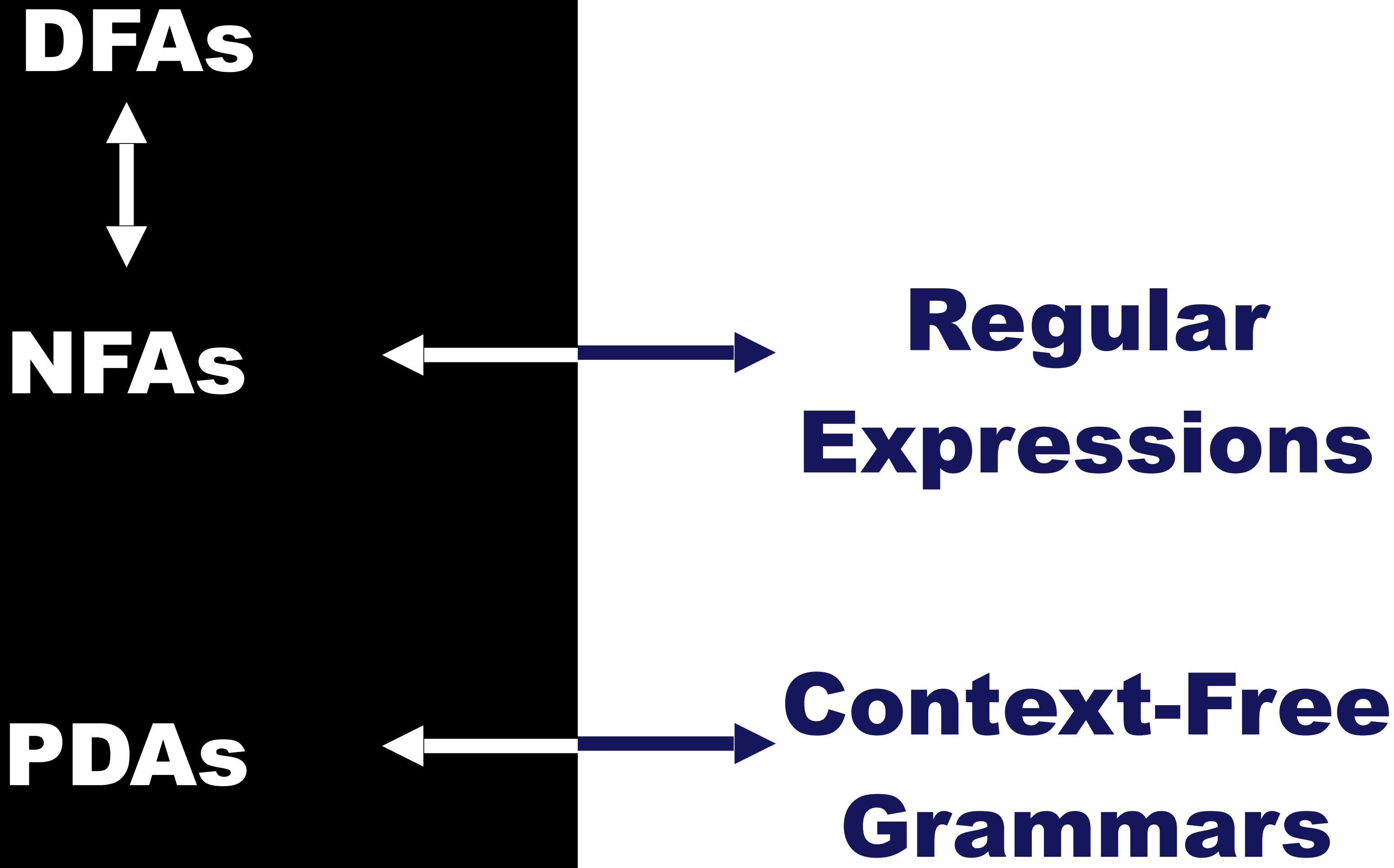
Machines

Syntactic Rules



DFAs

NFAs

PDAs

Regular Expressions

Context-Free Grammars

deterministic DFA

A ^ finite automaton ^ is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

Q is the set of states (finite)

Σ is the alphabet (finite)

$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept states

Let $w_1, \ldots, w_n \in \Sigma$ and **w** $= w_1 \ldots w_n \in \Sigma^*$

Then **M accepts w** if there are $r_0, r_1, \ldots, r_n \in Q$, s.t.

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for i = 0, ..., n-1, and
3. $r_n \in F$

Let $w \in \Sigma^*$ and suppose w can be written as $w_1 \dots w_n$ where $w_i \in \Sigma_\varepsilon$ (ε = empty string)

Then N **accepts** w if there are $r_0, r_1, \dots, r_n \in Q$ such that

1. $r_0 \in Q_0$
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for i = 0, ..., n-1, and
3. $r_n \in F$

L(N) = the language recognized by N
= set of all strings machine N accepts

A language L is **recognized** by an NFA N if L = L(N).

