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 $\Box \notin \Sigma$

Γ is the tape alphabet, where $\Box \in \Gamma$ and $\Sigma \subseteq \Gamma$

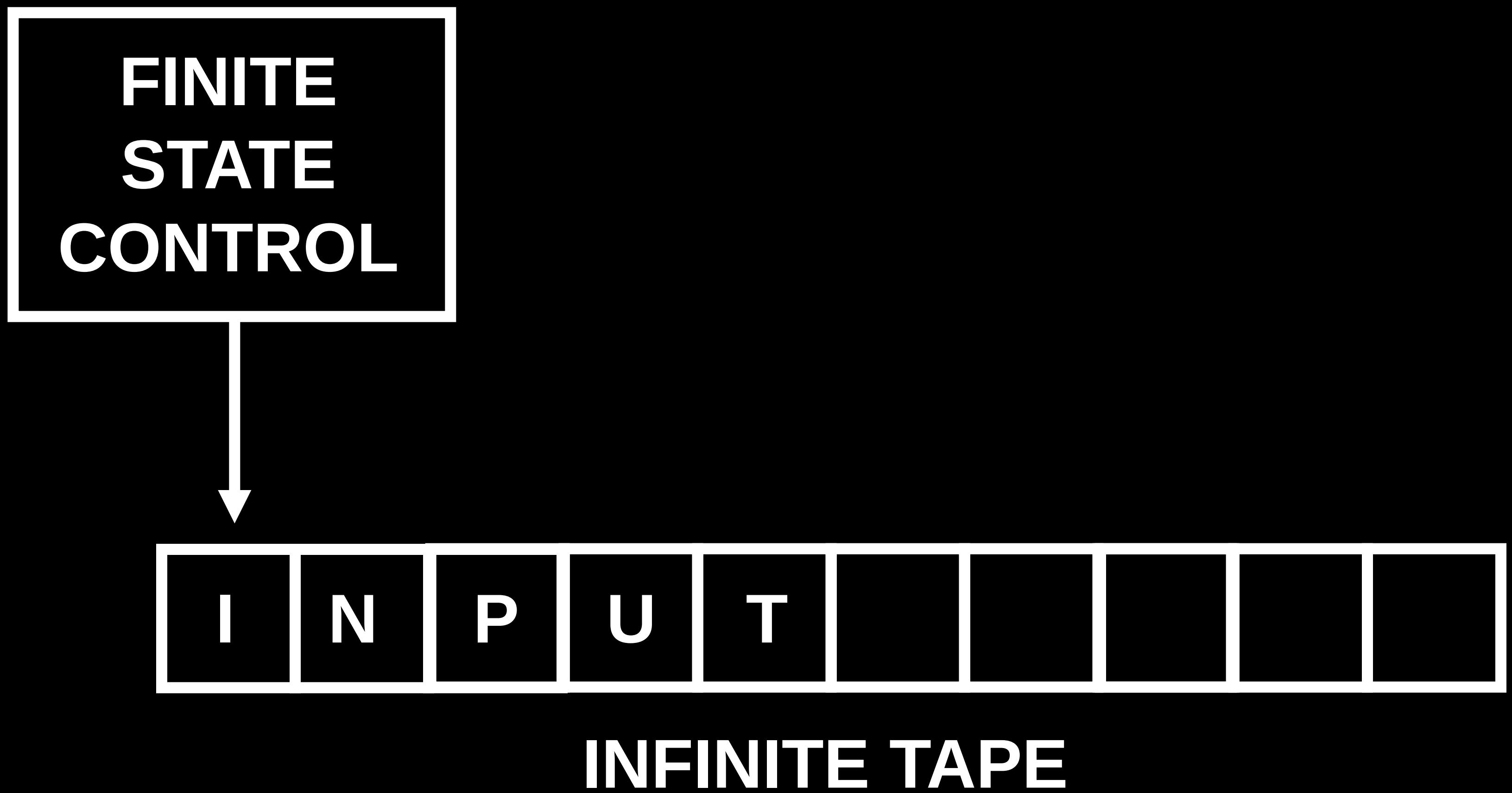
$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

$q_0 \in Q$ is the start state

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

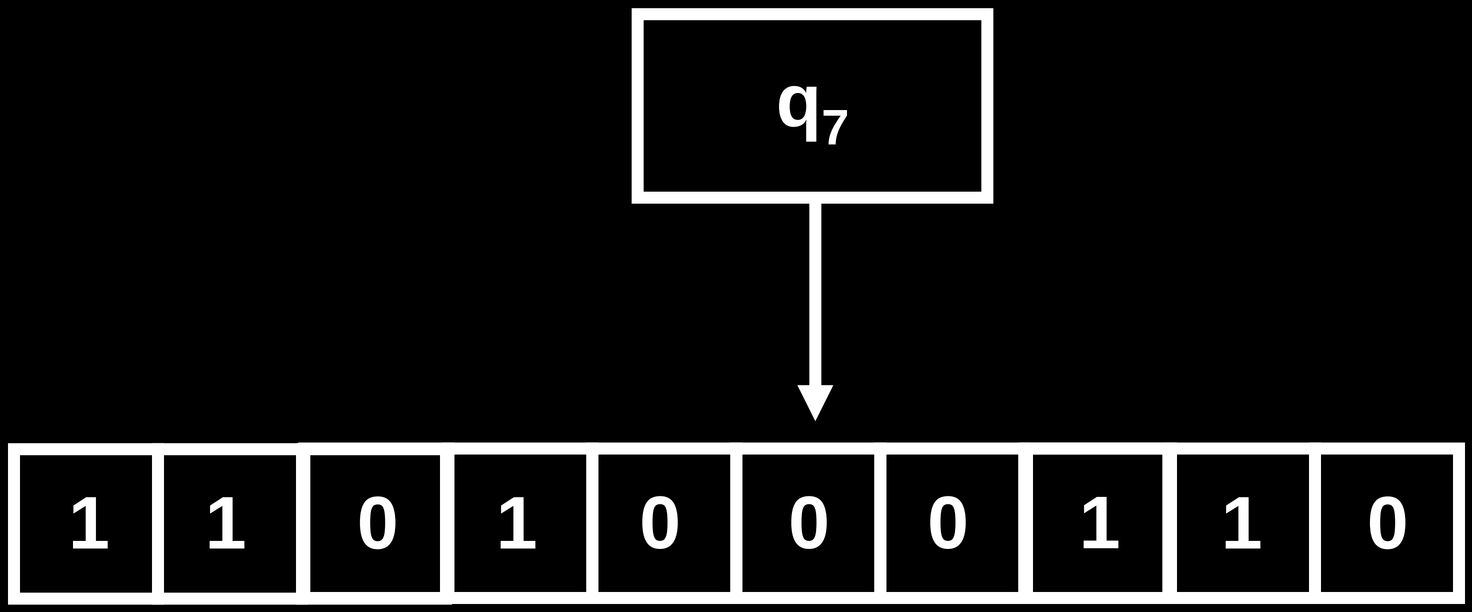
$q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$

TURING MACHINE



CONFIGURATIONS

$$11010q_7 00110$$



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

n states

start state

reject state

$$0^n 1 0^m 1 0^k 1 0^s 1 0^t 1 0^r 1 0^u 1 \ldots$$

m tape symbols (first k are input symbols)

accept state

blank symbol

$$(\,(p,a),\ (q,b,L)\,) = 0^p 1 0^a 1 0^q 1 0^b 1 0$$

NB. We assume a given convention of describing TMs by strings in Σ*.

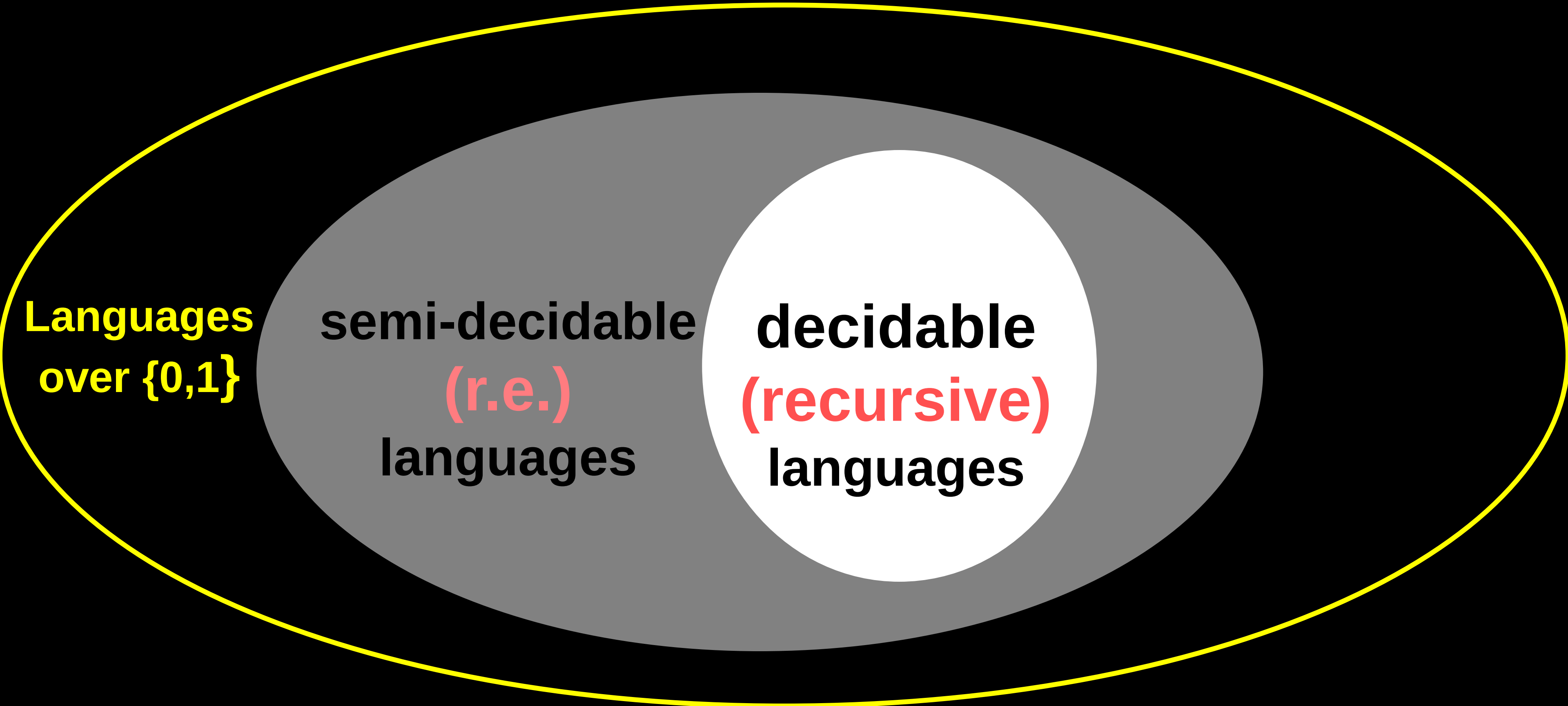
We may assume that any string in Σ* describes some TM:

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}

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

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

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

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}

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

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) | 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*.

$D_H (D_H) =$

- Reject if D_H accepts D_H
  (i.e. if $H(D_H , D_H) =$ Accept)
- Accept if D_H rejects D_H
  (i.e. if $H(D_H , D_H) =$ Reject)
- Loops if D_H loops on D_H
  (i.e. if $H(D_H , D_H)$ loops)

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 $t : \Sigma^* \times \Sigma^* \to \Sigma^*$.

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

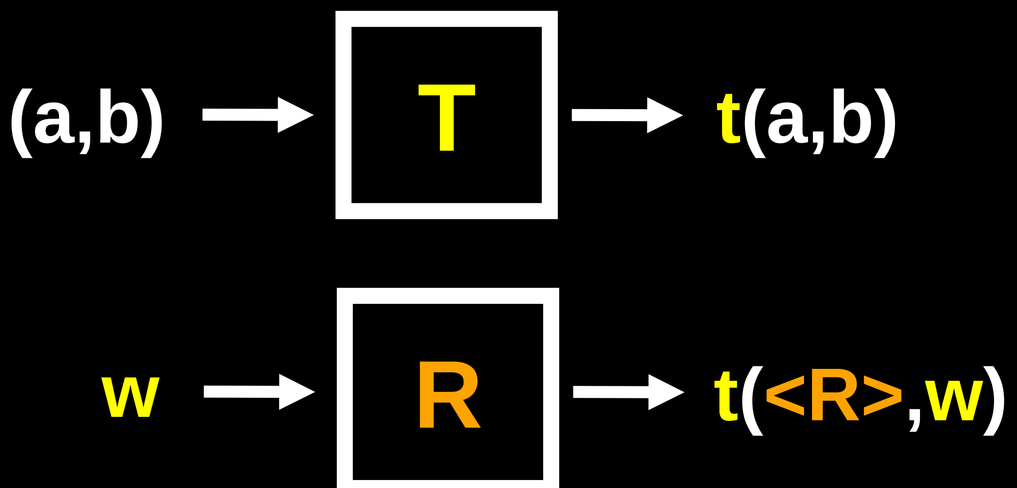
$$r(w) = t(\langle R \rangle, w)$$

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,

$$r(w) = t(\langle R \rangle, w)$$



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) = T_H**(<R>, w) =**

Reject if H (<R>, w) accepts

Accept if H (<R>, w) rejects

Contradiction!

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

Theorem: MIN_{TM} is not RE.

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

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

Theorem: MIN_{TM} is not RE.

Proof (using the Recursion Theorem):

Assume E enumerates 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?

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

Theorem: MIN_{TM} is not RE.

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

Assume E enumerates MIN_{TM} (Formal Proof)

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?

THE FIXED-POINT THEOREM

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

Proof: Pseudocode for the TM **R**:

(Informal Proof)

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 $f : \Sigma^* \to \Sigma^*$ be a computrable 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) 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 computrable 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 Σ^* (or in each TM).

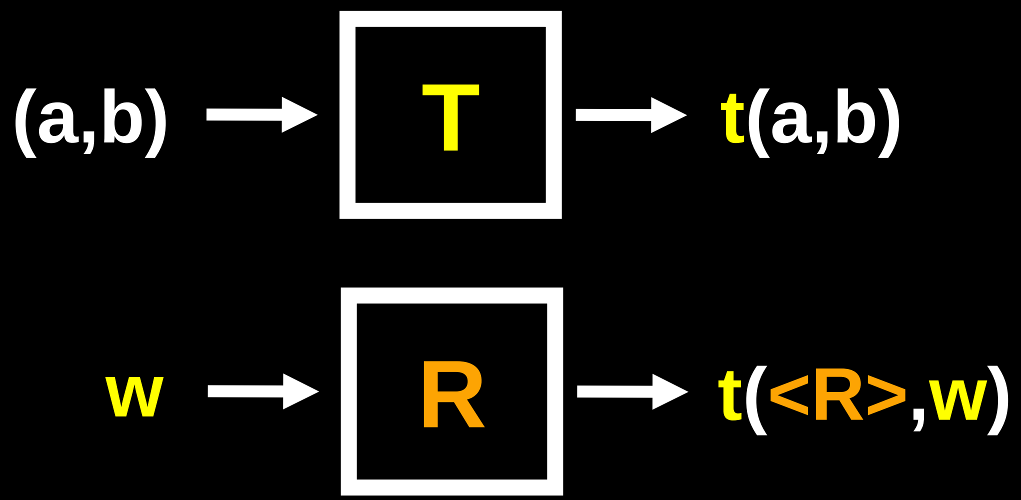
Then there is a TM R that "remains uninfected".

