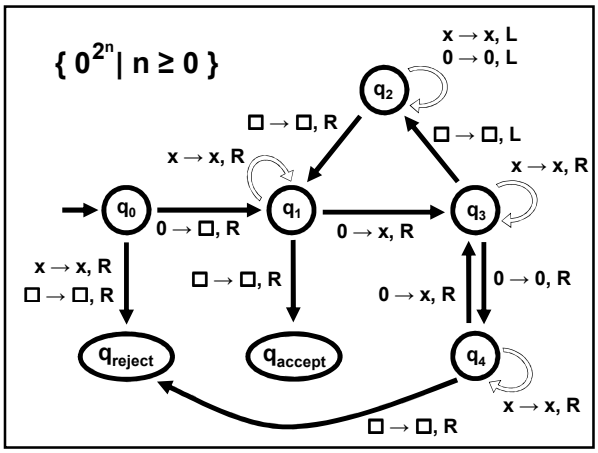
Proof: **We will reduce A_{TM} to FPCP**

Recall: an accepting computation history for M on w is a sequence of configs $C_0, C_1, \ldots, C_k$, where

1. **C_0 is the start configuration $q_0 w$,**
2. **C_k is an accepting configuration,**
3. **Each configuration C_i yields C_{i+1}**

M(w) accepts iff such a history exists.

We'll build an instance of FPCP such that a match encodes an accepting computation history



Given (M, w), we will construct an instance P of FPCP in seven steps

Step 1

Put $\boxed{\dfrac{\#}{\#q_0w_1w_2\ldots w_n\#}}$ into P

The Start Domino

Step 2

If $\delta(q,a) = (p,b,R)$ then add $\boxed{\dfrac{qa}{bp}}$

Step 3

If $\delta(q,a) = (p,b,L)$ then add $\boxed{\dfrac{cqa}{pcb}}$ for all $c \in \Gamma$

The Transition Dominos

Step 4

add $\boxed{\dfrac{a}{a}}$ for all $a \in \Gamma$

Step 5

add $\boxed{\dfrac{\#}{\#}}$ $\boxed{\dfrac{\#}{\square\#}}$

Extra dominoes for filling in the the configurations

Step 4

add $\boxed{\dfrac{a}{a}}$ for all $a \in \Gamma$

Step 5

add $\boxed{\dfrac{\#}{\#}}$ $\boxed{\dfrac{\#}{\square\#}}$

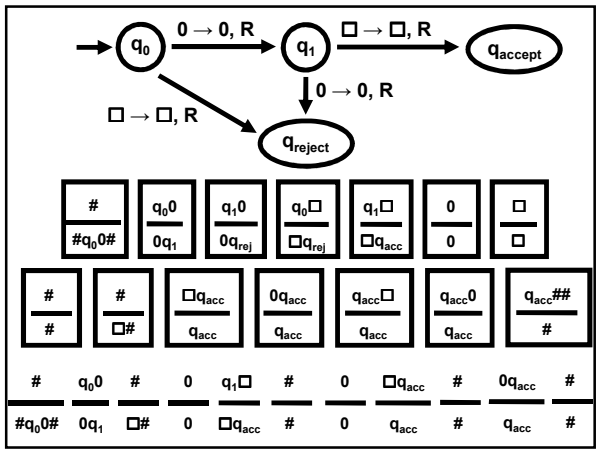
Step 6

add $\boxed{\dfrac{aq_{acc}}{q_{acc}}}$ $\boxed{\dfrac{q_{acc}a}{q_{acc}}}$ for all $a \in \Gamma$ (including □)

Step 7

add $\frac{q_{acc}\#\#}{\#}$

The final domino



Given (M, w), we can effectively construct an instance of FPCP that has a match if and only if M accepts w

Therefore
$A_{TM} \leq_m$ FPCP
and FPCP is undecidable

Can reduce FPCP to PCP:

For $u = u_1 u_2 \ldots u_n$, where $u_i \in \Gamma \cup Q \cup \{\#\}$, define:

$\star u = * u_1 * u_2 * u_3 * \ldots * u_n$

$u \star = u_1 * u_2 * u_3 * \ldots * u_n *$

$\star u \star = * u_1 * u_2 * u_3 * \ldots * u_n *$

FPCP: $\frac{t_1}{b_1}$ $\frac{t_2}{b_2}$ … $\frac{t_k}{b_k}$

PCP: $\frac{\star t_1}{\star b_1 \star}$ $\frac{\star t_1}{b_1 \star}$ $\frac{\star t_2}{b_2 \star}$ … $\frac{\star t_k}{b_k \star}$ $\frac{*\blacklozenge}{\blacklozenge}$

Given (M, w), we can effectively construct an instance of PCP that has a match if and only if M accepts w

Therefore
$A_{TM} \leq_m$ PCP
and PCP is undecidable

Oracle Turing Machines and Hierarchies of Undecidable Problems

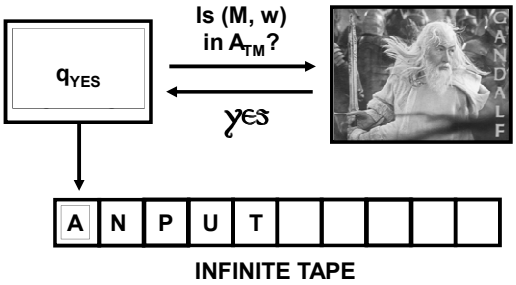
A_{TM} = { (M,w) | M is a TM that accepts string w }

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

$EMPTY_{TM}$ = { M | M is a TM and L(M) is empty }

Are all of these "equally" hard?
How can we even formalize that?