Let $w \in \Sigma^*$ and suppose w can be written as $w_1 \ldots w_n$ where $w_i \in \Sigma_\varepsilon$ (recall $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$)

Then P **accepts** w if there are

$r_0, r_1, \ldots, r_n \in Q$ and

$s_0, s_1, \ldots, s_n \in \Gamma^*$ (sequence of stacks) such that

1. $r_0 = q_0$ and $s_0 = \varepsilon$ (P starts in q_0 with empty stack)

2. For $i = 0, \ldots, n-1$:
$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$
(P moves correctly according to state, stack and symbol read)

3. $r_n \in F$ (P is in an accept state at the end of its input)

THEOREM

For every regular language L, there exists a UNIQUE (up to re-labeling of the states) minimal DFA M such that L = L(M)

EXTENDING δ

Given DFA **M** = $(Q, \Sigma, \delta, q_0, F)$, extend δ

to $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ as follows:

$$\hat{\delta}(q, \varepsilon) = q$$

$$\hat{\delta}(q, \sigma) = \delta(q, \sigma)$$

$$\hat{\delta}(q, w_1 \ldots w_{k+1}) = \delta(\hat{\delta}(q, w_1 \ldots w_k), w_{k+1})$$

Note: $\delta(q_0, w) \in F \Leftrightarrow$ M accepts w

String $w \in \Sigma^*$ **distinguishes** states q_1 and q_2 iff exactly ONE of $\hat{\delta}(q_1, w)$, $\hat{\delta}(q_2, w)$ is a final state

Fix $M = (Q, \Sigma, \delta, q_0, F)$ and let $p, q, r \in Q$

Definition:

$p \sim q$ iff p is indistinguishable from q

$p \nsim q$ iff p is distinguishable from q

Proposition: $\sim$ is an equivalence relation

$p \sim p$ (reflexive)

$p \sim q \Rightarrow q \sim p$ (symmetric)

$p \sim q$ and $q \sim r \Rightarrow p \sim r$ (transitive)

Proposition: ~ is an **equivalence relation**

so ~ partitions the set of states of M into disjoint equivalence classes

$[q] = \{ p \mid p \sim q \}$

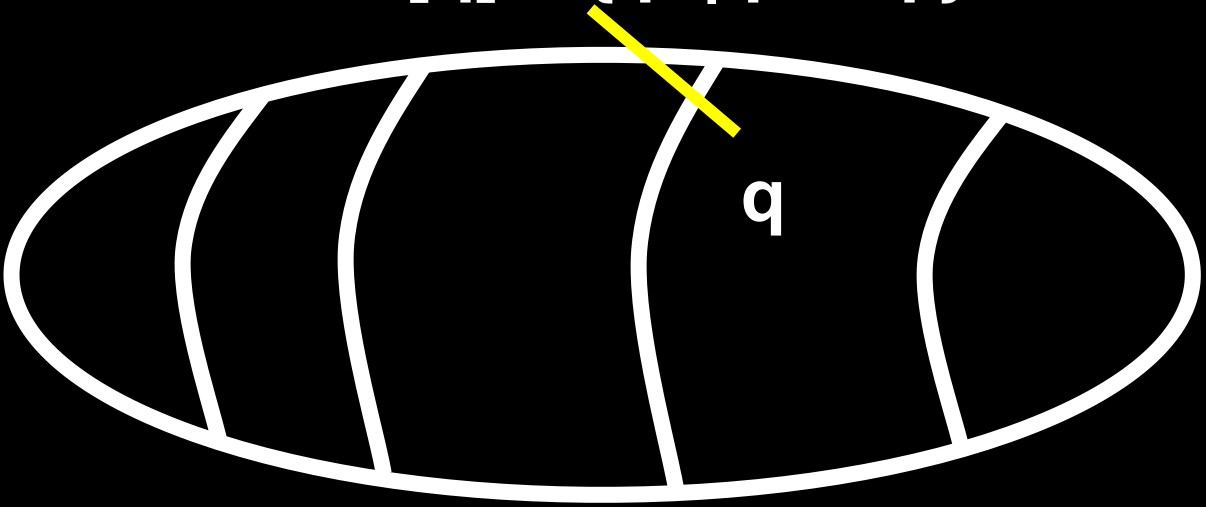
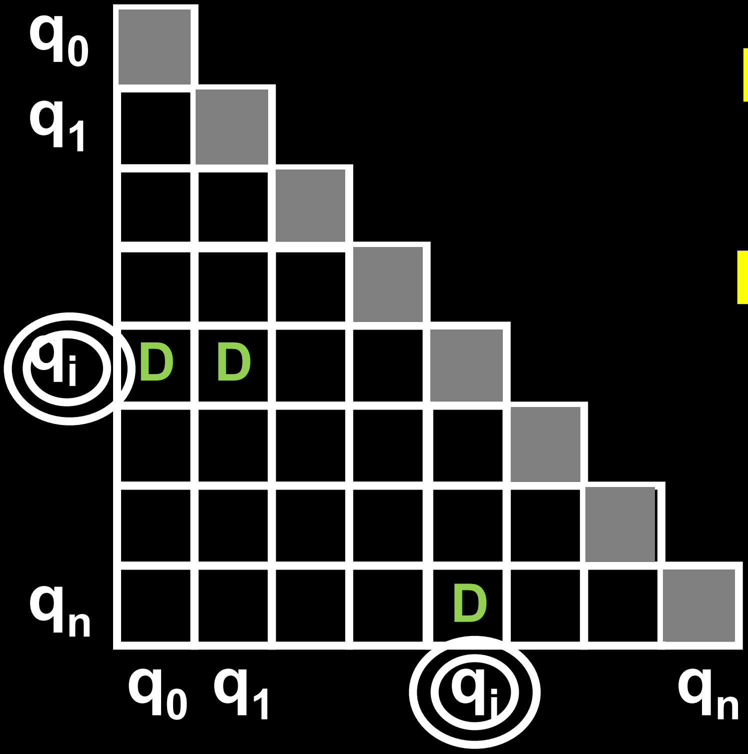