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,

$$r(w) = t(\langle R \rangle, w)$$



THE RECURSION THEOREM

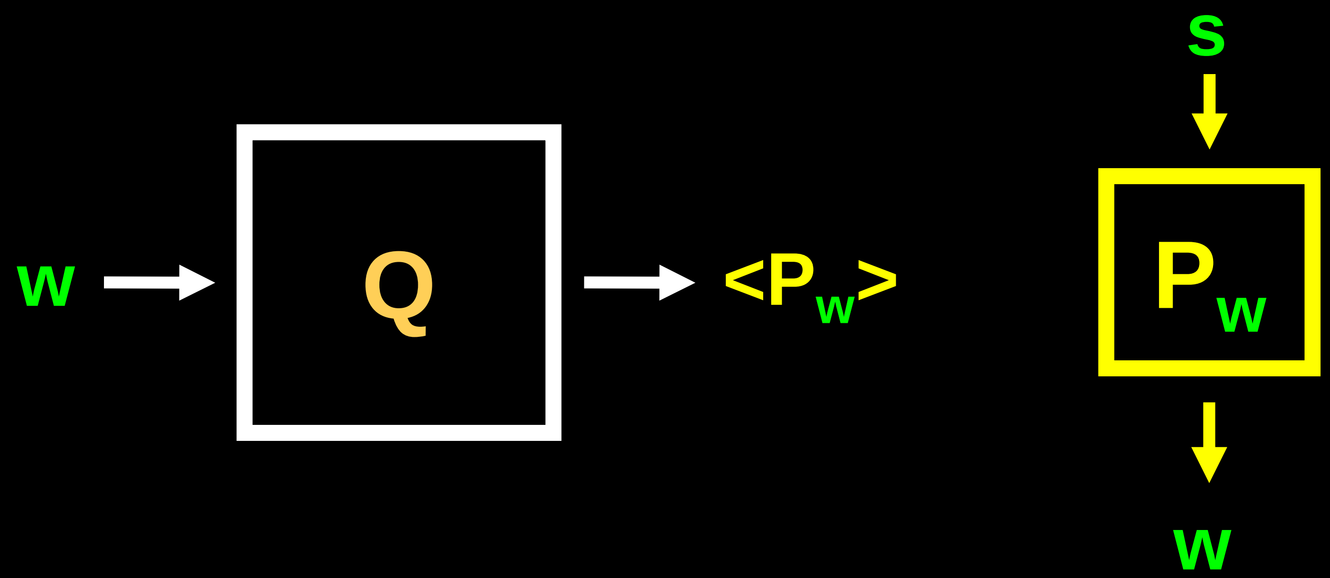
Theorem: **Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$.**

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

$$r(w) = t(\langle R \rangle, w)$$

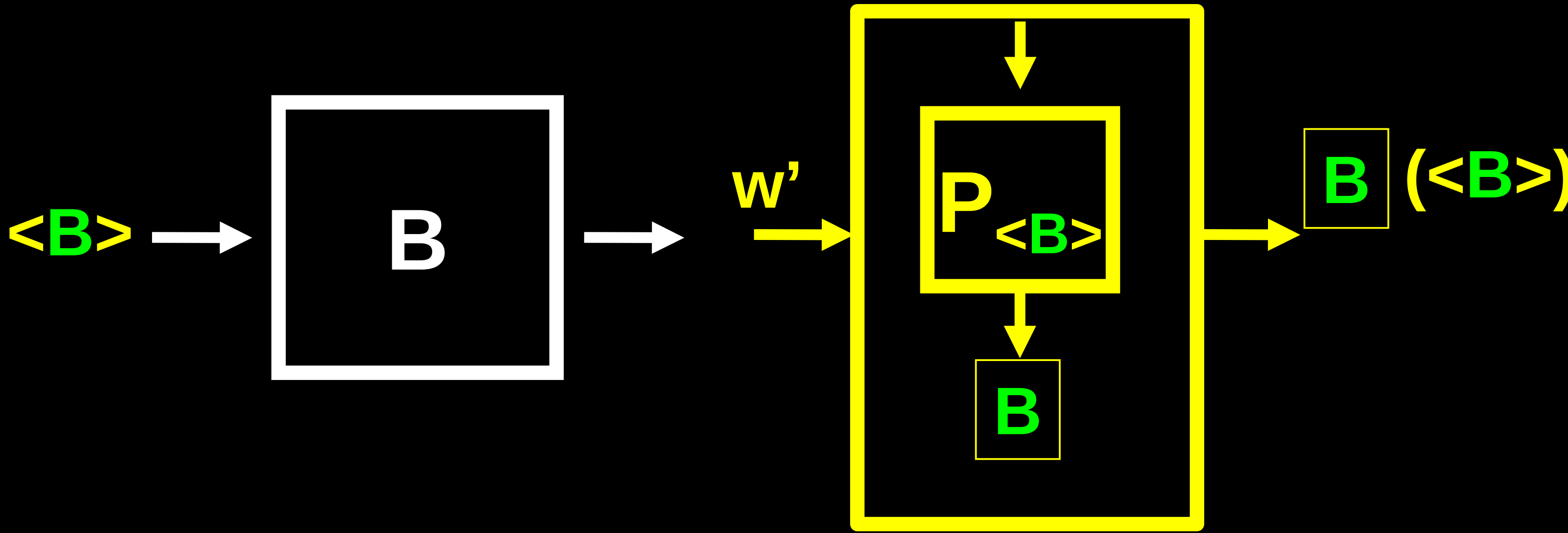
So first, need to show how to construct a TM that computes its own description (ie code).

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



TM Q computes q

A TM SELF THAT PRINTS <SELF>



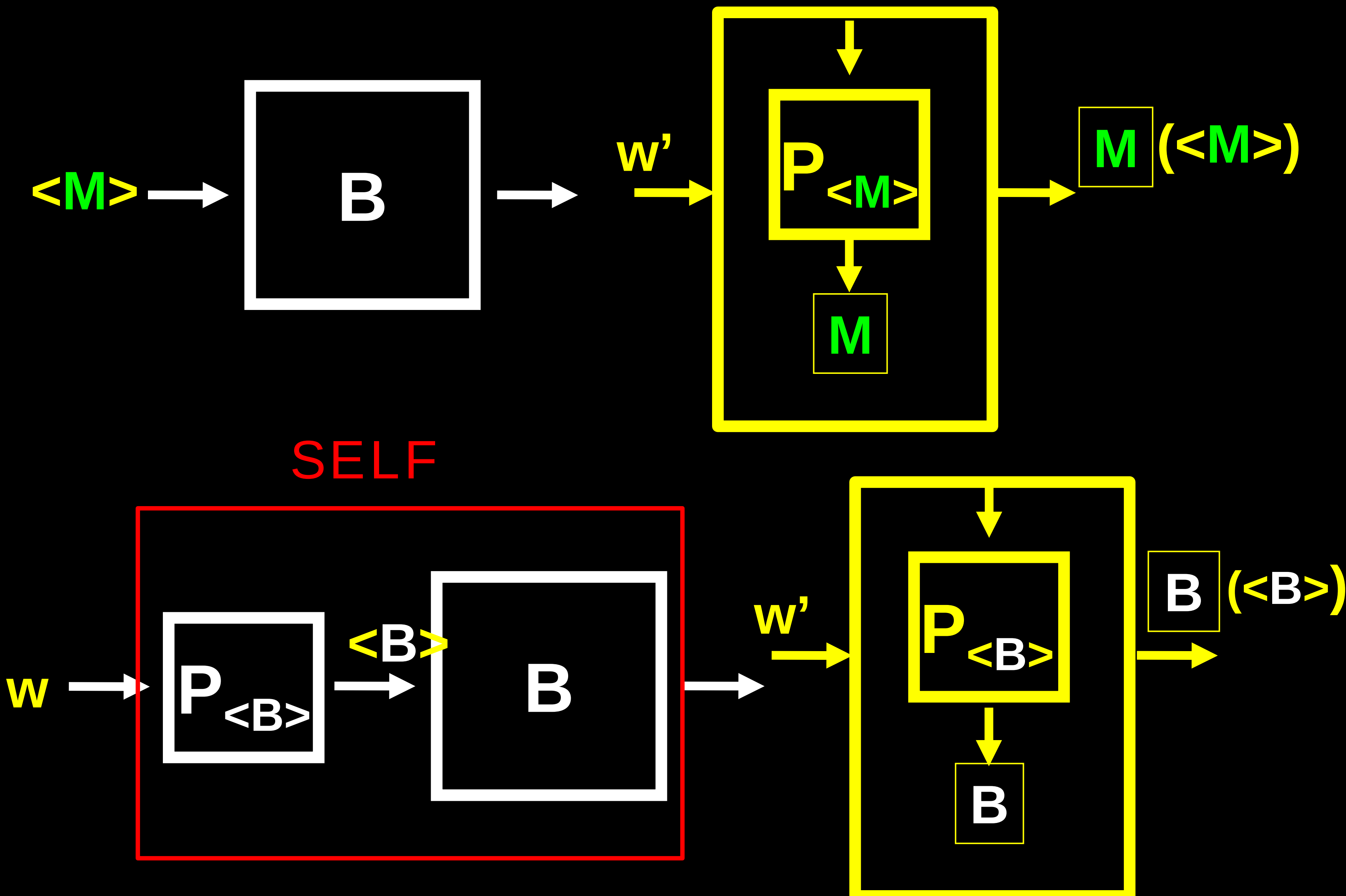
$B(\langle M\rangle) = \langle P_{\langle M\rangle} M\rangle$ **where** $P_{\langle M\rangle} M (w') = M(\langle M\rangle)$

So, $B(\langle B\rangle) = \langle P_{\langle B\rangle} B\rangle$ where $P_{\langle B\rangle} B (w') = B(\langle B\rangle)$

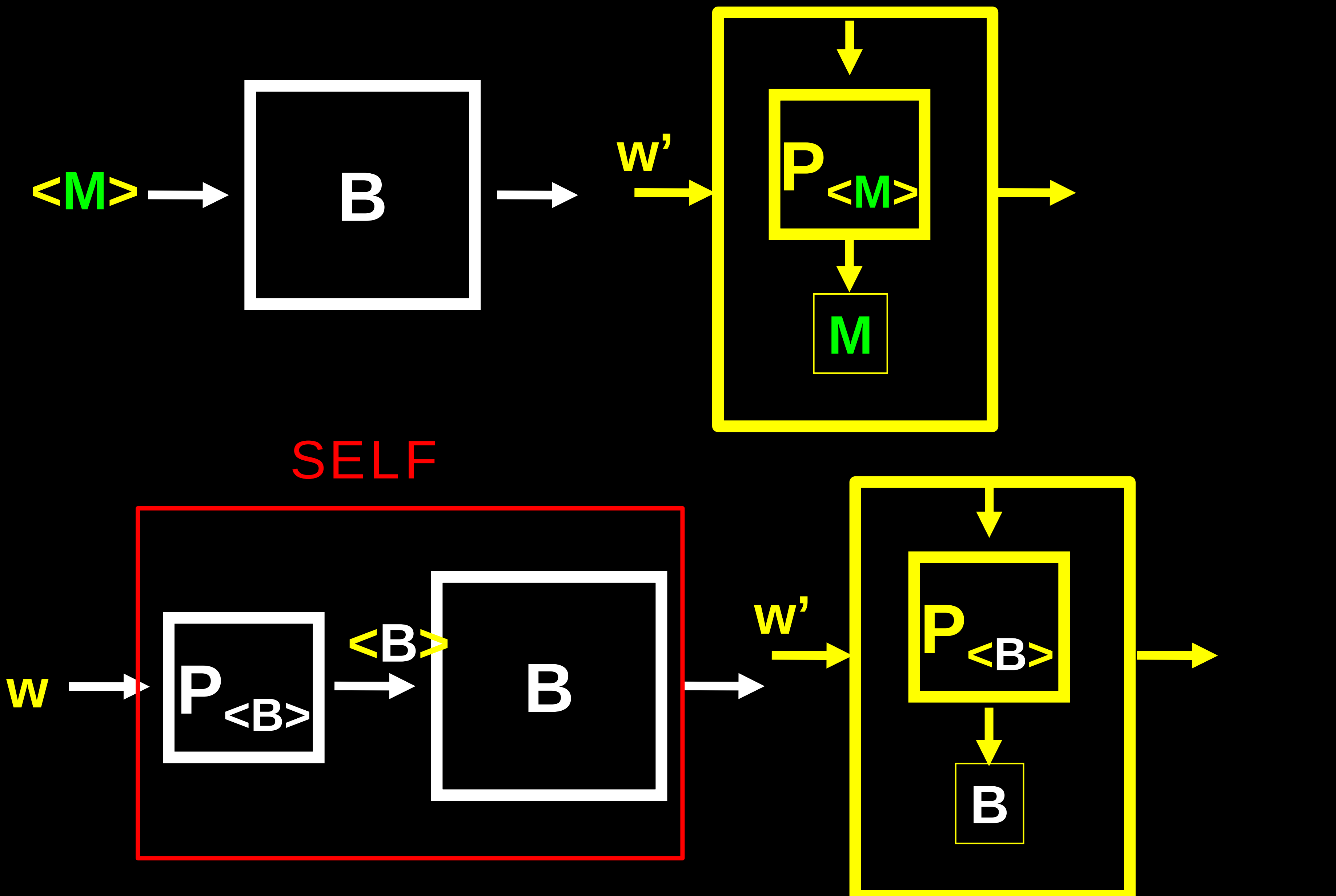
Now, $P_{\langle B\rangle} B (w') = B(\langle B\rangle) = \langle P_{\langle B\rangle} B\rangle$

So, let **SELF** $= P_{\langle B\rangle} B$

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?**

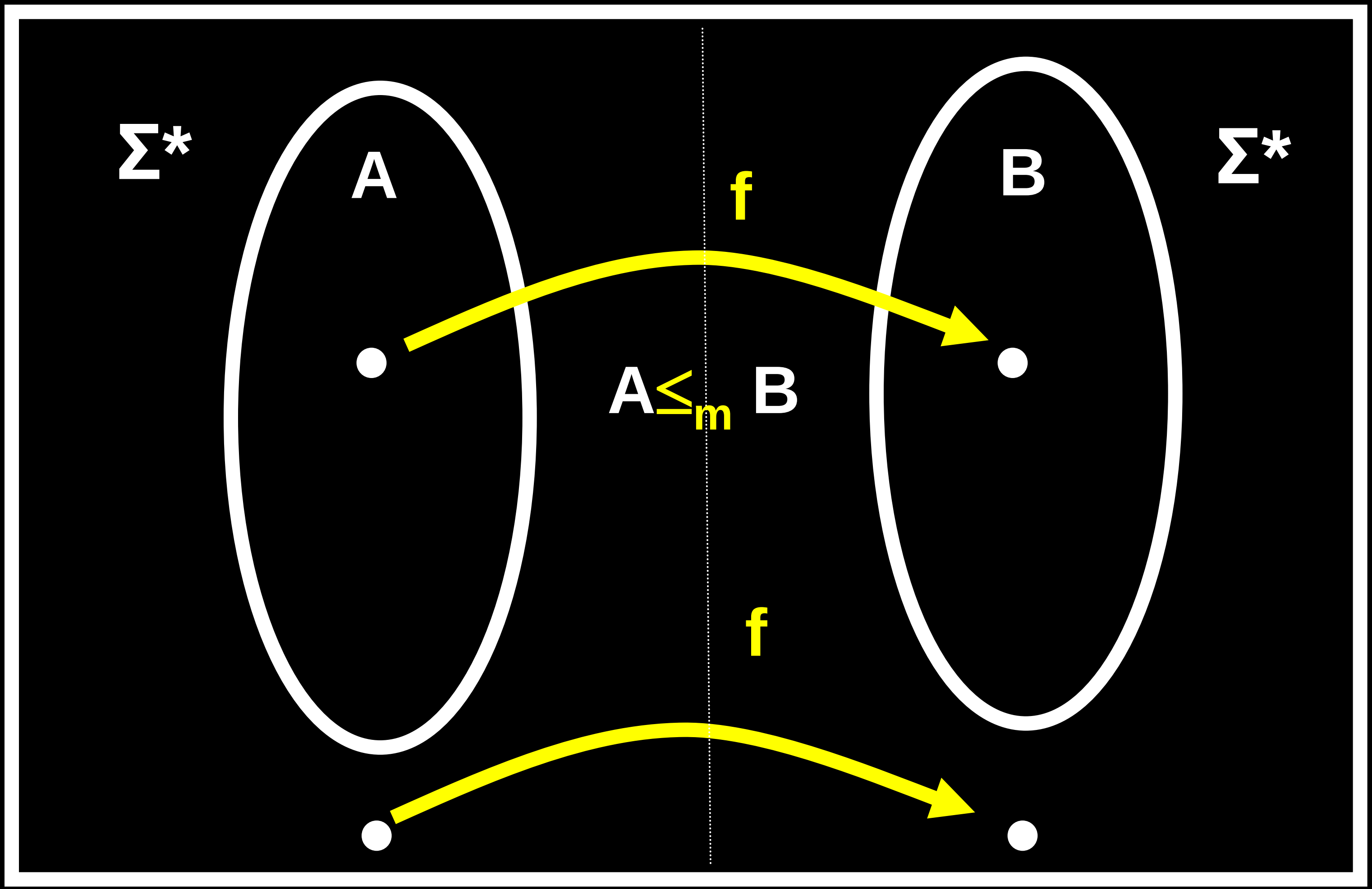
Print this sentence.

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

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

Let $f : \Sigma^* \rightarrow \Sigma^*$ be a **computable function**

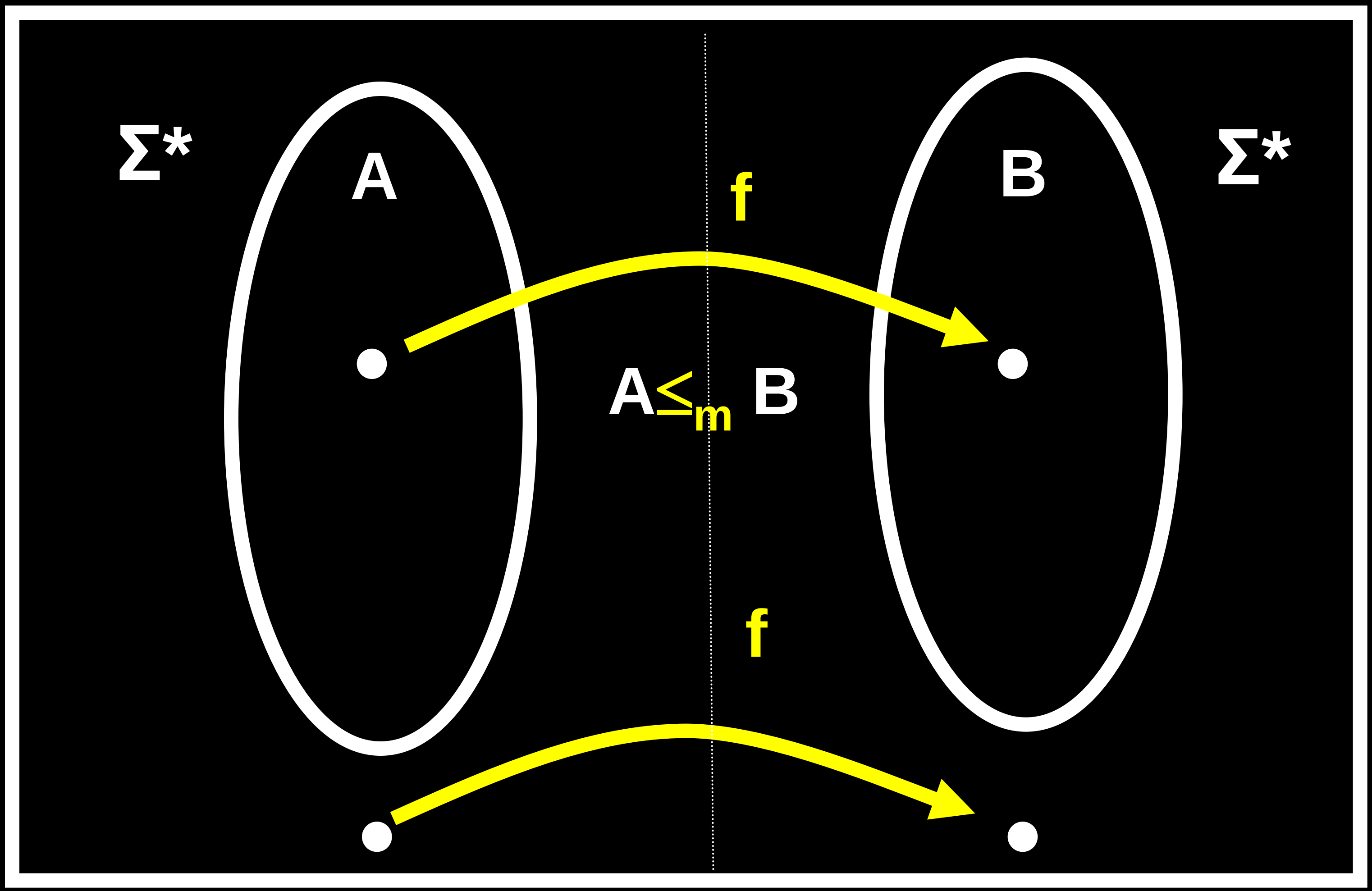
such that $w \in A \Leftrightarrow f(w) \in B$



Say: **A is Mapping Reducible to B**

Write: $A \leq_m B$ (also, $\neg A \leq_m \neg B$ (why?))