Oracle Turing Machines



Oracle Turing Machines

An oracle is a set B to which the TM may ask membership questions
(formally: TM enters state $q_?$)

and the TM always receives a correct answer in one step
(formally: if the string on the special oracle tape is in B, state $q_?$ is changed to q_{YES}, otherwise q_{NO})

This makes sense even if B is not decidable!

We say A is recognizable in B
if there is an oracle TM M with oracle B
that recognizes A

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

$HALT_{TM}$ is decidable in A_{TM}

On input (M,w), decide if M halts on w as follows:

1. Ask the oracle for A_{TM} if M accepts w.
   If yes, then ACCEPT

2. Switch the accept and reject states of M to get M′. Ask the oracle for A_{TM} if M′ accepts w.
   If yes, then ACCEPT

3. REJECT

A_{TM} is decidable in $HALT_{TM}$

On input (M,w), decide if M accepts w as follows:

Ask the oracle for $HALT_{TM}$ if M halts on w.
If yes, then run M(w) and output its answer.
If no, then REJECT.

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

A "Turing Reduces" to B

$$A \leq_T B$$

$\leq_T$ versus $\leq_m$

Theorem: If $A \leq_m B$ then $A \leq_T B$

Proof:

If $A \leq_m B$ then there is a computable function
$f : \Sigma^* \rightarrow \Sigma^*$, 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?

Limits on Oracle TMs

The following problem cannot be decided 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 still 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 iff H(D,D) accepts iff D(D) loops…

Next Epsiode:

Adventures in Self-Reference

The Arithmetic Hierarchy

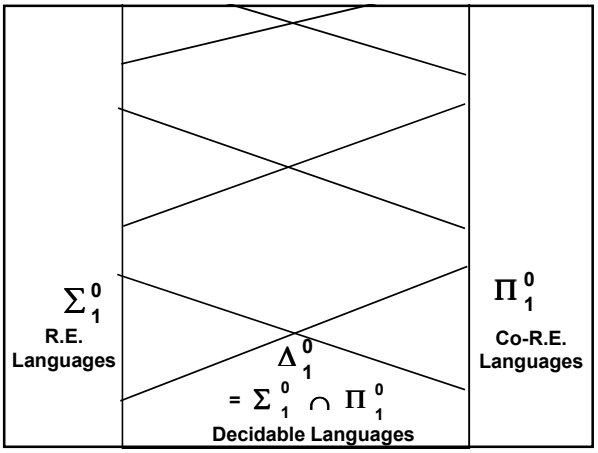
Δ^0_1 = { decidable sets }

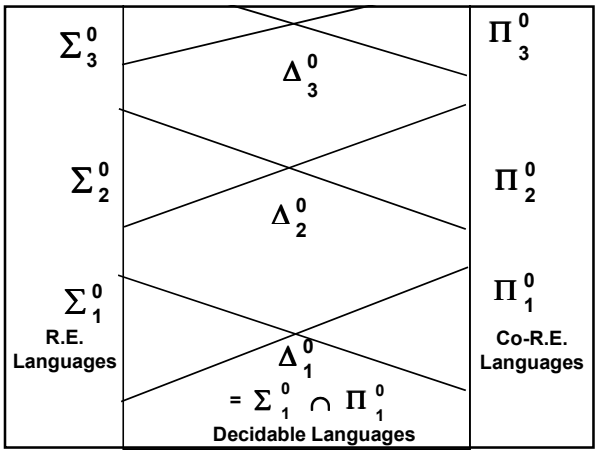
Σ^0_1 = { recognizable sets }

Σ^0_{n+1} = { sets recognizable 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 a proposition about the input strings x and y, such that some TM M implements R. That is,

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"

Note: A is decidable if and only if A = {x | R(x,ε)}, for some decidable predicate R.