TABLE-FILLING ALGORITHM

Input: DFA $M = (Q, \Sigma, \delta, q_0, F)$

Output: (1) $D_M = \{ (p,q) \mid p,q \in Q \text{ and } p \not\sim q \}$

(2) $E_M = \{ [q] \mid q \in Q \}$



Base Case: p accepts and q rejects $\Rightarrow p \not\sim q$

Recursion: if there is $\sigma \in \Sigma$ and states p', q' satisfying

$$\begin{array}{l} \delta(p, \sigma) = p' \\ \qquad\qquad\ \ \not\sim \quad \Rightarrow \quad p \not\sim q \\ \delta(q, \sigma) = q' \end{array}$$

Repeat until no more new **D**'s

CONVERTING NFAs TO DFAs

Input: NFA $N = (Q, \Sigma, \delta, Q_0, F)$

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$

$$Q' = 2^Q$$

$$\delta' : Q' \times \Sigma \to Q'$$

$$\delta'(R,\sigma) = \bigcup_{r \in R} \varepsilon(\, \delta(r,\sigma)\,) \quad *$$

$$q_0' = \varepsilon(Q_0)$$

$$F' = \{\, R \in Q' \mid f \in R \text{ for some } f \in F \,\}$$

* For **R** $\subseteq$ **Q,** the **ε-closure** **of R, ε(R)** = **{q** that can be reached from some r $\in$ **R** by traveling along zero or more **ε** arrows}

THE REGULAR OPERATIONS

Union: $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$

Intersection: $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$

Negation: $\neg A = \{ w \in \Sigma^* \mid w \notin A \}$

Reverse: $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \}$

Concatenation: $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

Star: $A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \}$

REGULAR **EXPRESSIONS**

σ **is a regexp representing $\{\sigma\}$**

ε **is a regexp representing $\{\varepsilon\}$**

$\varnothing$ **is a regexp representing $\varnothing$**

If R_1 and R_2 are regular expressions representing L_1 and L_2 then:

(R_1R_2) represents $L_1 \cdot L_2$

$(R_1 \cup R_2)$ represents $L_1 \cup L_2$

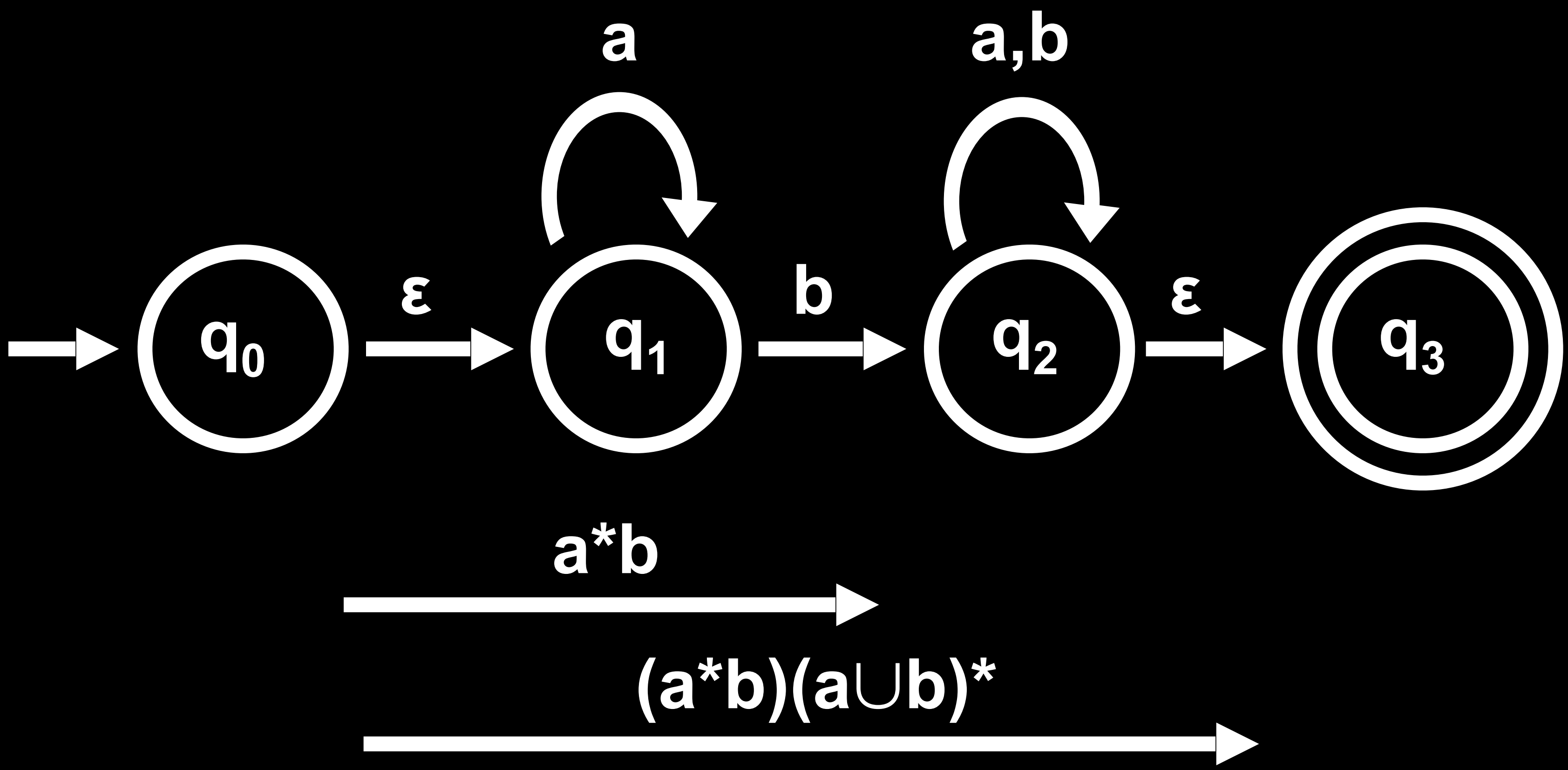
$(R_1)^*$ represents L_1^*

EQUIVALENCE

L can be represented by a regexp

⇔

L is a regular language



$R(q_0,q_3) = (a^*b)(a \cup b)^*$

How can we test if two regular expressions are the same?

$$\begin{tikzcd}
\text{Length n} & R_1 \arrow[d] & & R_2 \arrow[d] \\
\text{O(n) states} & N_1 \arrow[d] & & N_2 \arrow[d] \\
\text{O}(2^n)\text{ states} & M_1 \arrow[d] & & M_2 \arrow[d] \\
 & M_{1\ \mathrm{MIN}} & ?= & M_{2\ \mathrm{MIN}}
\end{tikzcd}$$

CONTEXT-FREE LANGUAGES

A context-free grammar (CFG) is a tuple $G = (V, \Sigma, R, S)$, where:

V is a finite set of variables

Σ is a finite set of terminals (disjoint from V)

R is set of production rules of the form $A \rightarrow W$, where $A \in V$ and $W \in (V \cup \Sigma)^*$

$S \in V$ is the start variable

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$ Strings Generated by G

A Language L is context-free if there is a CFG that generates precisely the strings in L

CHOMSKY NORMAL FORM

A context-free grammar is in Chomsky normal form if every rule is of the form:

$A \to BC$	**B and C aren't start variables**
$A \to a$	**a is a terminal**
$S \to \varepsilon$	**S is the start variable**

Any variable A that is not the start variable can only generate strings of length > 0

Theorem: Any context-free language can be generated by a context-free grammar in Chomsky normal form

Theorem: If G is in CNF, $w \in L(G)$ and $|w| > 0$, then any derivation of w in G has length $2|w| - 1$

Theorem: There is an O(n^3 + size G) membership algorithm (CYK) any Chomsky normal form G.

Theorem: The set of PDAS that accept all strings is not r.e.

Definition: A (non-deterministic) PDA is a tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where:

Q is a finite set of states

Σ is the input alphabet

Γ is the stack alphabet

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow 2^{Q \times \Gamma_\varepsilon}$

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept states

2^Q is the set of subsets of Q and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

A Language L is generated by a CFG

$\Leftrightarrow$

L is recognized by a PDA

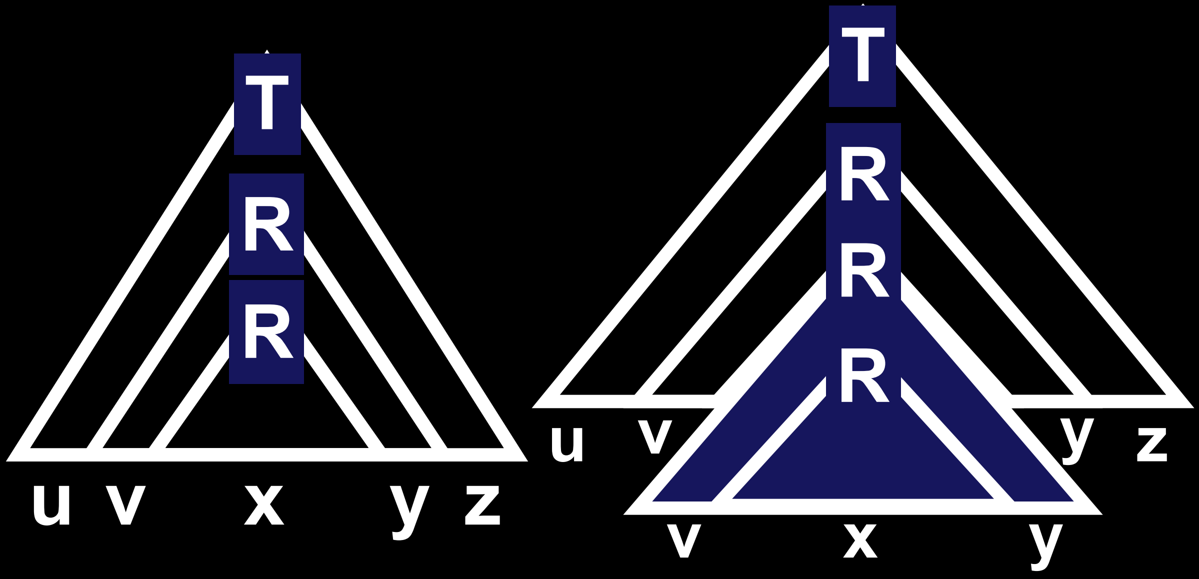
THE PUMPING LEMMA

(for Context Free Grammars)

Let L be a context-free language with $|L| = \infty$

Then **there is an integer P** such that
if $w \in L$ and $|w| \geq P$

then can write $w = uvxyz$, where:

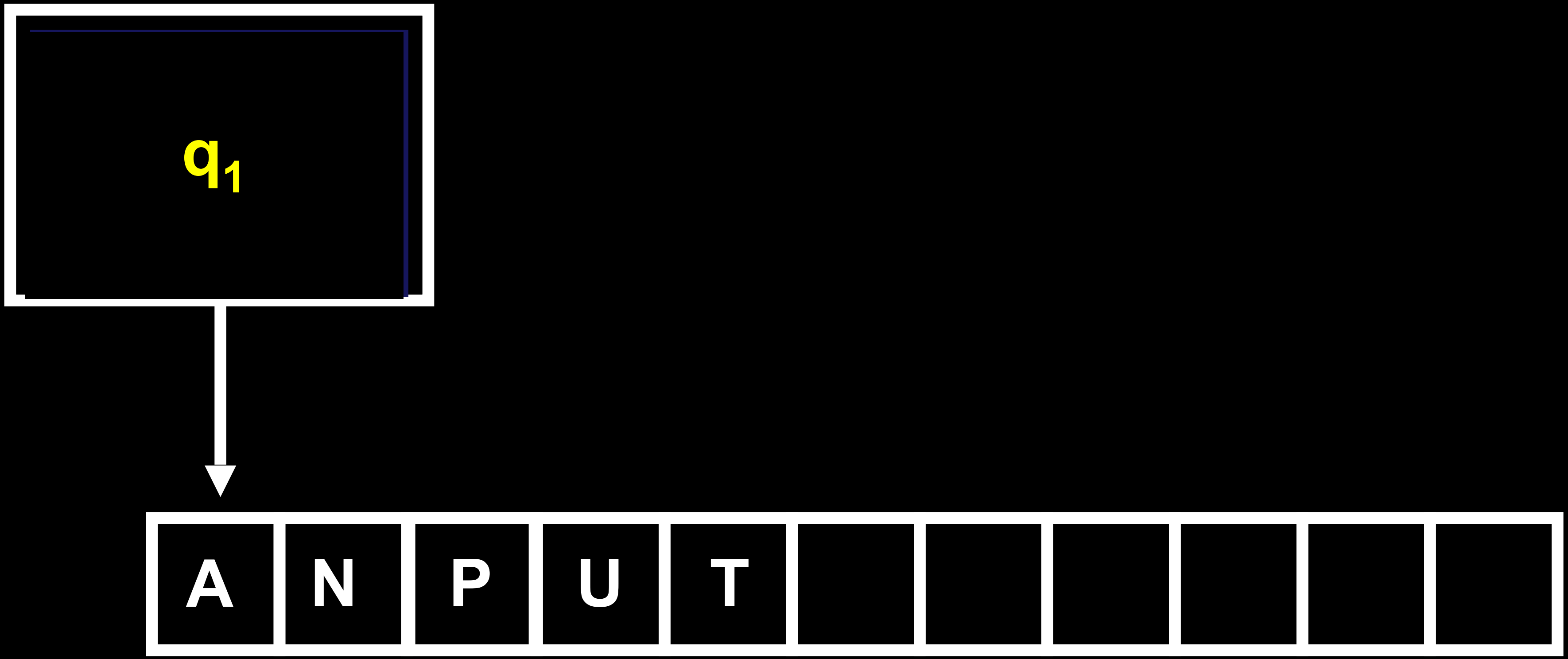


1. $|vy| > 0$

2. $|vxy| \leq P$

3. $uv^ixy^iz \in L$, for any $i \geq 0$

TURING MACHINE



q_1

A N P U T

INFINITE TAPE

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

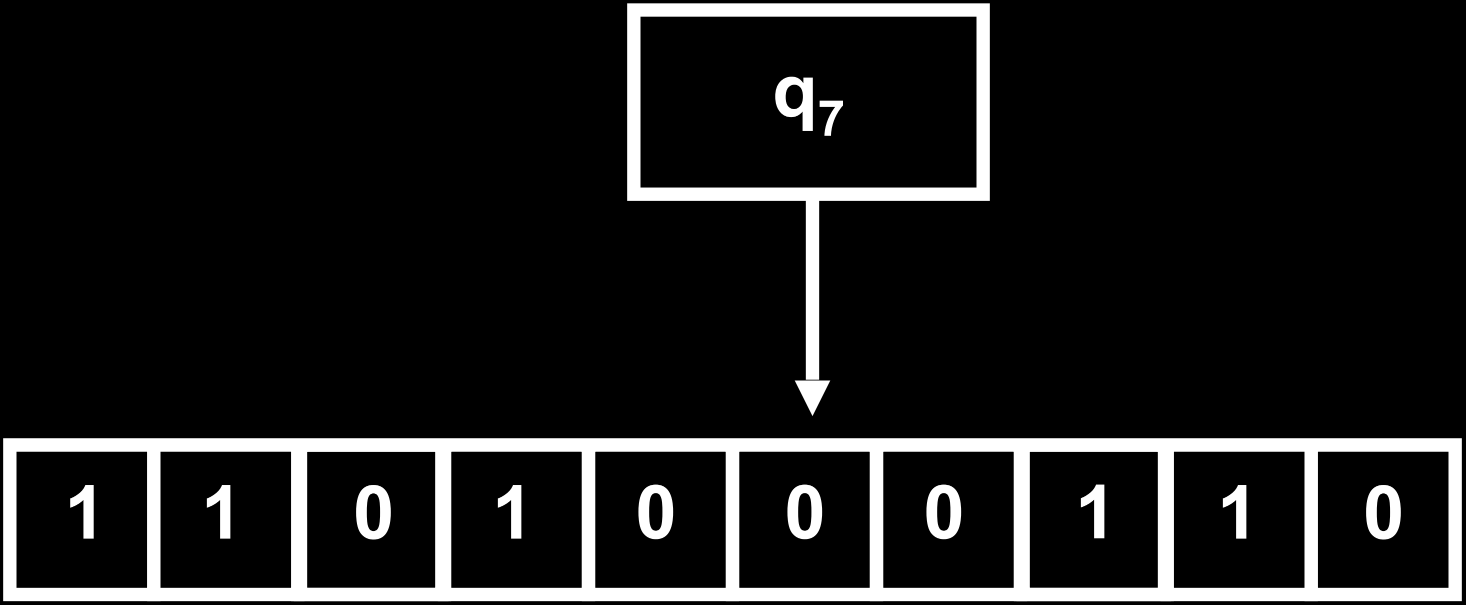
$q_{accept} \in Q$ is the accept state

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

CONFIGURATIONS

$$11010q_700110$$

corresponds to:



q_7

1	1	0	1	0	0	0	1	1	0

A Turing Machine M accepts input w if there is a sequence of configurations $C_1, \ldots , C_k$ such that

1. C_1 is a *start* configuration of M on input w, ie C_1 is q_0w
2. each C_i *yields* C_{i+1}, ie M can legally go from C_i to C_{i+1} in a single step

ua q_i bv *yields* u q_j acv if δ (q_i, b) = (q_j, c, L)
ua qi bv *yields* uac q_j v if δ (q_i, b) = (q_j, c, R)

A Turing Machine M accepts input w if there is a sequence of configurations $C_1, \dots, C_k$ such that

1. C_1 is a *start* configuration of M on input w, ie C_1 is q_0w
2. each C_i *yields* C_{i+1}, ie M can legally go from C_i to C_{i+1} in a single step
3. C_k is an *accepting* configuration, ie the state of the configuration is q_{accept}

A Turing Machine M *rejects* input w if there is a sequence of configurations $C_1, \ldots, C_k$ such that