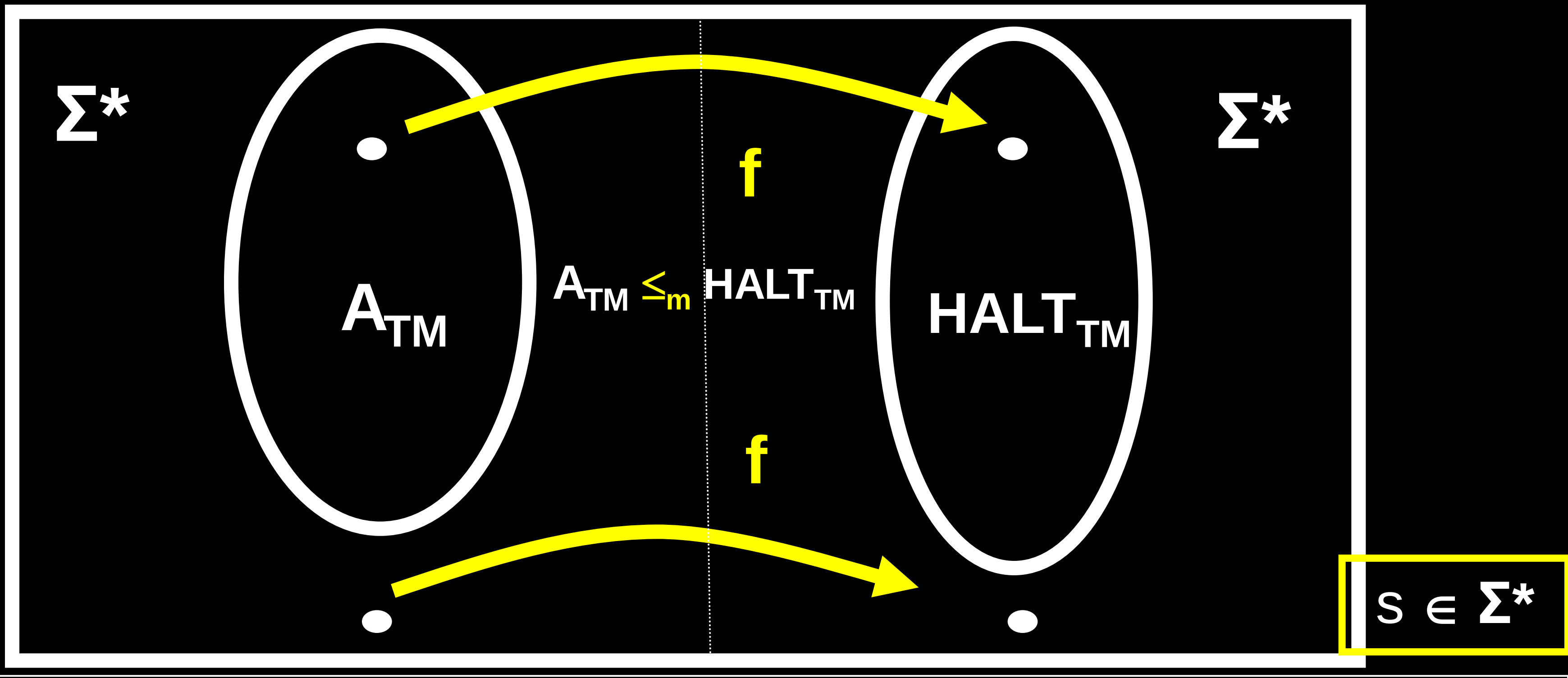
Let f : Σ* → Σ* be a computable function such that w ∈ A ⇔ f(w) ∈ B



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

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

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

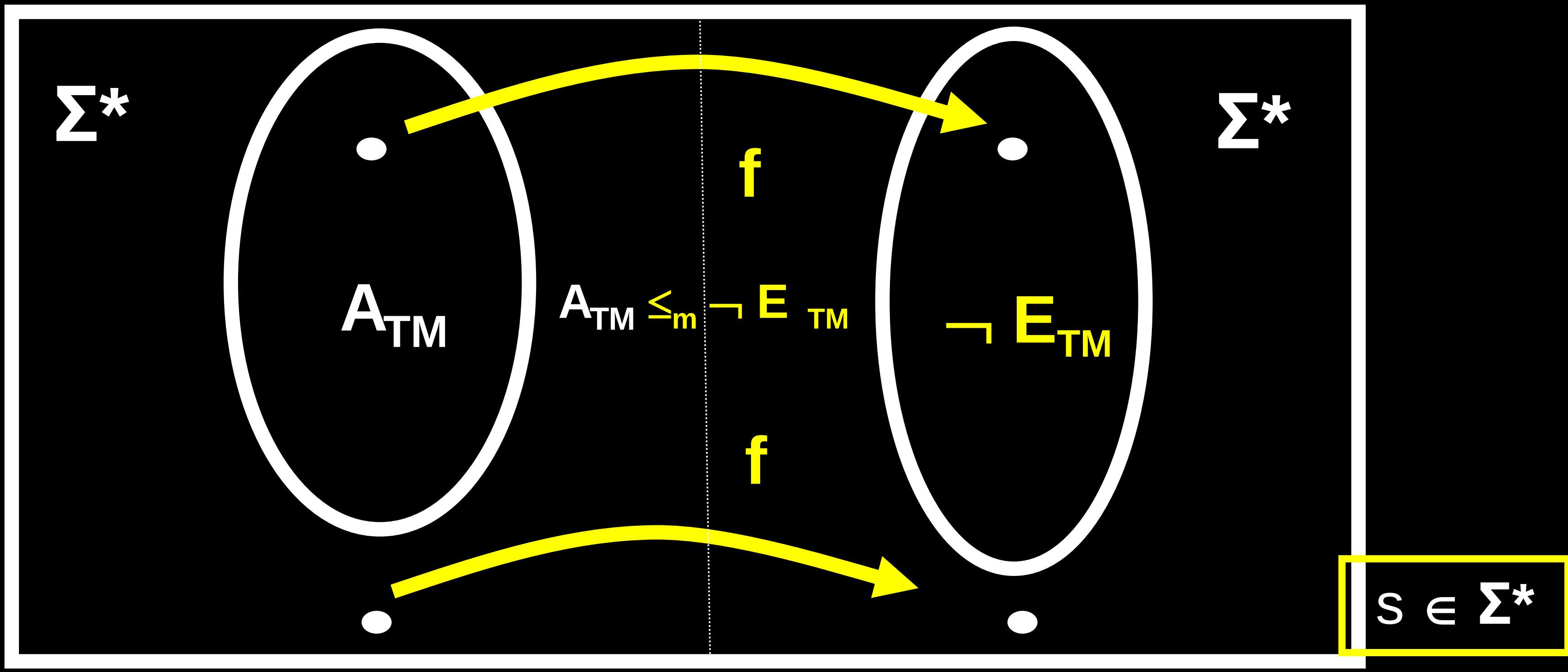


f: (M,w) → (M', w) where **M'(s) = M(s)** if **M(s)** accepts,
Loops otherwise

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

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

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

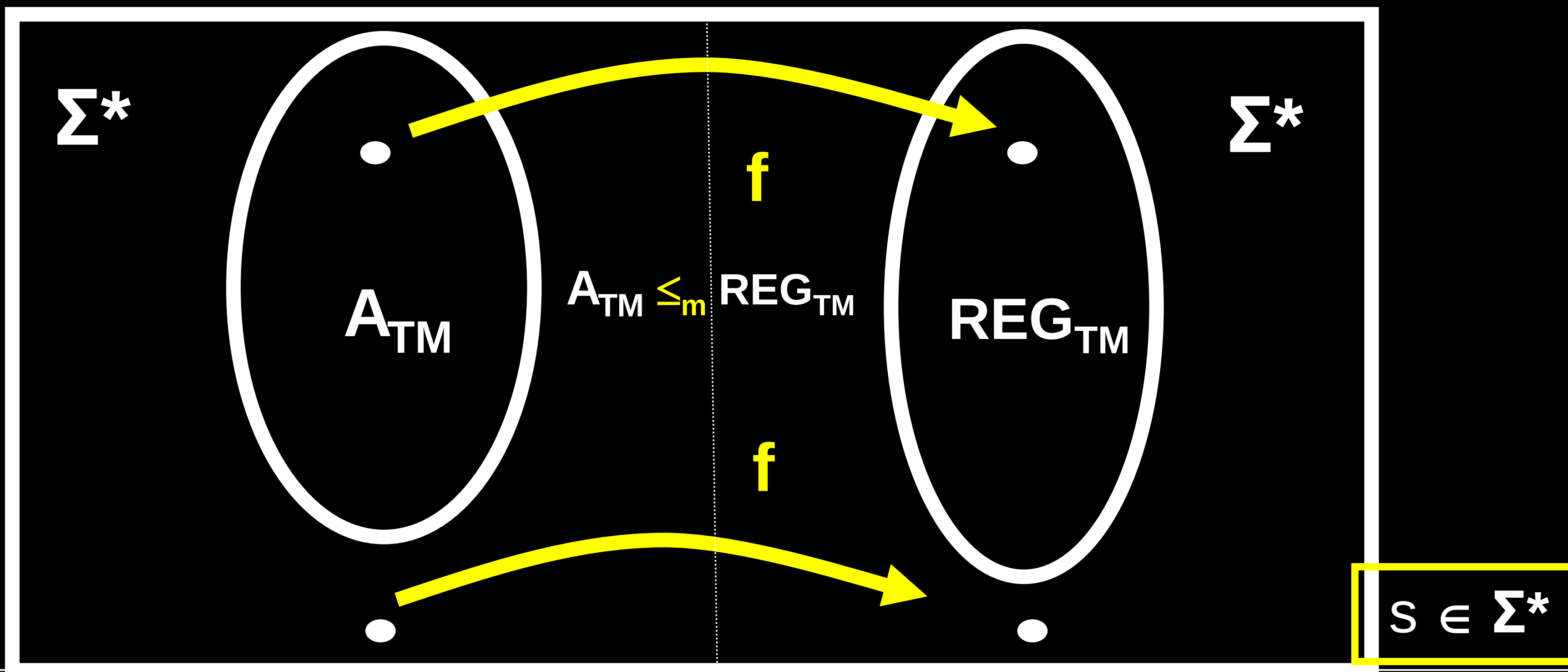


f: **(M,w) → M_w** where **M_w (s) = M(w)** if **s = w,**
Loops otherwise

So, (M, w) ∈ **A_{TM}** ⇔ M_w ∈ **¬ E_{TM}**

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

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



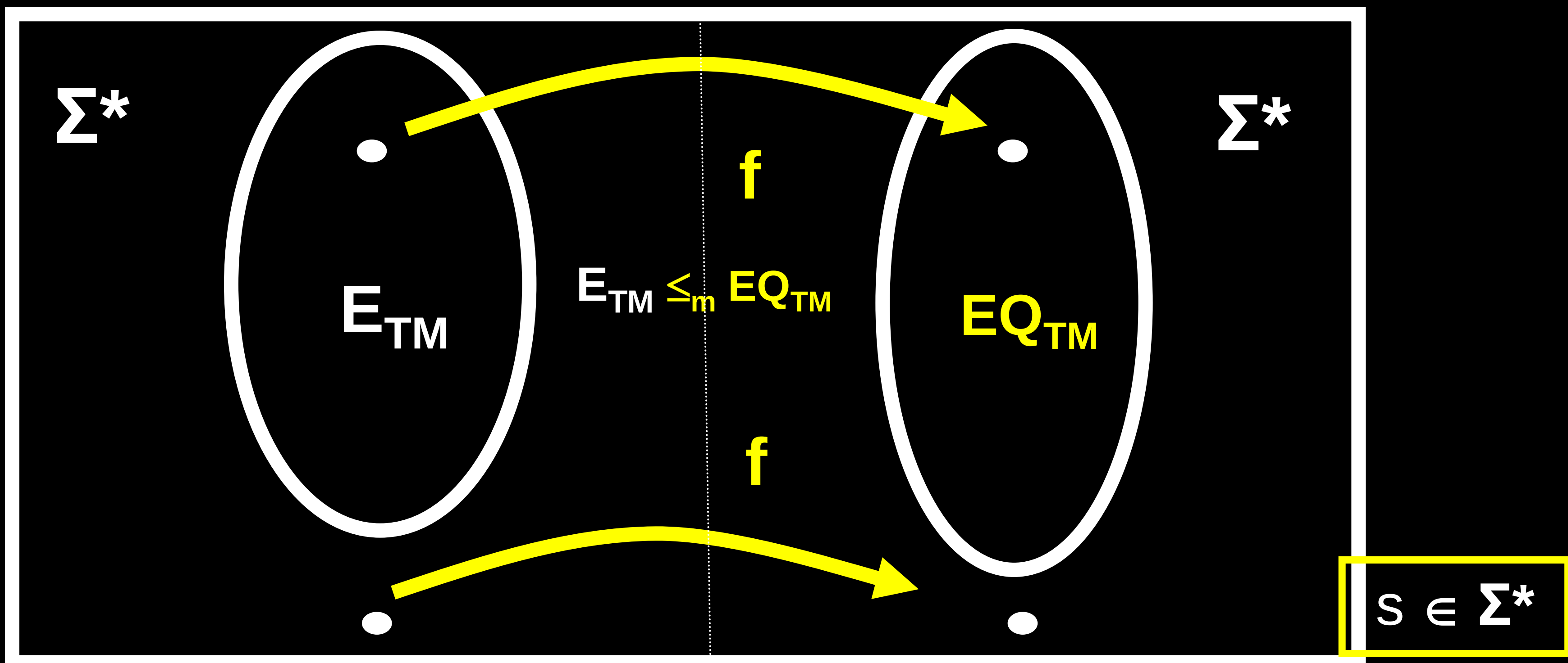
f: (M,w) → M'_w where M'_w (s) = accept if s = 0^n1^n,

M(w) otherwise

So, (M, w) ∈ A_{TM} ⇔ M'_w ∈ REG_{TM}

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

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

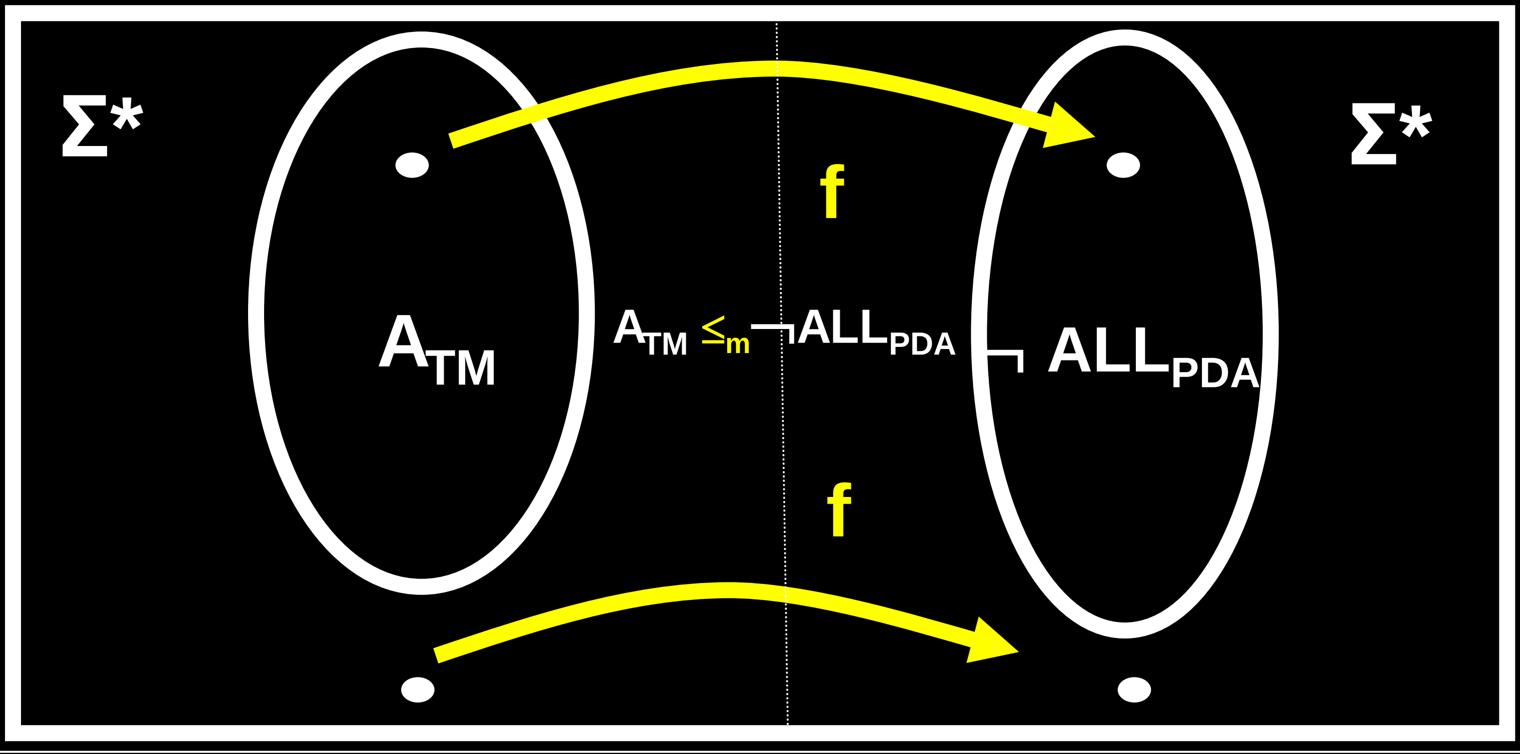


f: $M \rightarrow (M, M_{\varnothing})$ where $M_{\varnothing}(s) = \text{Loops}$