Theorem: A language A is recognizable 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 | ∃y R(x,y) } then A is recognizable

A TM can enumerate over all y's and try them

(2) If A is recognizable, then A = { x | ∃y R(x,y) }

Let M recognize A and define R(x,y) to be true iff M accepts x in y steps (here, y is interpreted as an integer)

Σ^0_1 **= { recognizable sets }**
= languages of the form { x | ∃y R(x,y) }

Π^0_1 **= { complements of recognizable 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

Σ^0_2 **= { sets recognizable in some recognizable 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

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

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

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

where R is a decidable predicate

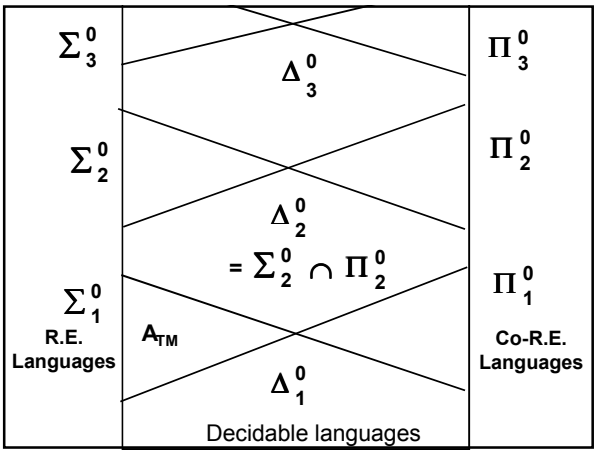
Decidable predicate

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

Theorem: A_{TM} is in Σ^0_1

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

decidable predicate



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

Theorem: EMPTY = { M | L(M) = ∅ } is in Π^0_1

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

two quantifiers??

Pairing Functions

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 = <w, t>
if and only if π_1 (z) = w and π_2(z) = t

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

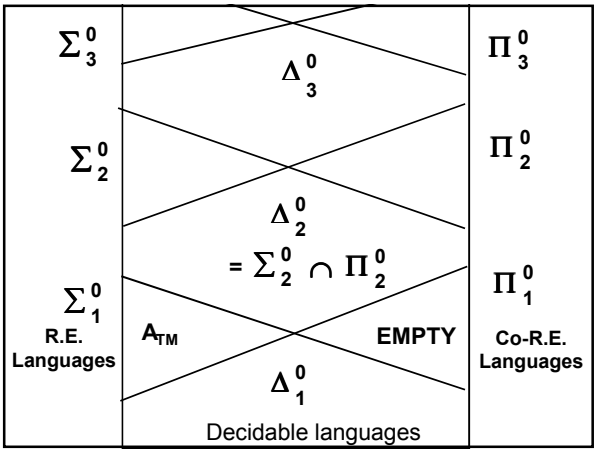
EMPTY = { M | ∀z[M doesn't accept π_1(z) in π_2(z) steps]}

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 = <w, t> iff π_1 (z) = w and π_2(z) = t

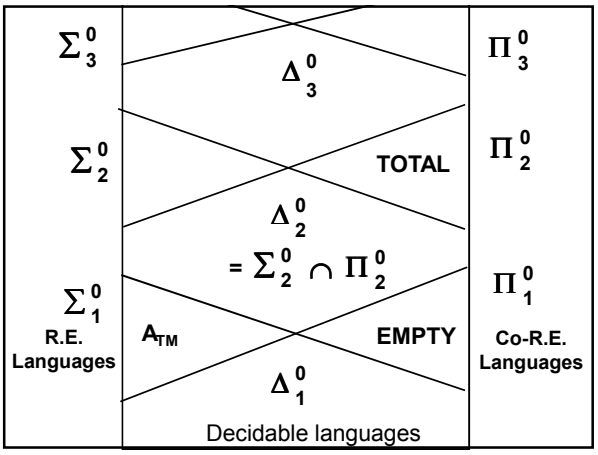
Proof: Let $w = w_1 \ldots w_n \in \Sigma^*$, $t \in \Sigma^*$.
Let $a, b \in \Sigma$, $a \neq b$.
<w, t> := a $w_1 \ldots$ a w_n b t
π_1 (z) := "if z has the form a $w_1 \ldots$ a w_n b t, then output $w_1 \ldots w_n$, else output ε"
π_2(z) := "if z has the form a $w_1 \ldots$ a w_n b t, then output t, else output ε"



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

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

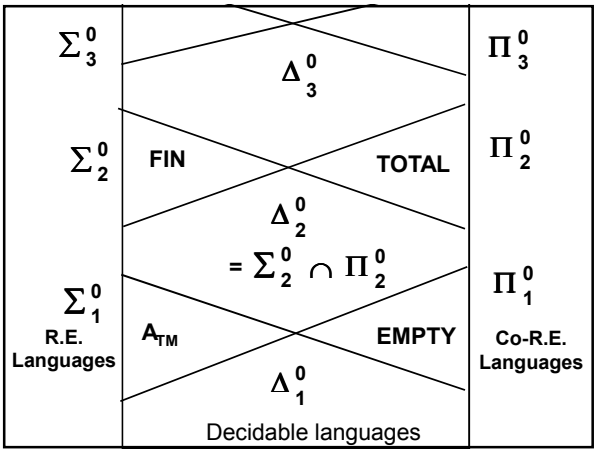
TOTAL = { M | ∀w ∃t [M halts on w in t steps] }



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

Theorem: 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] }



Limits on Oracle TMs

Theorem: The arithmetic hierarchy is strict: the nth level contains a language that isn't in any of the levels below n.

Proof IDEA: Same idea as for SUPERHALT.

$SUPERHALT^0$ = 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}