1. **C_1 is a *start* configuration of M on input w, ie C_1 is q_0w**
2. **each C_i *yields* C_{i+1}, ie M can legally go from C_i to C_{i+1} in a single step**
3. **C_k is a *rejecting* configuration, ie the state of the configuration is q_{reject}**

A TM **decides** a language if it accepts all strings in the language and rejects all strings not in the language

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

Theorem: L decidable <-> ¬L decidable

Proof: L has a machine M that accepts or rejects on all inputs. Define M’ to be M with accept and reject states swapped. M’ decides ¬L.

A TM recognizes a language if it accepts all and only those strings in the language

A language is called Turing-recognizable or recursively enumerable, (or r.e. or semi-decidable) if some TM recognizes it

A TM decides a language if it accepts all strings in the language and rejects all strings not in the language

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

A TM recognizes a language if it accepts all and only those strings in the language

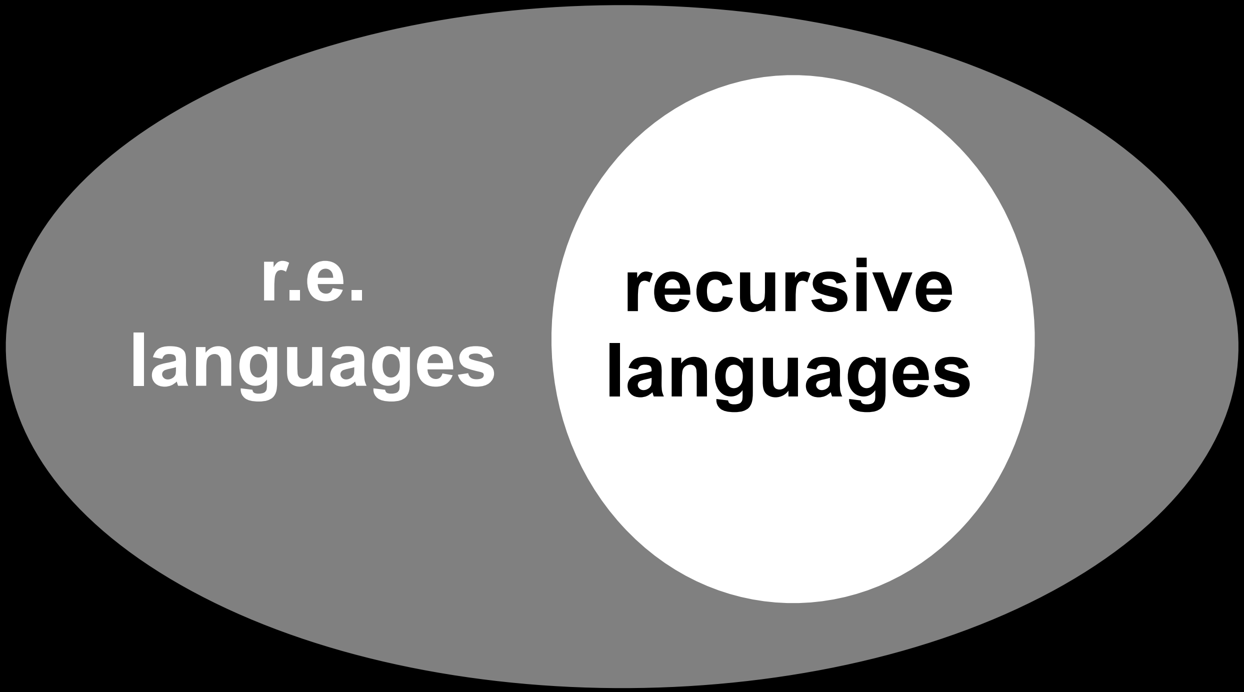
A language is called Turing-recognizable or recursively enumerable, (or r.e. or semi-decidable) if some TM recognizes it

FALSE: L r.e. <-> ¬L r.e.