So, $M \in E_{TM} \Leftrightarrow (M, M_{\varnothing}) \in EQ_{TM}$

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

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



f: (M,w) → PDA P_w where

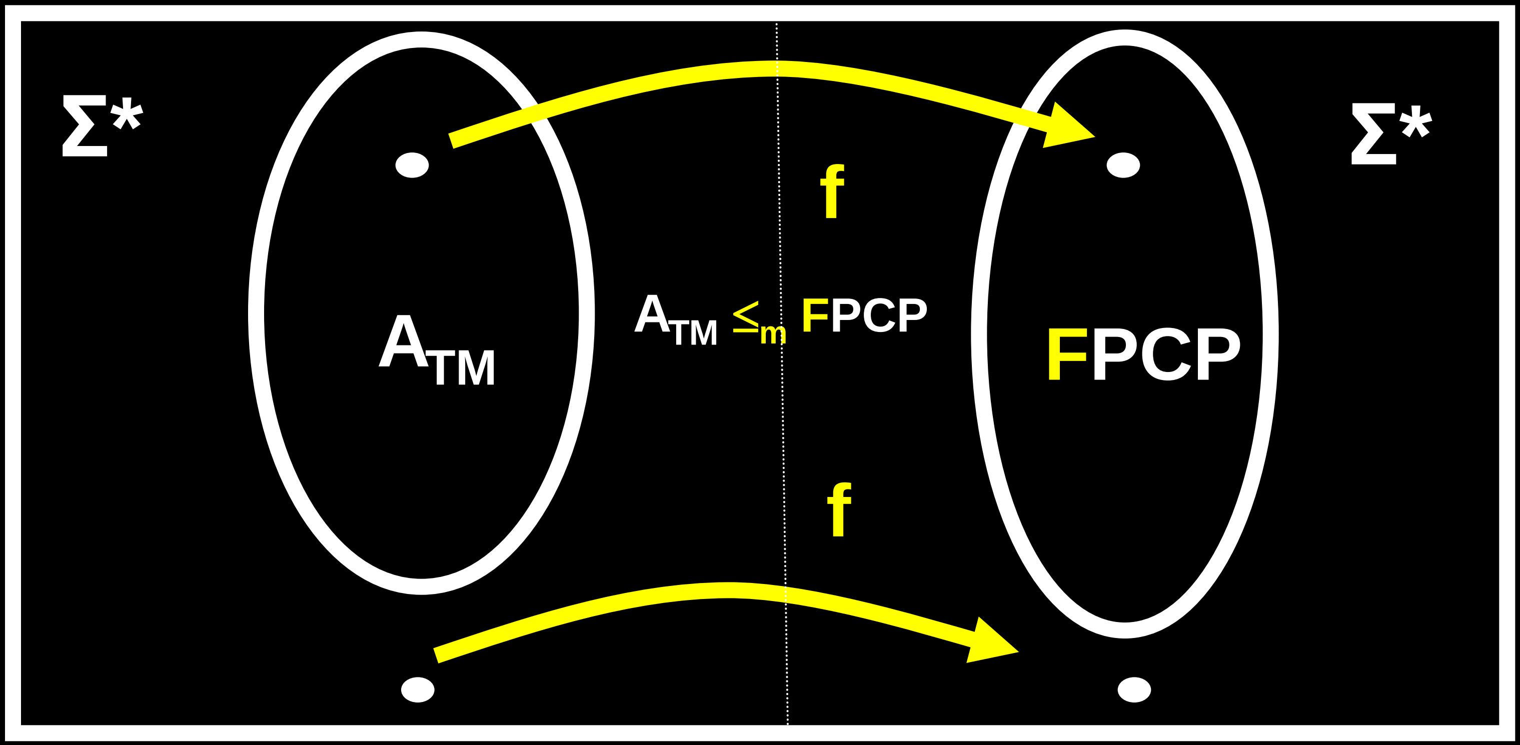
s ∈ **Σ***

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

So, (M, w) ∈ A_{TM} ⇔ P_w ∈ ¬ **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 }

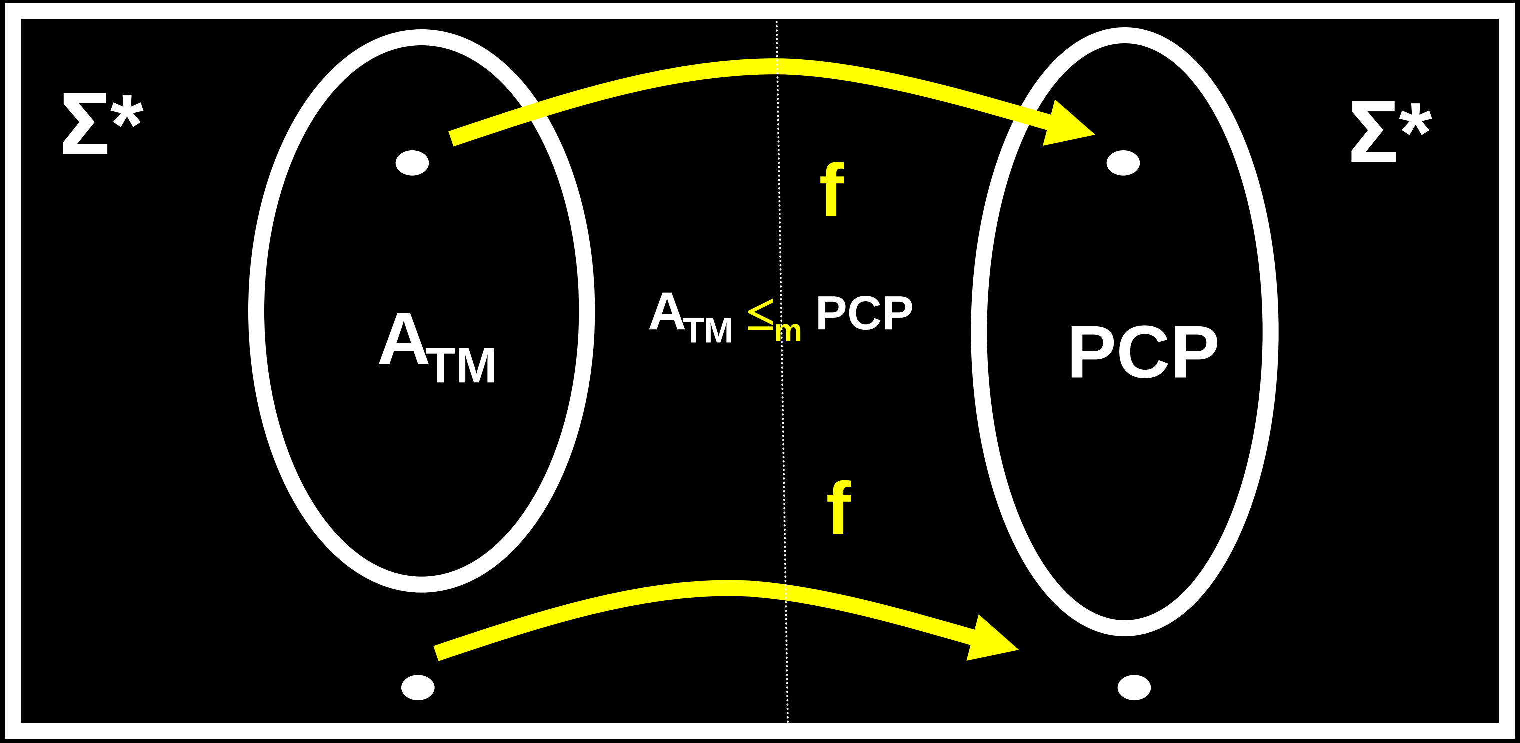


Construct f: (M,w) → $P_{(M,w)}$ such that

(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) = $\emptyset$ }

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

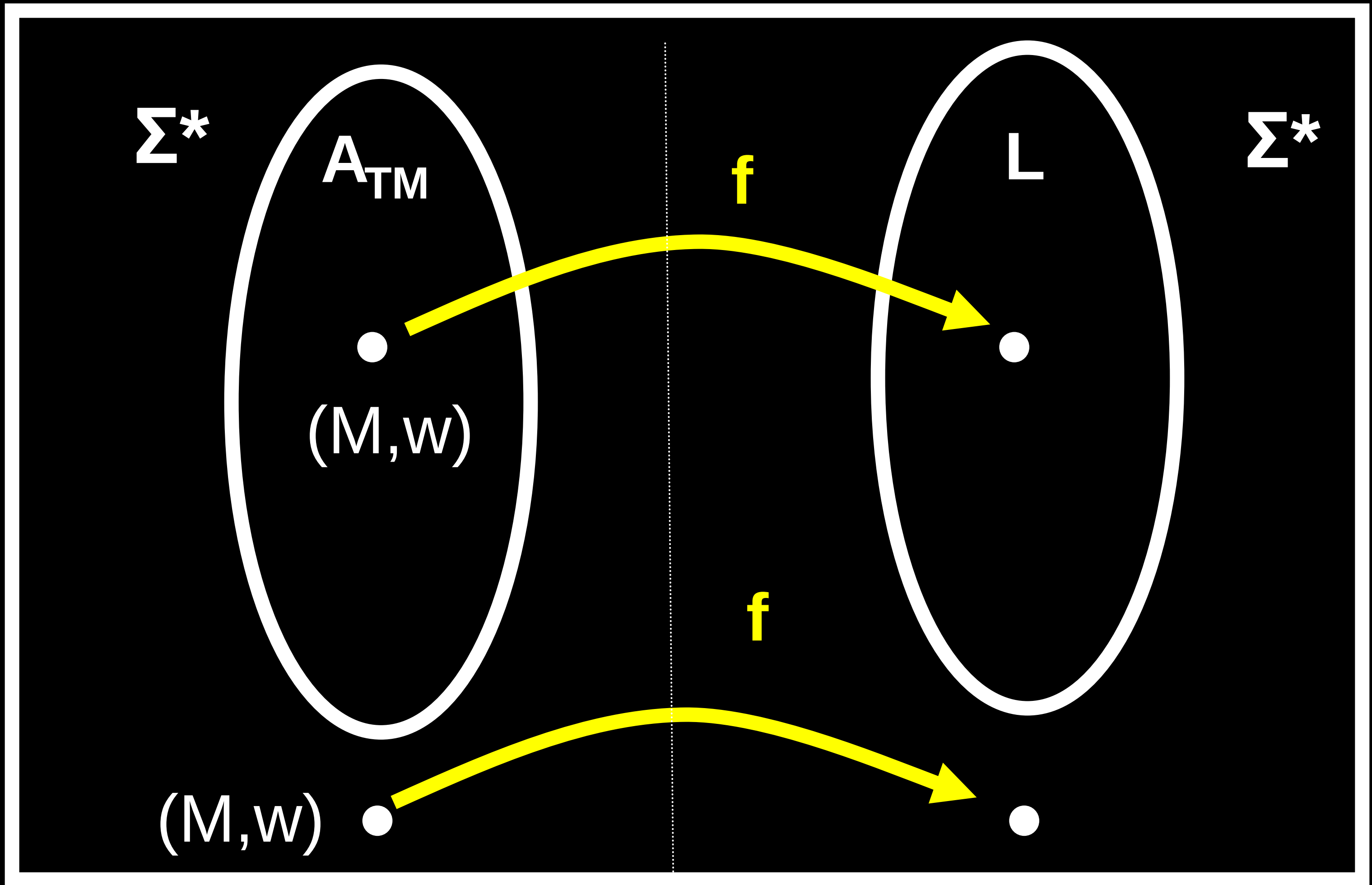
FIN_{TM} = { M | M is a TM and L(M) is finite}

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

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

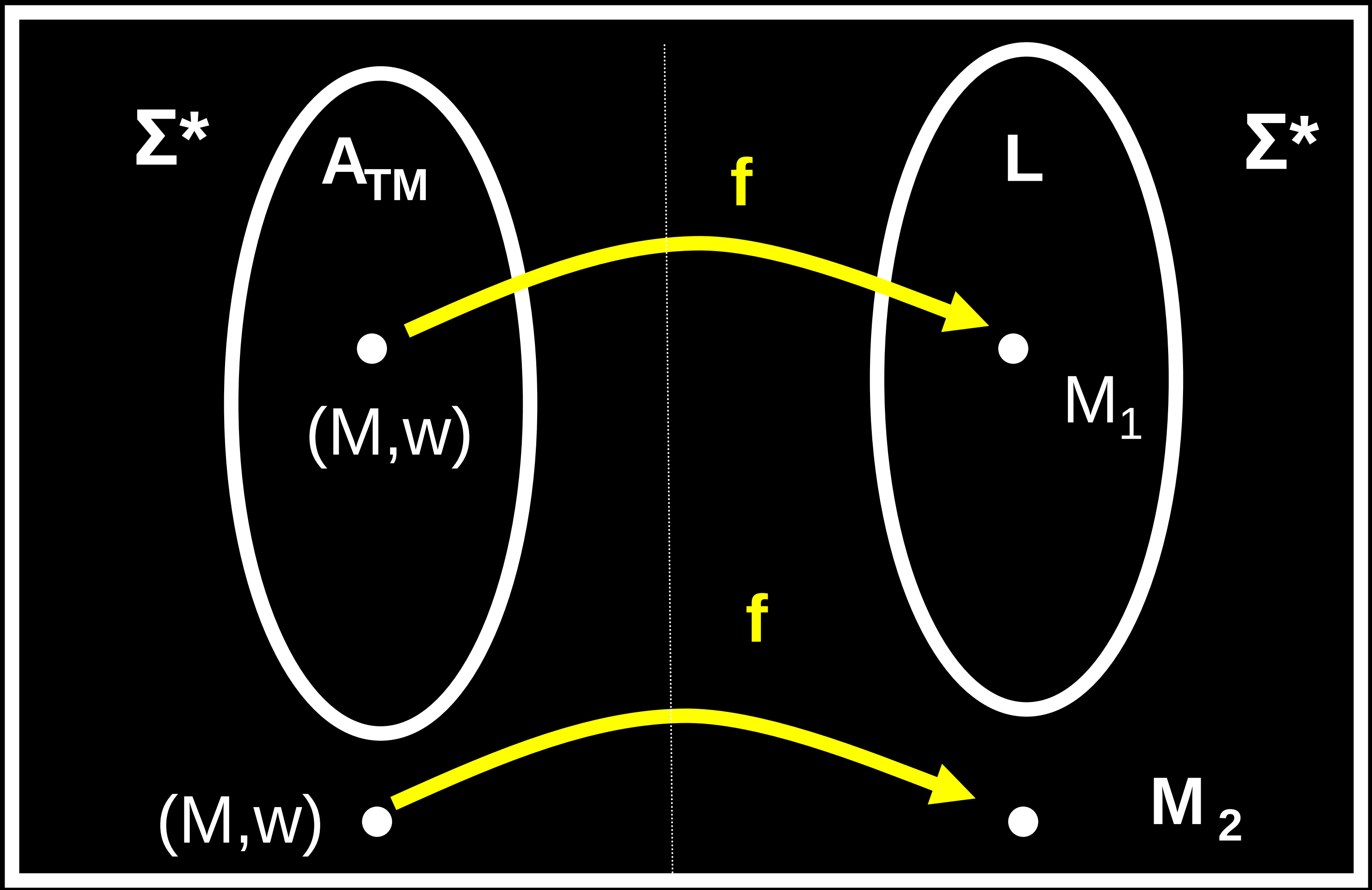
Proof: Show L is undecidable

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 $M_\emptyset$ 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)

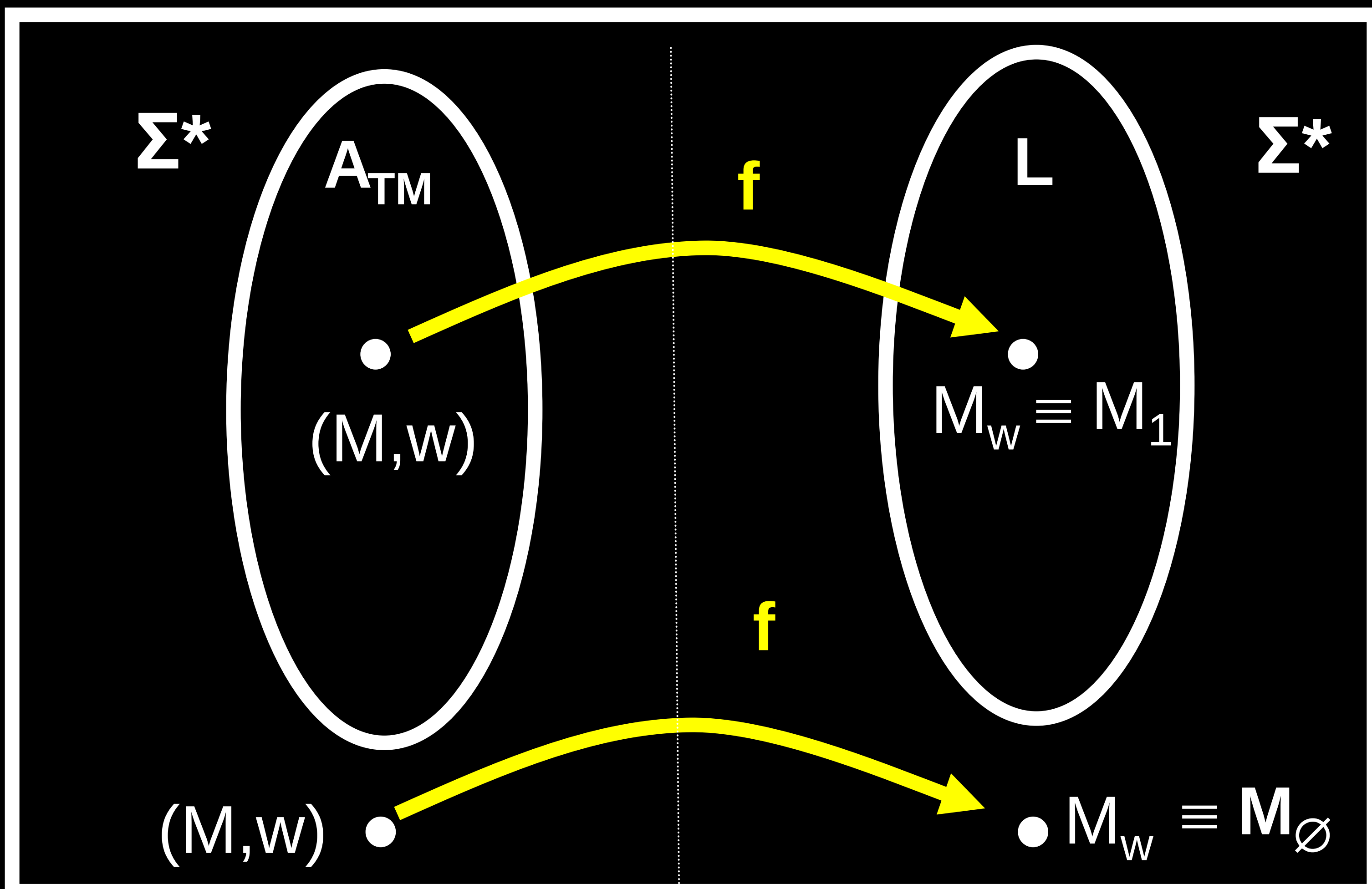
Show A_{TM} is mapping reducible to L :

Map $(M, w) \rightarrow M_w$ 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

q_{YES}

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

YES

I	N	P	U	T					

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

$\leq_T$ 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 : $\Sigma^* \rightarrow \Sigma^*$, 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?**

THE ARITHMETIC HIERARCHY

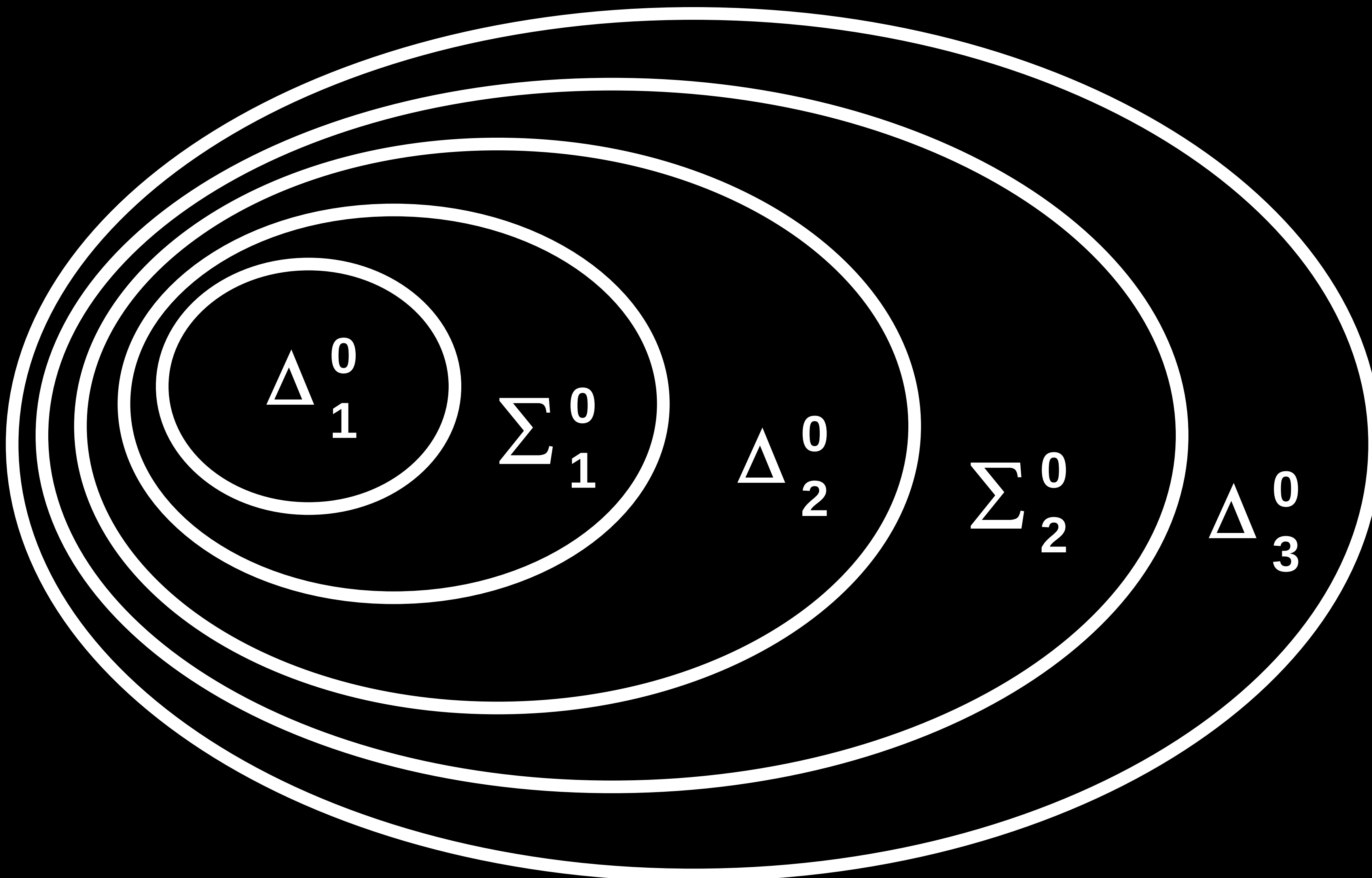
Δ^0_1 = { decidable sets } (sets = languages)

Σ^0_1 = { semi-decidable sets }

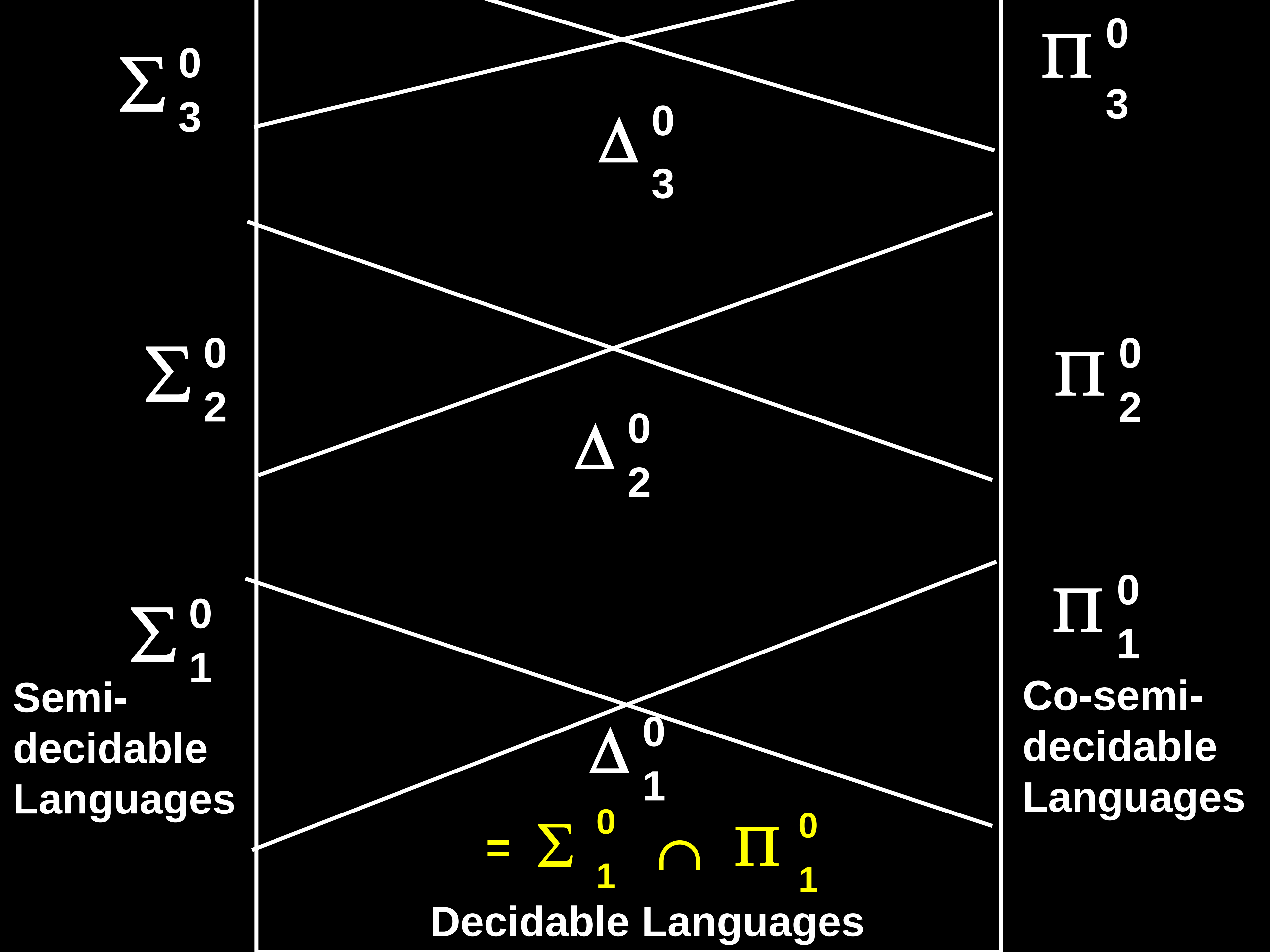
Σ^0_{n+1} = { sets semi-decidable in some B $\in \Sigma^0_n$ }

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

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



Δ^0_1
Σ^0_1
Δ^0_2
Σ^0_2
Δ^0_3


Σ^0_3
Π^0_3
Δ^0_3
Σ^0_2
Π^0_2
Δ^0_2
Σ^0_1
Π^0_1
Semi-decidable Languages
Co-semi-decidable Languages
Δ^0_1
$= \Sigma^0_1 \cap \Pi^0_1$
Decidable Languages

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 ∃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 $\{ x \mid \exists y_1 \forall y_2 \exists y_3 \ldots Q y_n \, R(x, y_1, \ldots, y_n) \}$

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

Example

Decidable predicate

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

We know that A_{TM} is in Σ^0_1 Why?

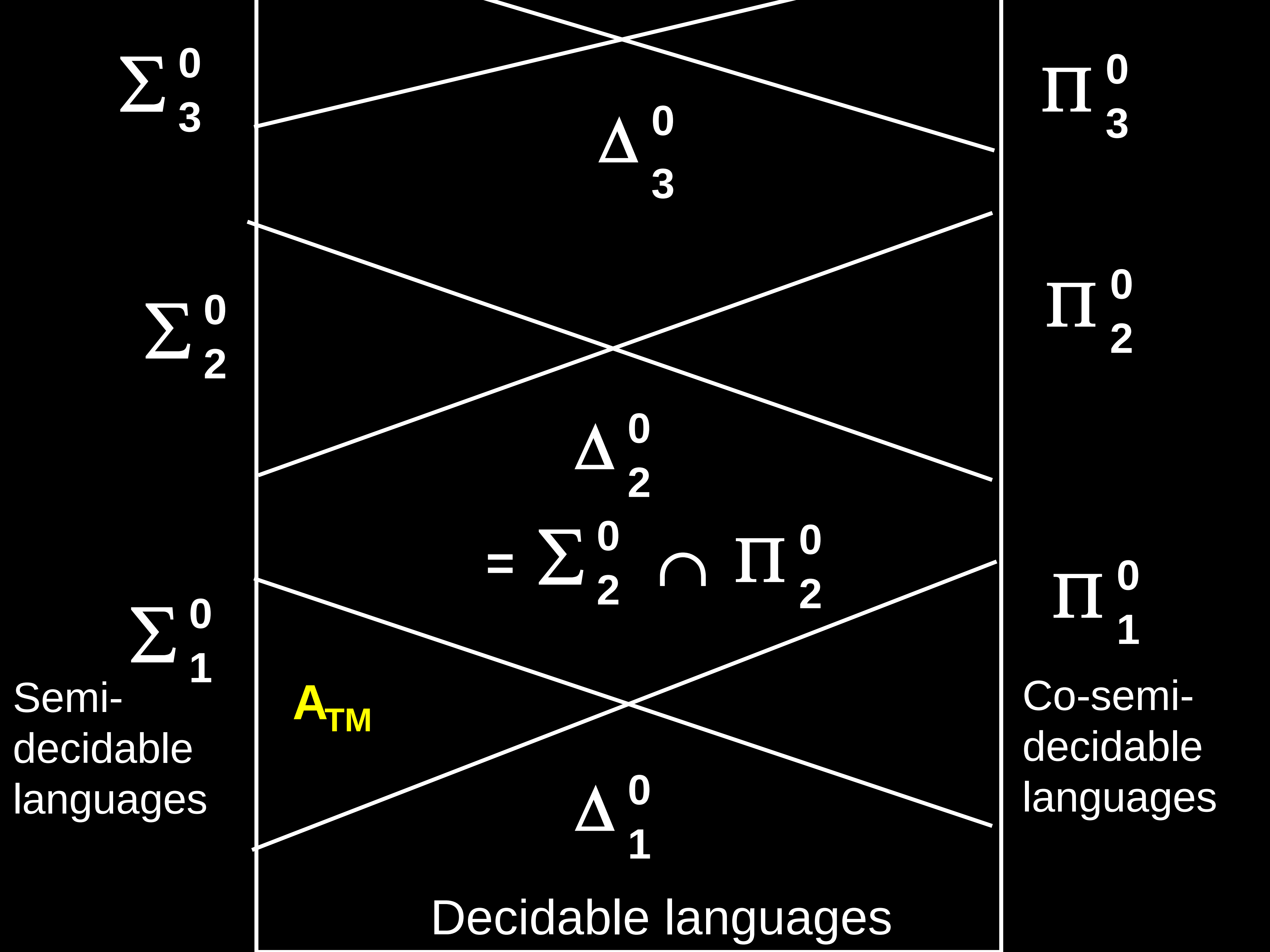
Show it can be described in this form:

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}



Σ_3^0
Π_3^0
Δ_3^0
Σ_2^0
Π_2^0
Δ_2^0
$= \Sigma_2^0 \cap \Pi_2^0$
Σ_1^0
Π_1^0
Semi-decidable languages
A_{TM}
Co-semi-decidable languages
Δ_1^0
Decidable languages

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

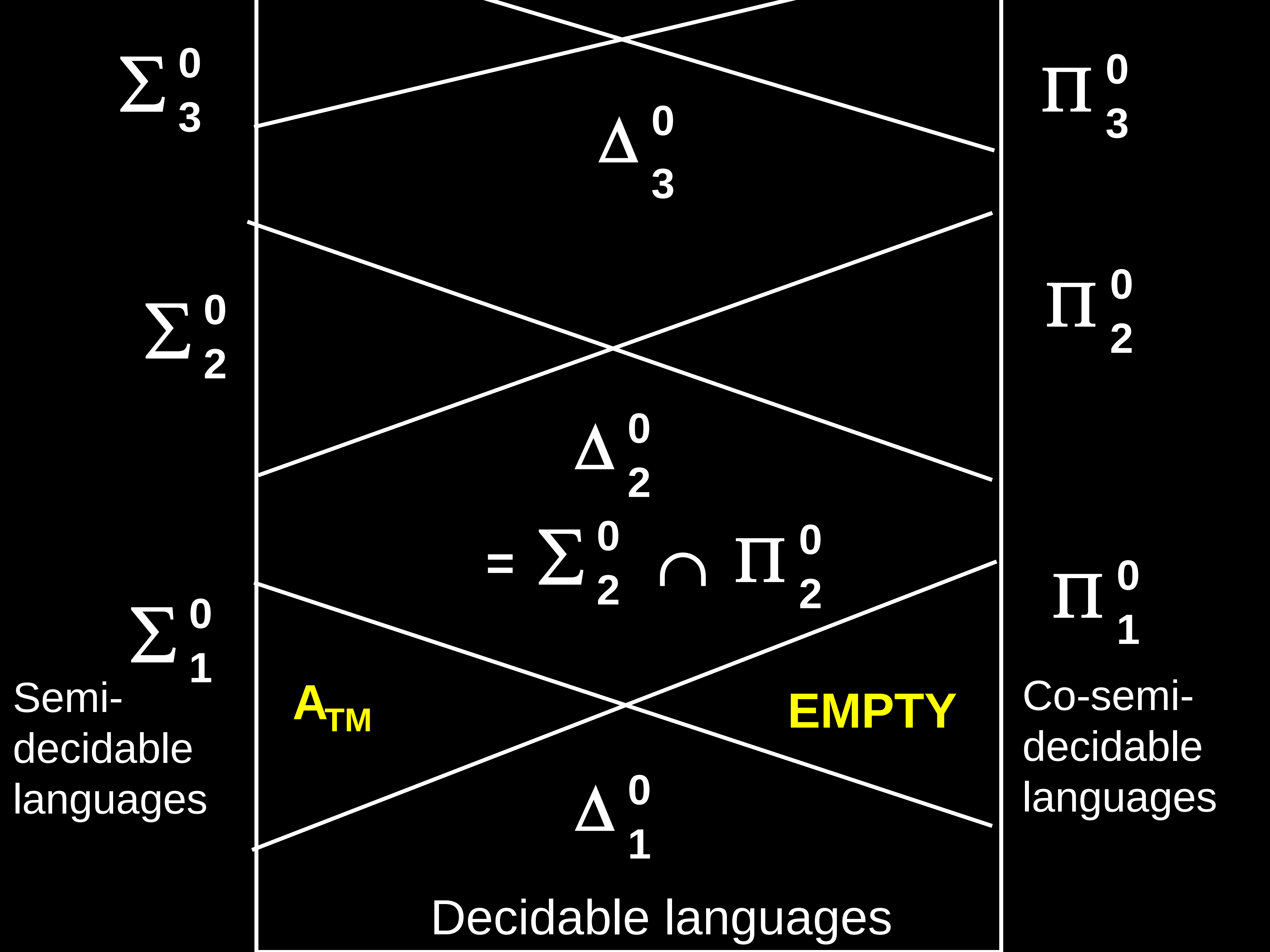
Theorem. There is a 1-1 and onto computable function $< , >: \Sigma^* \times \Sigma^* \to \Sigma^*$ and computable functions π_1 and $\pi_2 : \Sigma^* \to \Sigma^*$ such that