Proof: L has a machine M that accepts or rejects on all inputs. Define M' to be M with accept and reject states swapped. M' decides ¬L.

A language is called **Turing-recognizable** or **recursively enumerable (r.e.)** or **semi-decidable** if some TM **recognizes** it

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

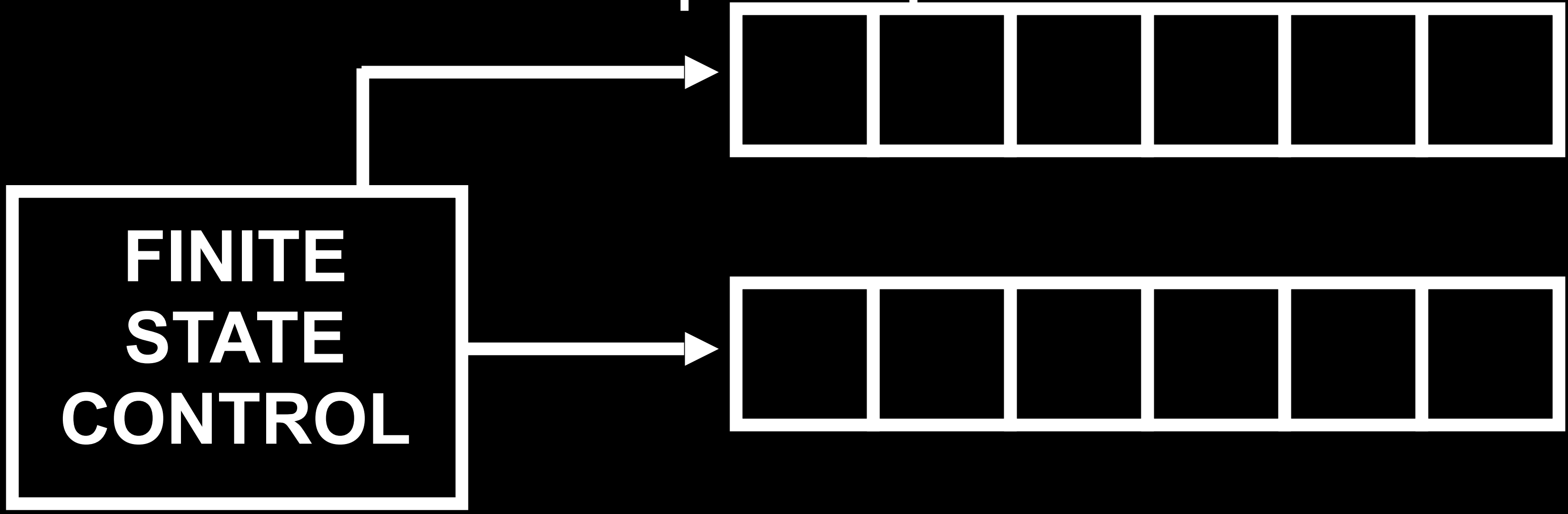


Theorem: If A and ¬A are r.e. then A is recursive

Theorem: If A and ¬A are r.e. then A is recursive

Suppose M accepts A. M' accepts ¬A decidable
Use Odd squares/ Even squares simulation of M and M'. If x is accepted by the even squares reject it/ accepted by the odd squares then accept x.

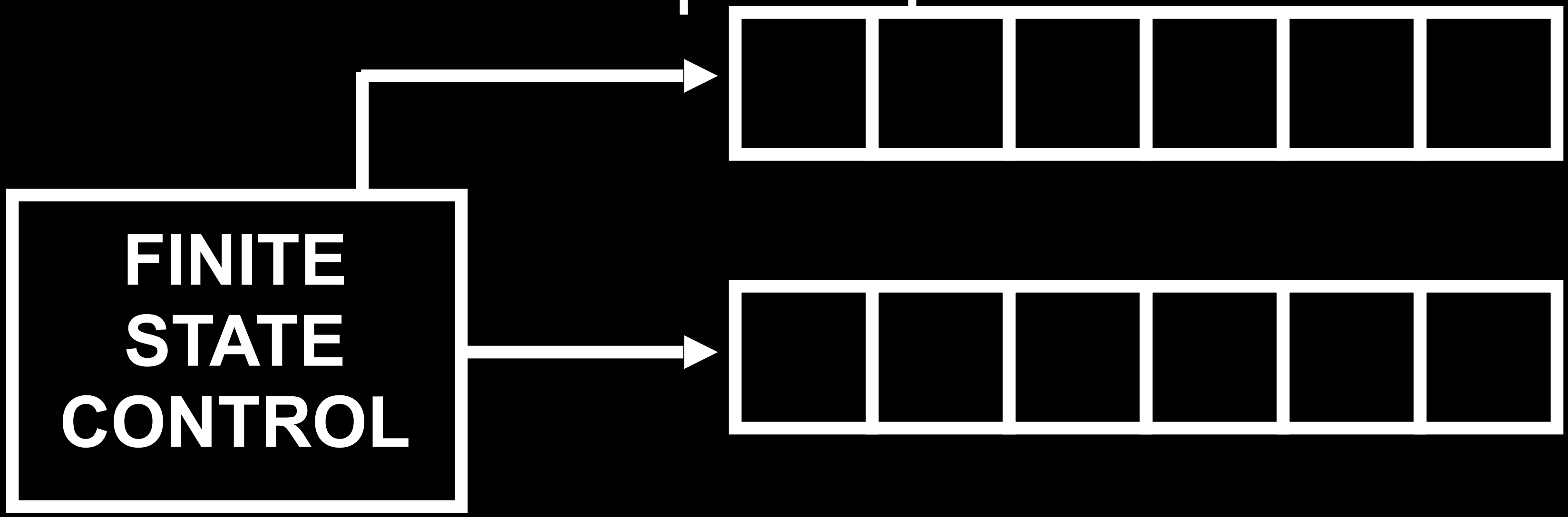
TURING MACHINE with WRITE ONLY output tape.