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

EMPTY = { M | $\forall w \forall t$ [M doesn't accept w in t steps] }

EMPTY = { M | $\forall z$[M doesn't accept $\pi_1(z)$ in $\pi_2(z)$ steps]}

EMPTY = { M | $\forall z[\neg T(\langle M\rangle, \pi_1(z), \pi_2(z))]$ }


Σ_3^0
Δ_3^0
Π_3^0
Σ_2^0
Π_2^0
Δ_2^0
$= \Sigma_2^0 \cap \Pi_2^0$
Σ_1^0
Π_1^0
Semi-decidable languages
A_{TM}
EMPTY
Co-semi-decidable languages
Δ_1^0
Decidable languages

Example

Π_2^0 = languages of the form { x | ∀y∃z R(x,y,z) }

Show that TOTAL = { M | M halts on all inputs }

is in Π_2^0

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

decidable predicate

Example

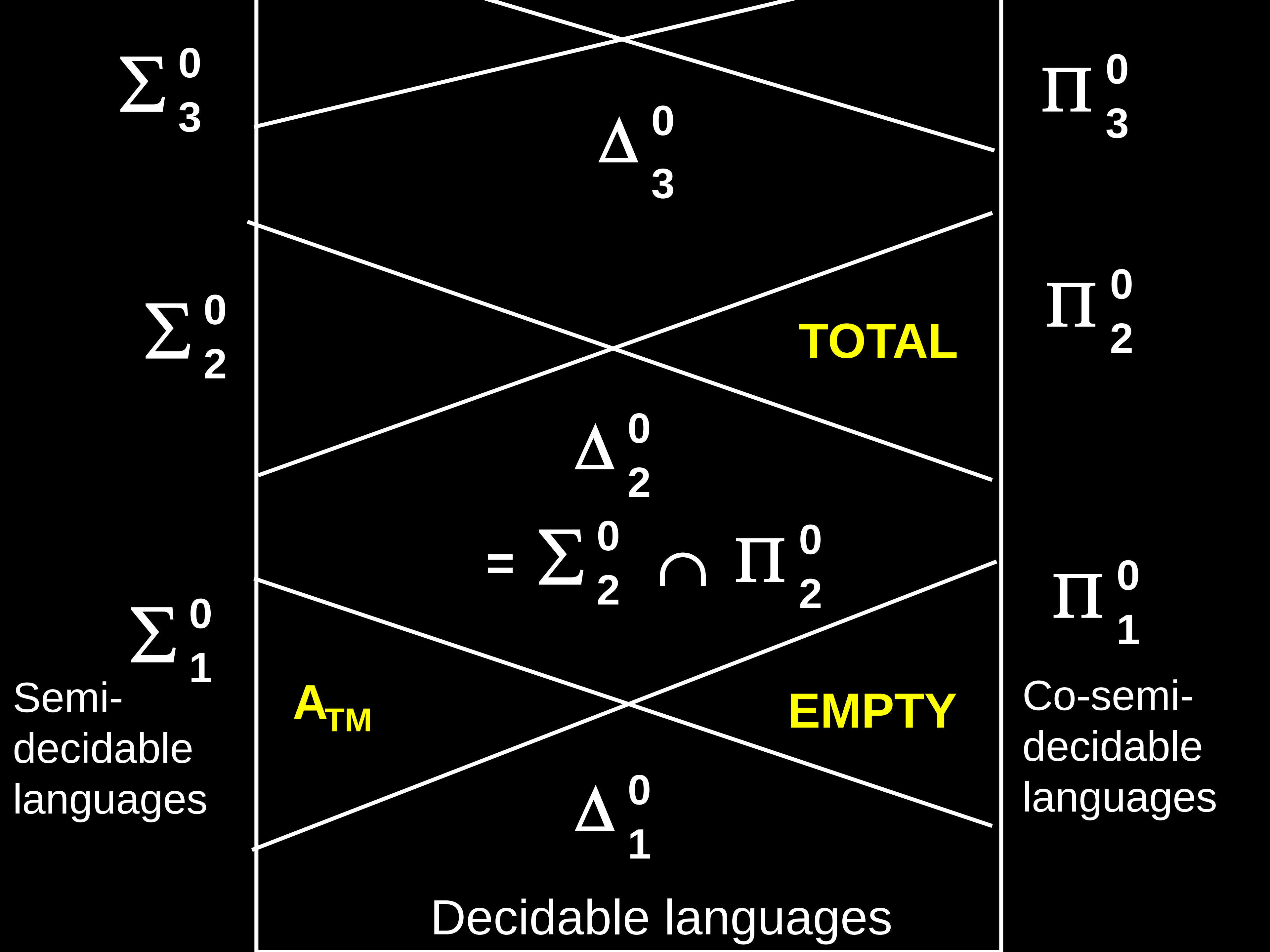
Π^0_2 = languages of the form { x | ∀y∃z R(x,y,z) }

Show that TOTAL = { M | M halts on all inputs }

is in Π^0_2

TOTAL = { M | ∀w ∃t [T(<M>, w, t)] }

decidable predicate


Σ_3^0
Δ_3^0
Π_3^0
Σ_2^0
TOTAL
Π_2^0
Δ_2^0
$= \Sigma_2^0 \cap \Pi_2^0$
Σ_1^0
Π_1^0
Semi-decidable languages
A_{TM}
EMPTY
Co-semi-decidable languages
Δ_1^0
Decidable languages

Example

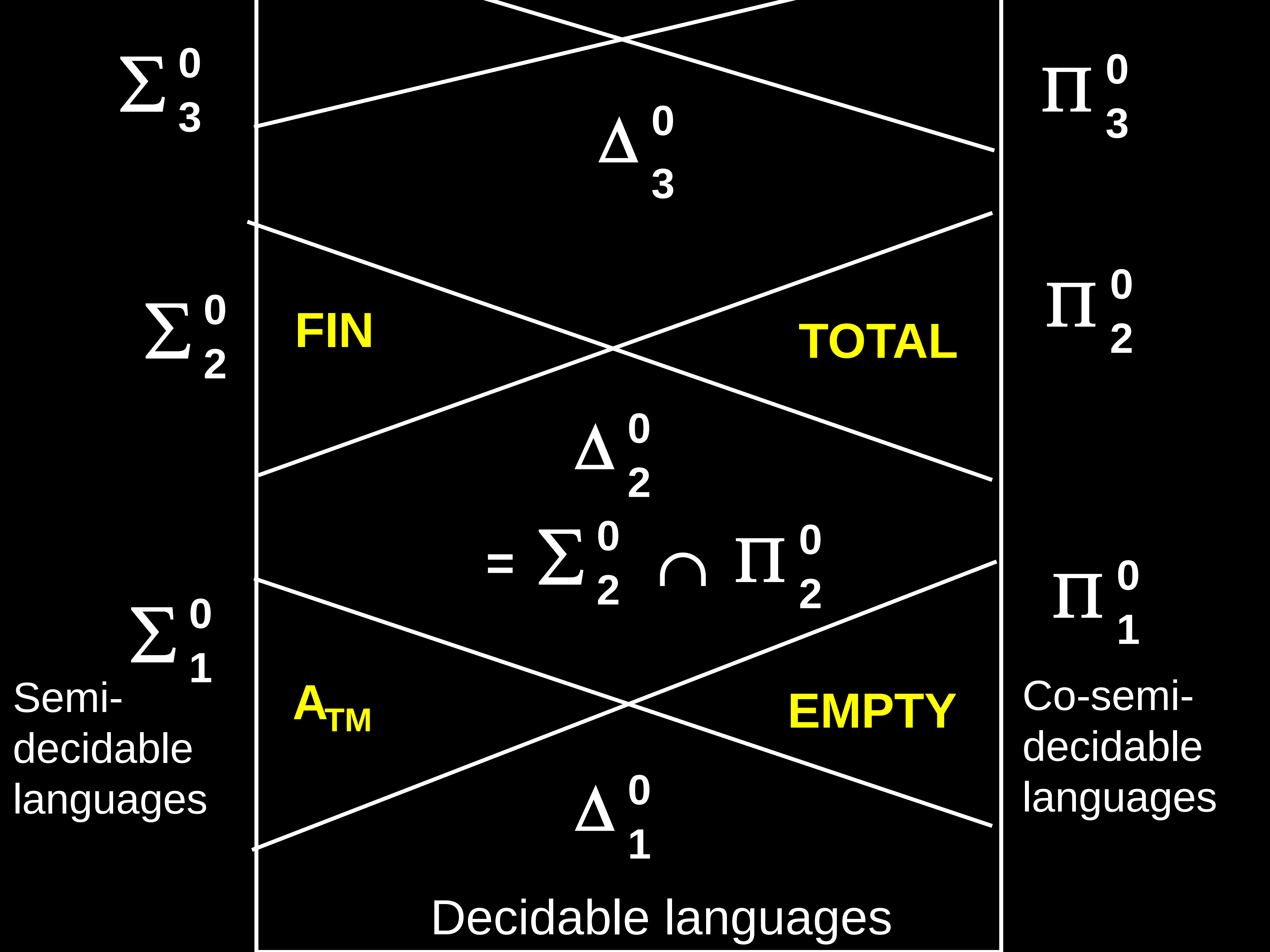
Σ^0_2 = languages of the form { x | ∃y∀z R(x,y,z) }

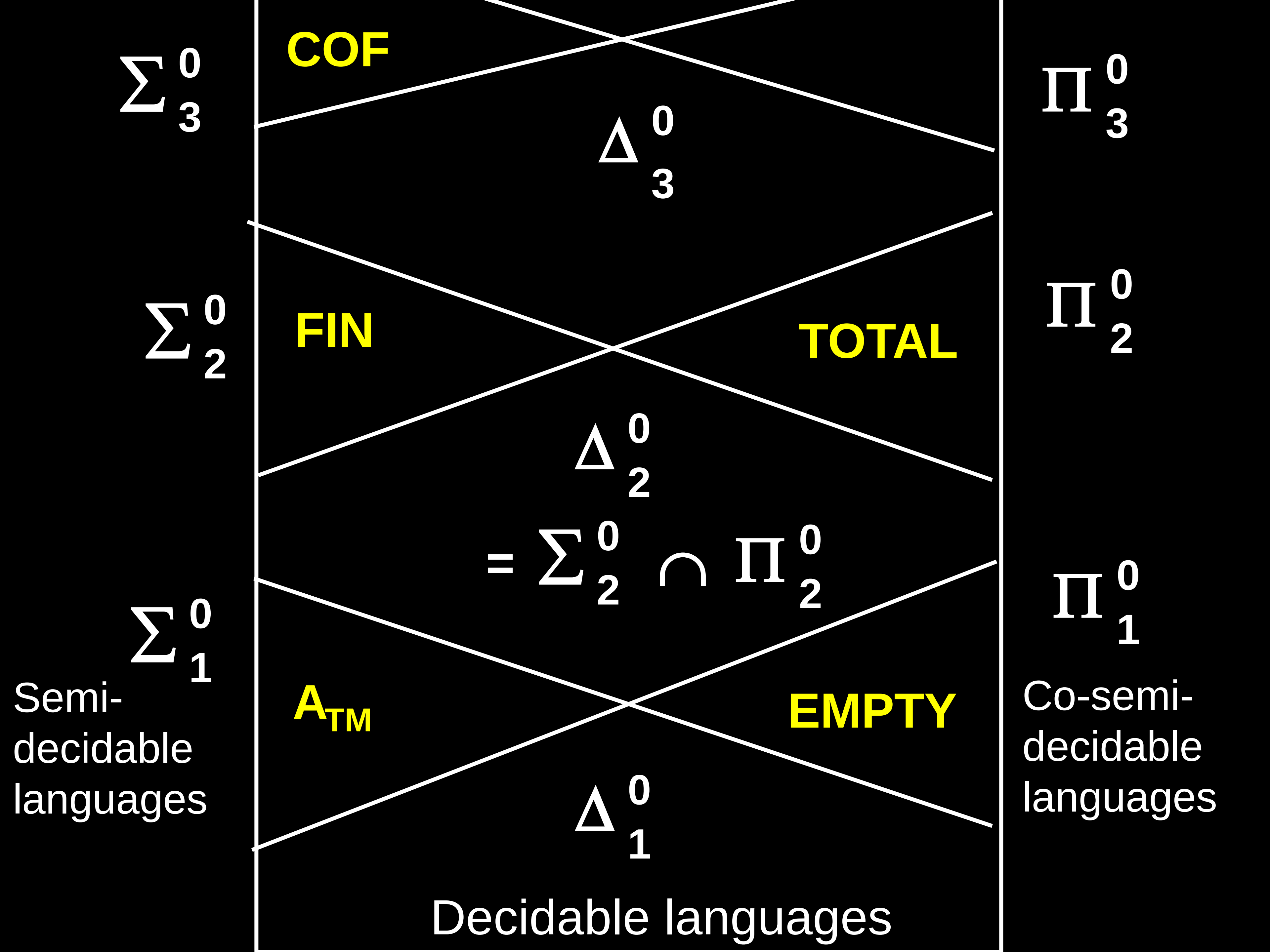
Show that FIN = { M | L(M) is finite } is in Σ^0_2

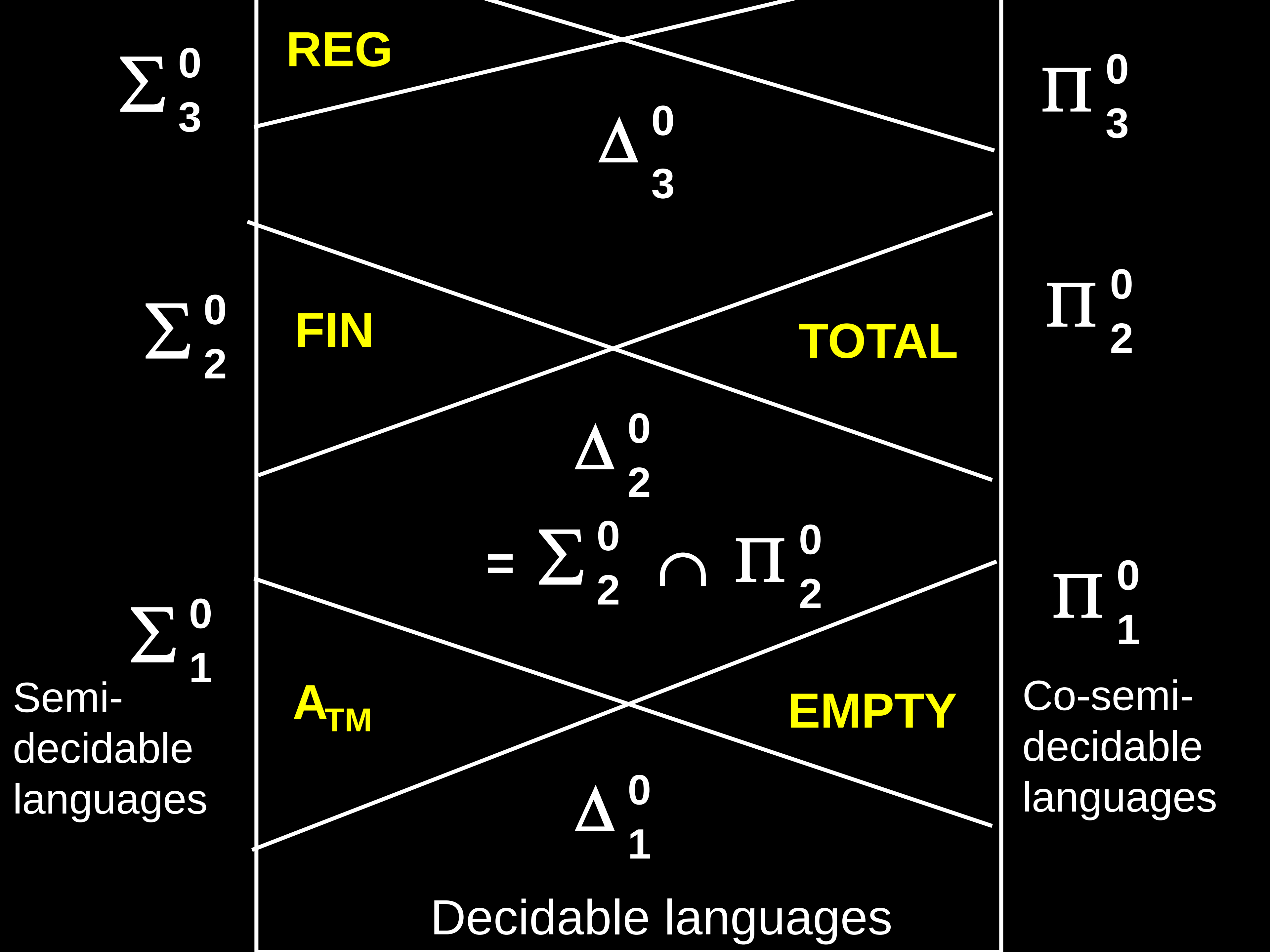
FIN = { M | ∃n∀w∀t [Either |w| < n, or M doesn't accept w in t steps] }

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

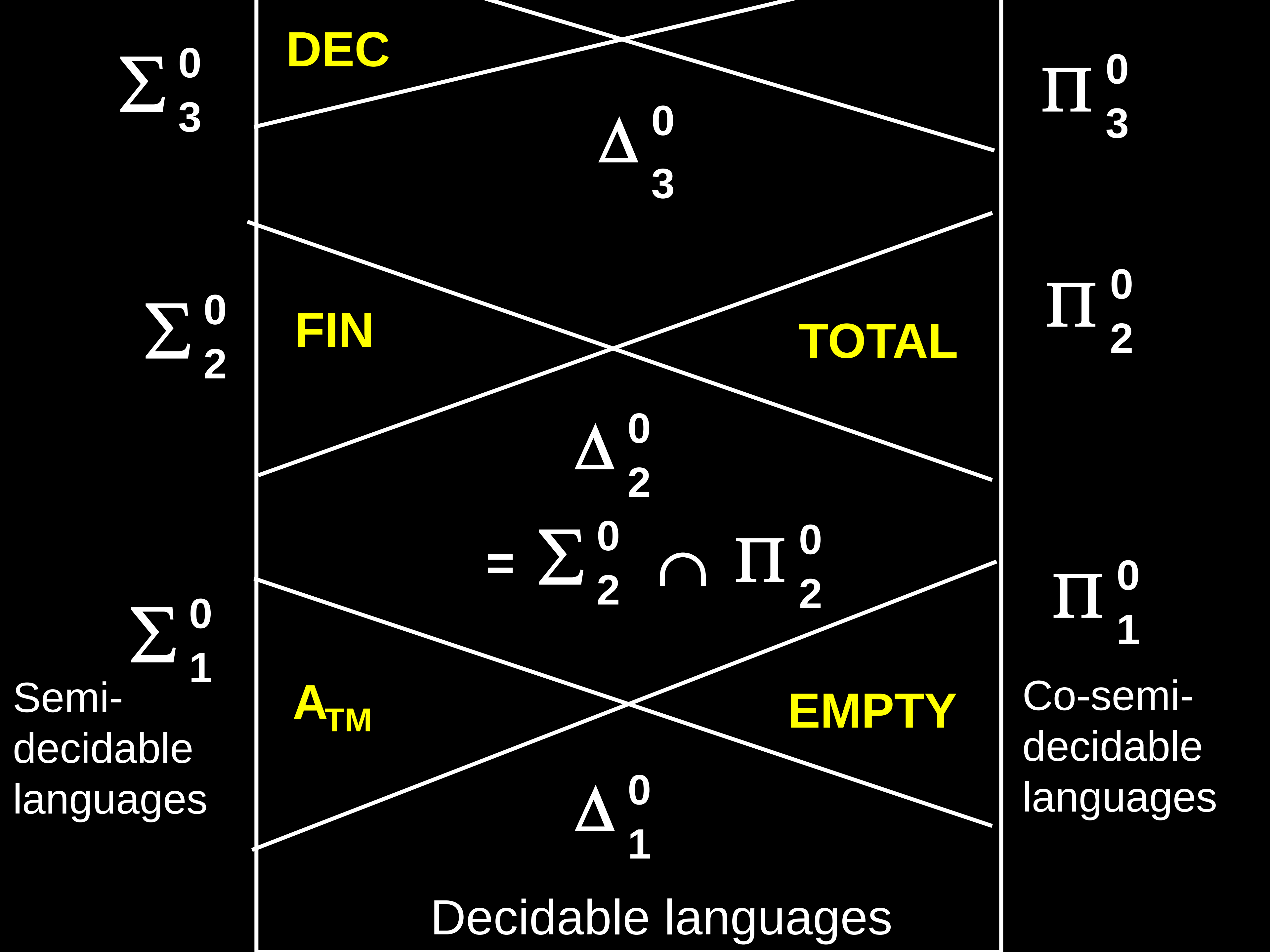
decidable predicate


Σ_3^0
Π_3^0
Δ_3^0
Σ_2^0
FIN
TOTAL
Π_2^0
Δ_2^0
$= \Sigma_2^0 \cap \Pi_2^0$
Σ_1^0
Π_1^0
Semi-decidable languages
A_{TM}
EMPTY
Co-semi-decidable languages
Δ_1^0
Decidable languages

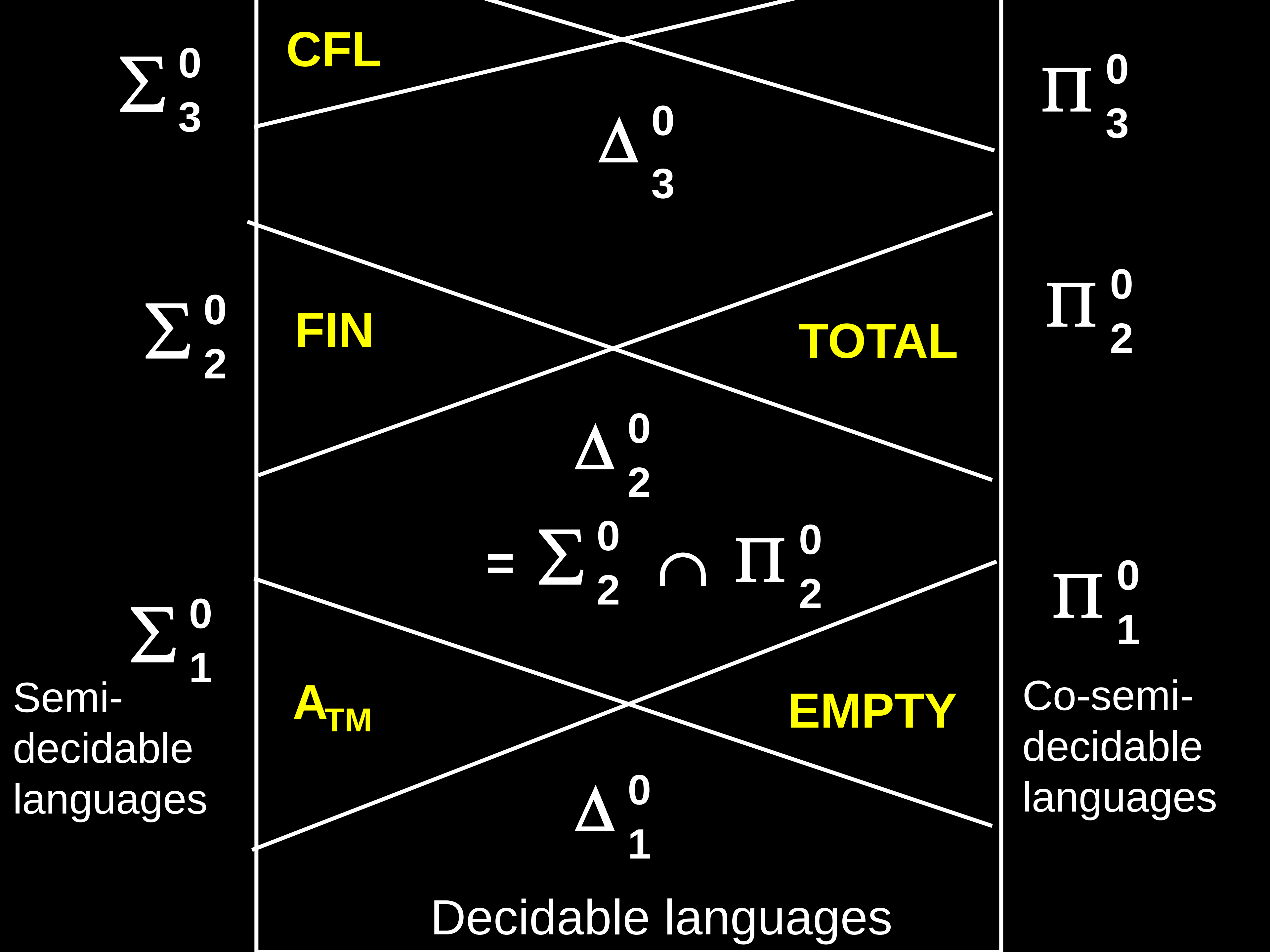

COF
Σ^0_3
Δ^0_3
Π^0_3
Σ^0_2
FIN
TOTAL
Π^0_2
Δ^0_2
$= \Sigma^0_2 \cap \Pi^0_2$
Σ^0_1
Π^0_1
Semi-decidable languages
A_{TM}
EMPTY
Co-semi-decidable languages
Δ^0_1
Decidable languages



REG
Σ_3^0
Δ_3^0
Π_3^0
Σ_2^0
FIN
TOTAL
Π_2^0
Δ_2^0
$= \Sigma_2^0 \cap \Pi_2^0$
Σ_1^0
Π_1^0
Semi-decidable languages
A_{TM}
EMPTY
Co-semi-decidable languages
Δ_1^0
Decidable languages



Σ^0_3
DEC
Π^0_3
Δ^0_3
Σ^0_2
FIN
TOTAL
Π^0_2
Δ^0_2
$= \Sigma^0_2 \cap \Pi^0_2$
Σ^0_1
Π^0_1
Semi-decidable languages
A_{TM}
EMPTY
Co-semi-decidable languages
Δ^0_1
Decidable languages



Σ^0_3
CFL
Δ^0_3
Π^0_3
Σ^0_2
FIN
TOTAL
Π^0_2
Δ^0_2
$= \Sigma^0_2 \cap \Pi^0_2$
Σ^0_1
Π^0_1
Semi-decidable languages
A_{TM}
EMPTY
Co-semi-decidable languages
Δ^0_1
Decidable languages

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 ⇔ H(D,D) accepts ⇔ 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 $A_{TM,1} = A_{TM}$

$A_{TM,\, n+1} = \{(M,x) \mid M$ 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

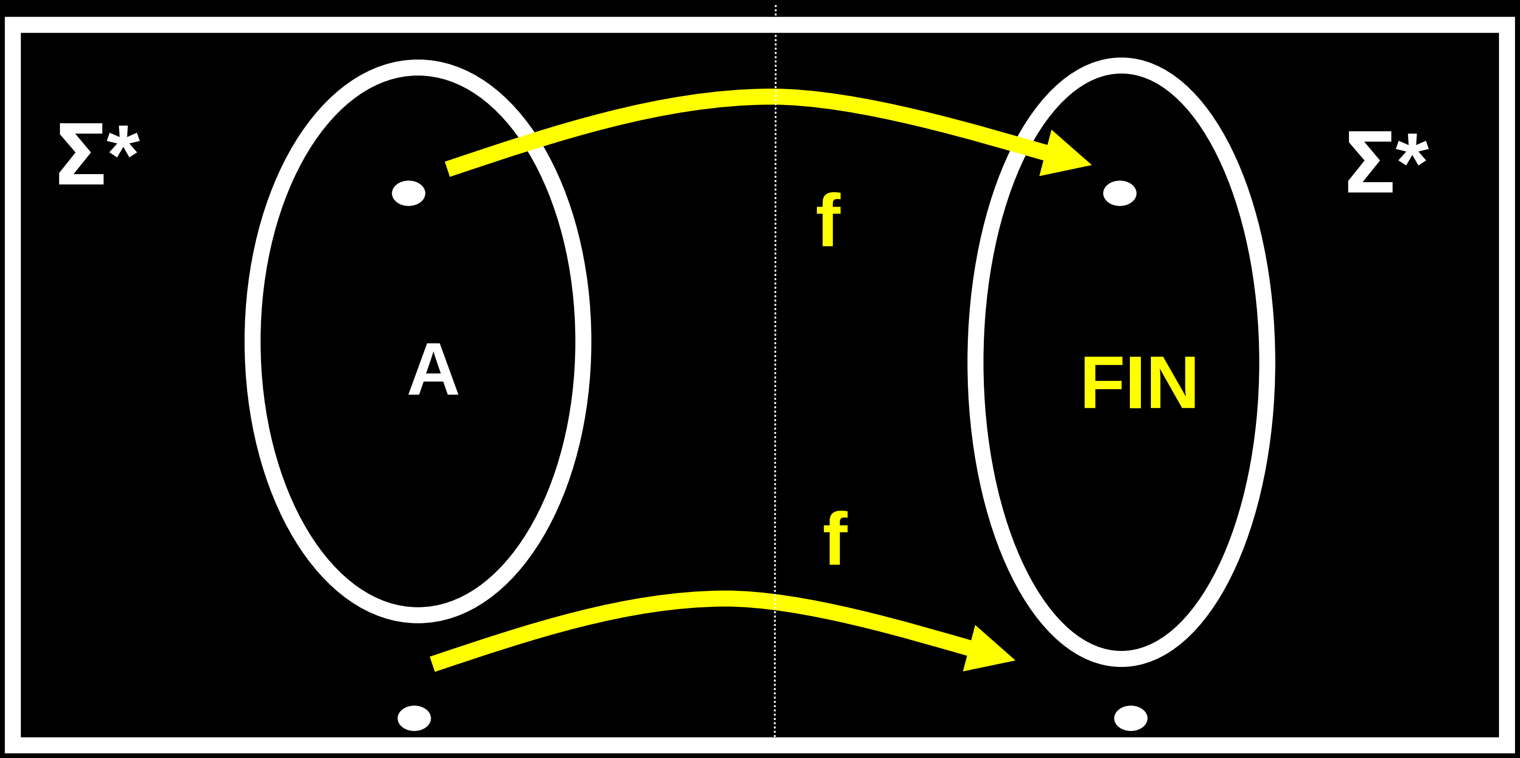
FIN = { M| ∃**n** ∀**x** ∀**t (|x| <n or M does not accept x in t steps)}**

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

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

For A $\in \Sigma^0_2$, A={ x | ∃y∀z R(x,y,z)}

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



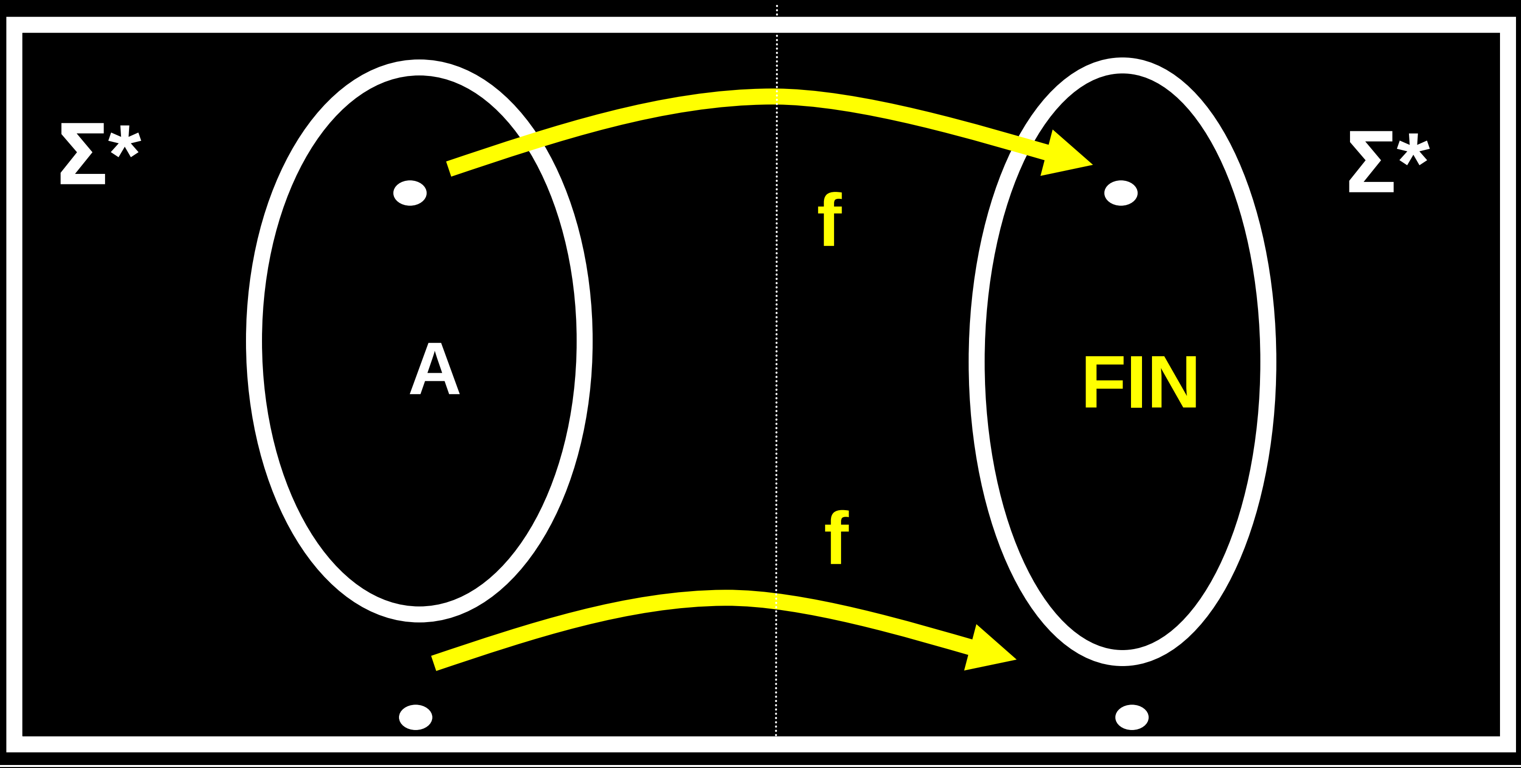
f: x → M_x

Given input w:

For each y of length |w| or less, look for z such that ¬ R(x,y,z)} . If found for all such y, Accept. Otherwise keep on running.

For A $\in \Sigma_2^0$, A={ x | ∃y∀z R(x,y,z)}

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



- **If x ∈ A, then ∃y∀z R(x,y,z)}** , so when |w| >| y|, **M_x** keeps on running, so **M_x∈ FIN**.
- If x ∉ A, **then ∀y∃z ¬ R(x,y,z)}** , so **M_x** recognizes **Σ***

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….

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

$$K(x) \leq |\langle M,x \rangle| \leq 2|M| + |x| + 1 \leq c + |x|$$

(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**,

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

"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) $\leq$ 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) $\leq$ n}

Theorem: COMPRESS is undecidable!

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. (y,n) $\notin$ 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>| $\leq$ c + log n

So n < K(y*) $\leq$ c + log n. CONTRADICTION!

DETERMINING COMPRESSIBILITY

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 $|\langle M,x\rangle| \leq c + \log n$

So $n < K(y^*) \leq c + \log n$. 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 $\Rightarrow$ 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, $e_i \leq \log n$, 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 $f : N \to N$, 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 $\Box \notin \Sigma$

Γ is the tape alphabet, where $\Box \in \Gamma$ and $\Sigma \subseteq \Gamma$

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

$q_0 \in Q$ is the start state

$q_{accept} \in Q$ 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 }

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

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

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

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

Proof:

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

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

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

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

$f : \Sigma^* \to \Sigma^*$ 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 $f : \Sigma^* \to \Sigma^*$ 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)

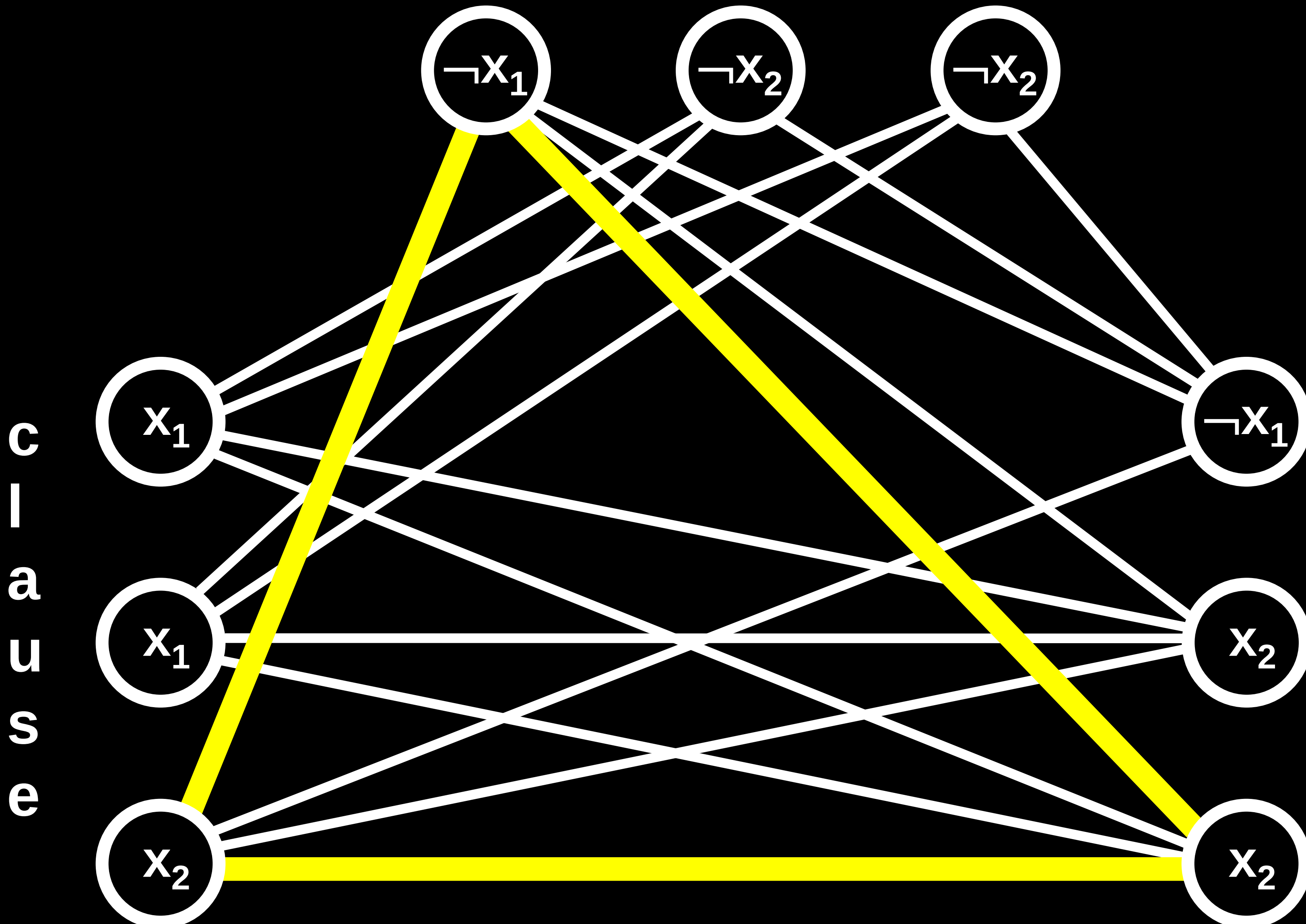
CLIQUE = { (G,k) | G is an undirected graph with a k-clique }

Theorem: CLIQUE is NP-Complete

(1) CLIQUE $\in$ NP

(2) 3SAT $\leq_P$ CLIQUE

$(x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2)$



#nodes = 3(# clauses)

k = #clauses

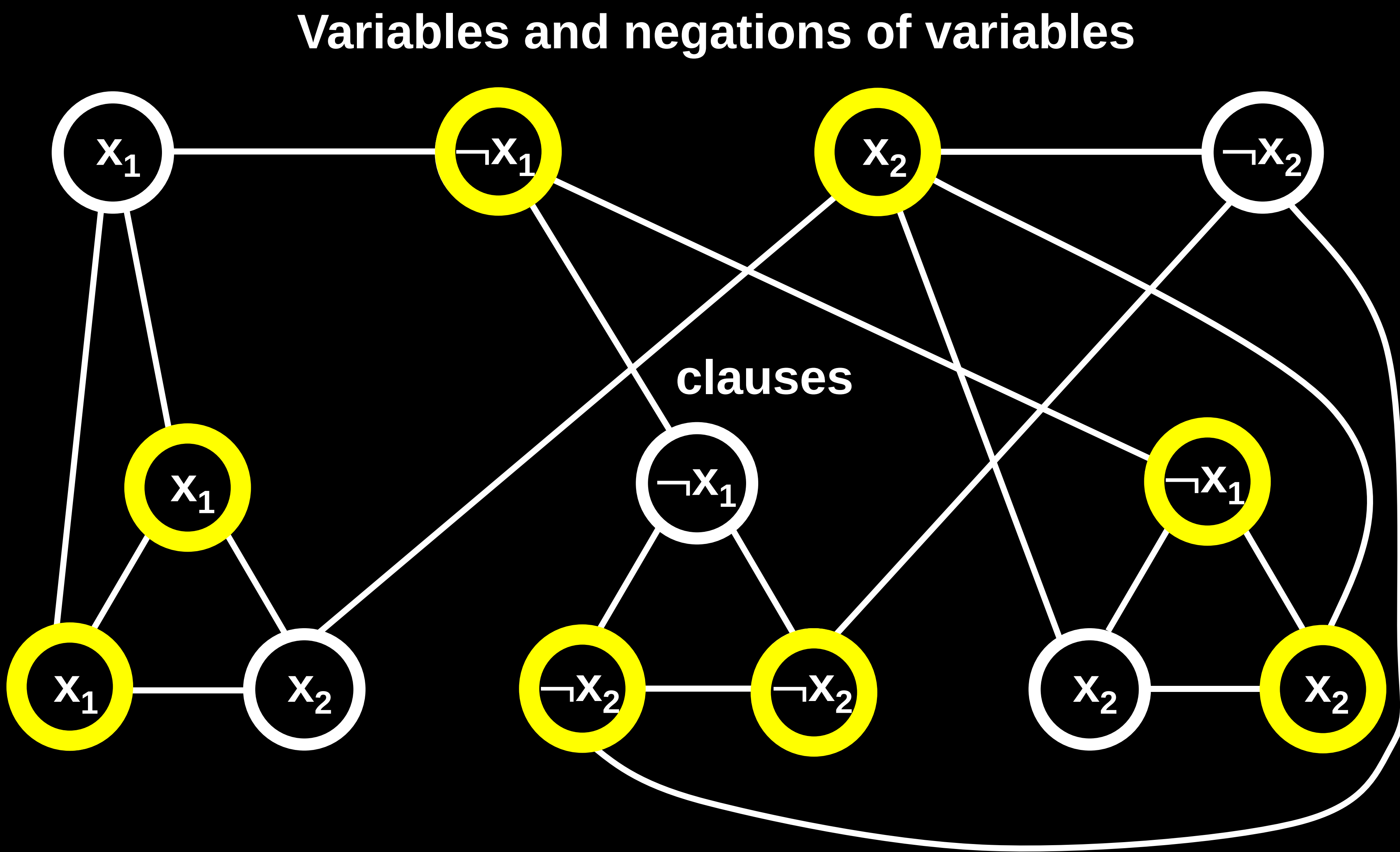
VERTEX-COVER = { (G,k) | G is an undirected graph with a k-node vertex cover }

Theorem: VERTEX-COVER is NP-Complete

(1) VERTEX-COVER $\in$ NP

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

$$(x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2)$$



k = 2(#clauses) + (#variables)

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

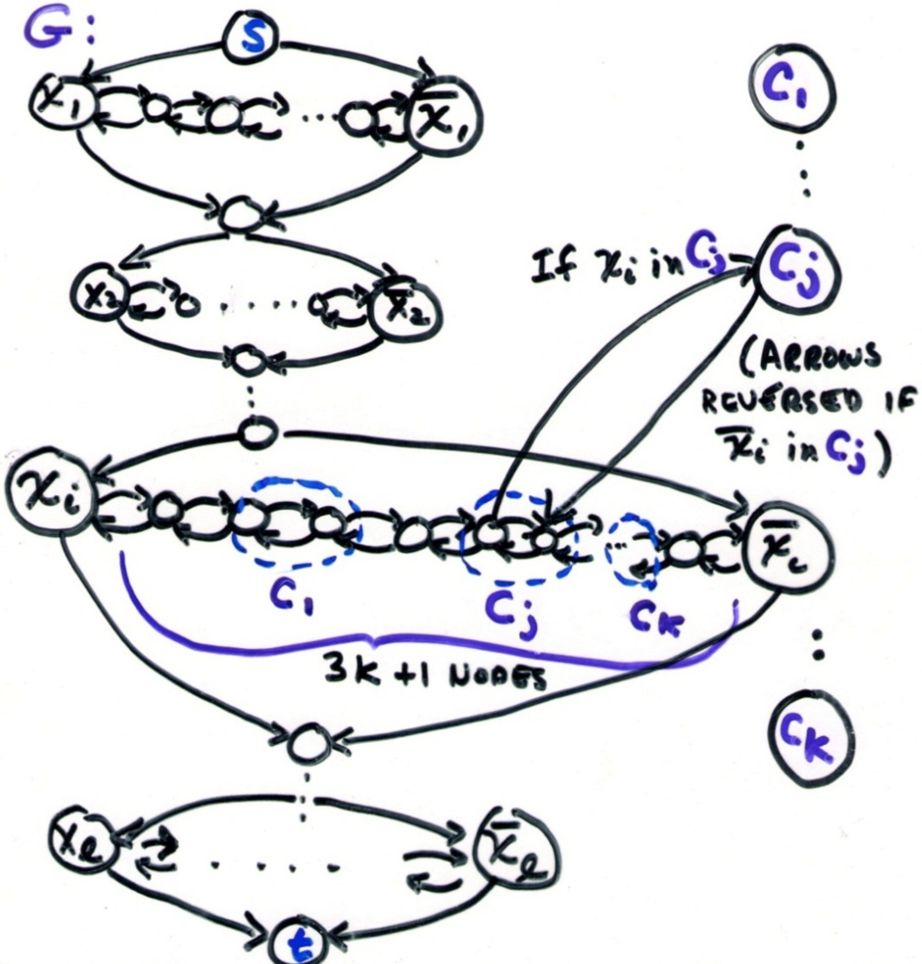
(2) 3SAT $\leq_P$ HAMPATH

Proof is in Sipser, Chapter 7.5

3 SAT $\leq_p$ HAM PATH

$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_j \wedge \cdots \wedge C_k$ C_j, CLAUSE

$x_1, \ldots, x_\ell$ VARIABLES



SUPPOSE φ SATISFIABLE WITH SOME TRUTH ASSIGNMENT. ZIG ZAG if x_i is TRUE, ZAG-ZIG IF $\overline{x}_i$ TRUE. DETOUR ON CLAUSES NOT ALREADY COVERED.

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

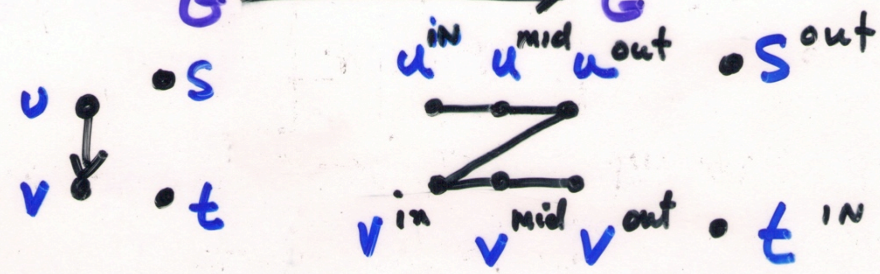
Theorem: UHAMPATH is NP-Complete

(1) UHAMPATH $\in$ NP

(2) HAMPATH $\leq_P$ UHAMPATH

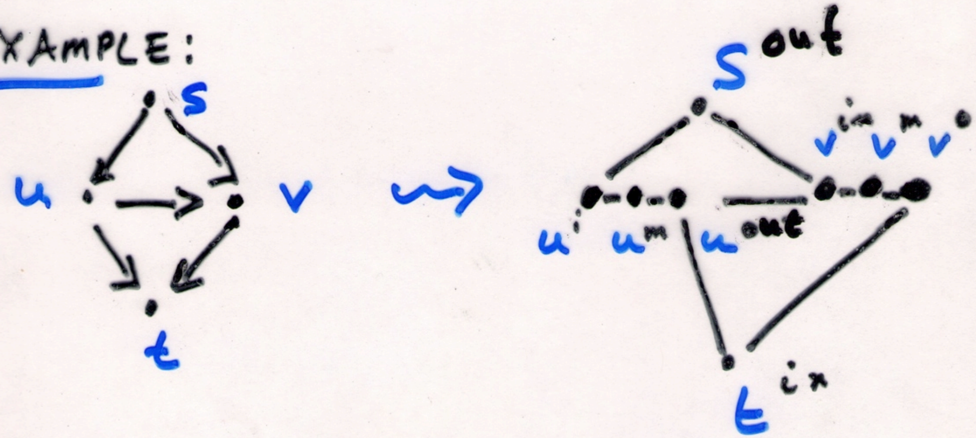
HAMPATH $\leq_p$ UHAMPATH

$G \longrightarrow G'$



Rule: $u \to v$ then u^{out} — v^{in}

Example:



- Why do we need mid ?

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

Theorem: SUBSETSUM is NP-Complete

(1) SUBSETSUM $\in$ NP

(2) 3SAT $\leq_P$ SUBSETSUM

$3SAT \leq_P$ SUBSET SUM

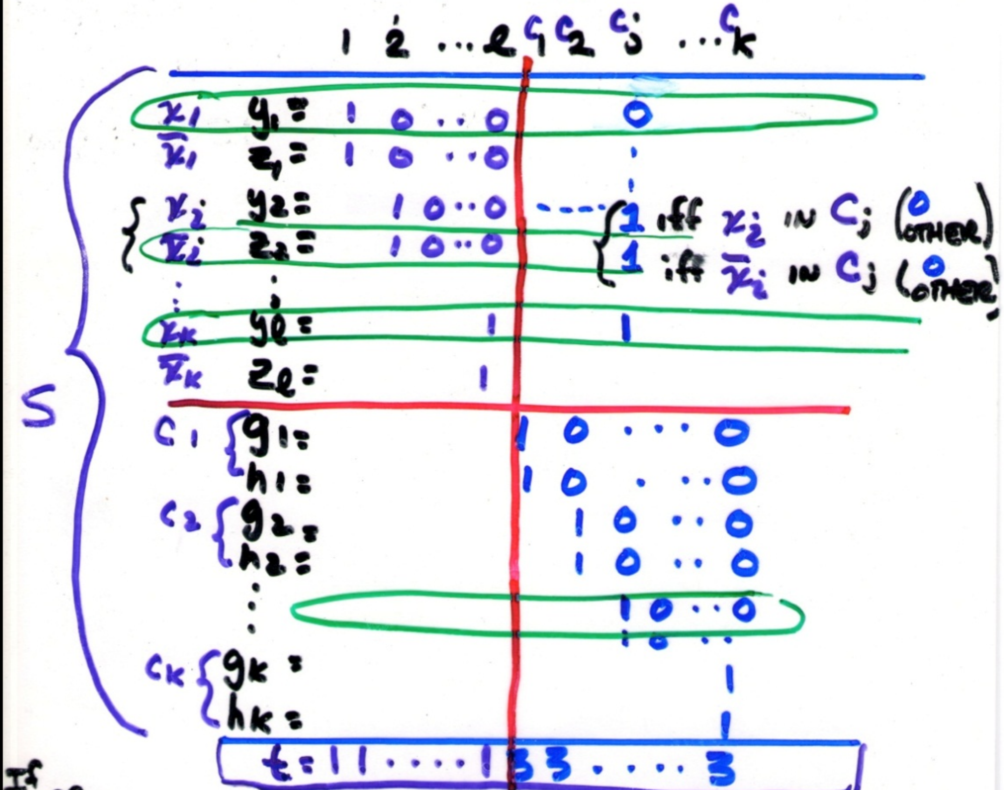
$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ $\quad C_j$, CLAUSE

VARIABLES: $x_1, \ldots, x_\ell$

↓

(S, t) $\quad S = \{ y_i, z_i, g_j, h_j \mid i = 1, \ldots, \ell;\ j = 1, \ldots, k \}$

$t = \underbrace{11\cdots1}_{\ell}\underbrace{33\cdots3}_{k}$



If φ SATISFIABLE WITH some truth assignment FOR SUBSET CHOOSE ROWS WITH LITERALS TRUE & g_j's & h_j's AS NECESSARY TO ADD UP.

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!