FINITE STATE CONTROL

Outputs a sequence of strings separated by hash marks. Allows for a well defined infinite sequence of strings in the limit. The machine is said to enumerate the sequence of strings occurring on the

TURING MACHINE with WRITE ONLY output tape.



Outputs a sequence of strings separated by hash marks. Allows for a well defined infinite sequence of strings in the limit. The machine is said to enumerate the set of strings occurring on the tape.

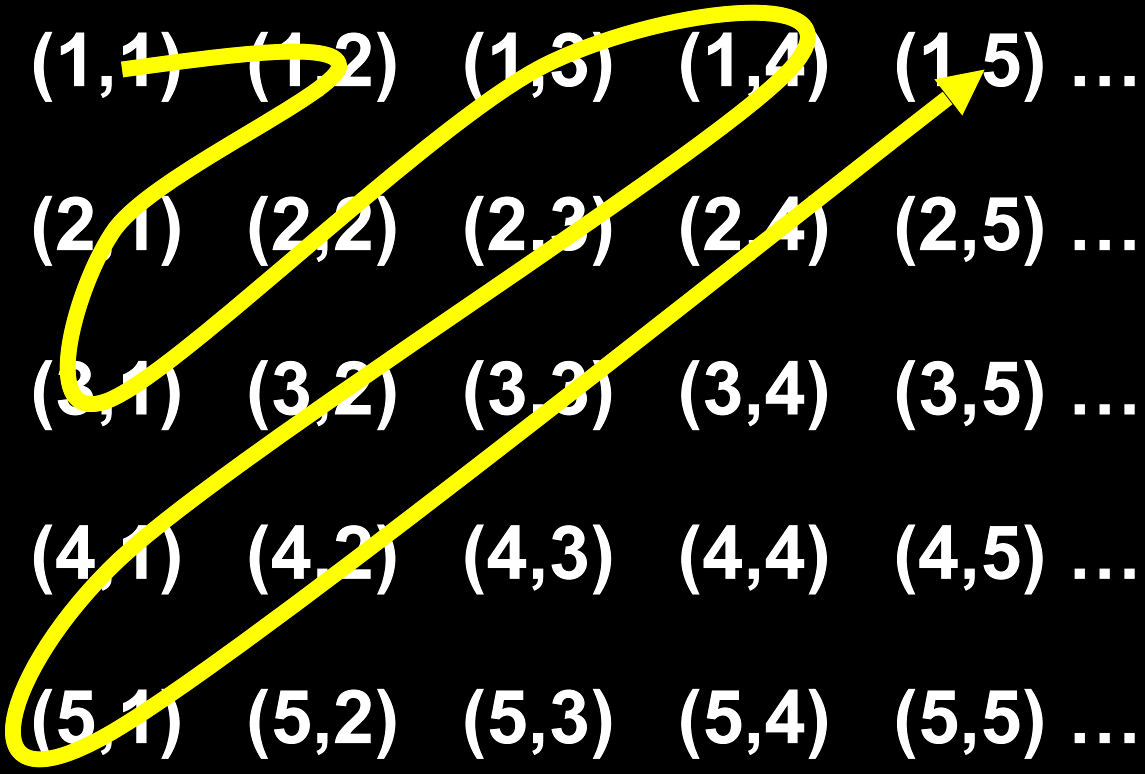
From every TM M accepting A. there is a TM M' outputting A.

```
For n = 0 to forever do
{       {Do n parallel simulations of M for n steps for the first n inputs}
M(0). M(1), M(2), M(3)..
}
```

From every TM M outputting A.
there is a TM M' accepting A.

M"(X) run M, accept if X output on tape.

Let $Z^+ = \{1,2,3,4\ldots\}$. There exists a bijection between Z^+ and $Z^+ \times Z^+$ (or Q^+)



(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	…
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	…
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	…
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	…
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	…

Lex-order has an enumerator strings of length 1, the length 2, ….

Pairs of binary strings have a lex-order enumerator

for each n>0 list all pairs of strings a,b as #a#b# where total length of a and b is n.

Let BINARY(w) = pair of binary strings be any fixed way of encoding a pair of binary strings with a single binary string

THE ACCEPTANCE PROBLEM

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

Theorem: A_{TM} is semi-decidable (r.e.)

but NOT decidable

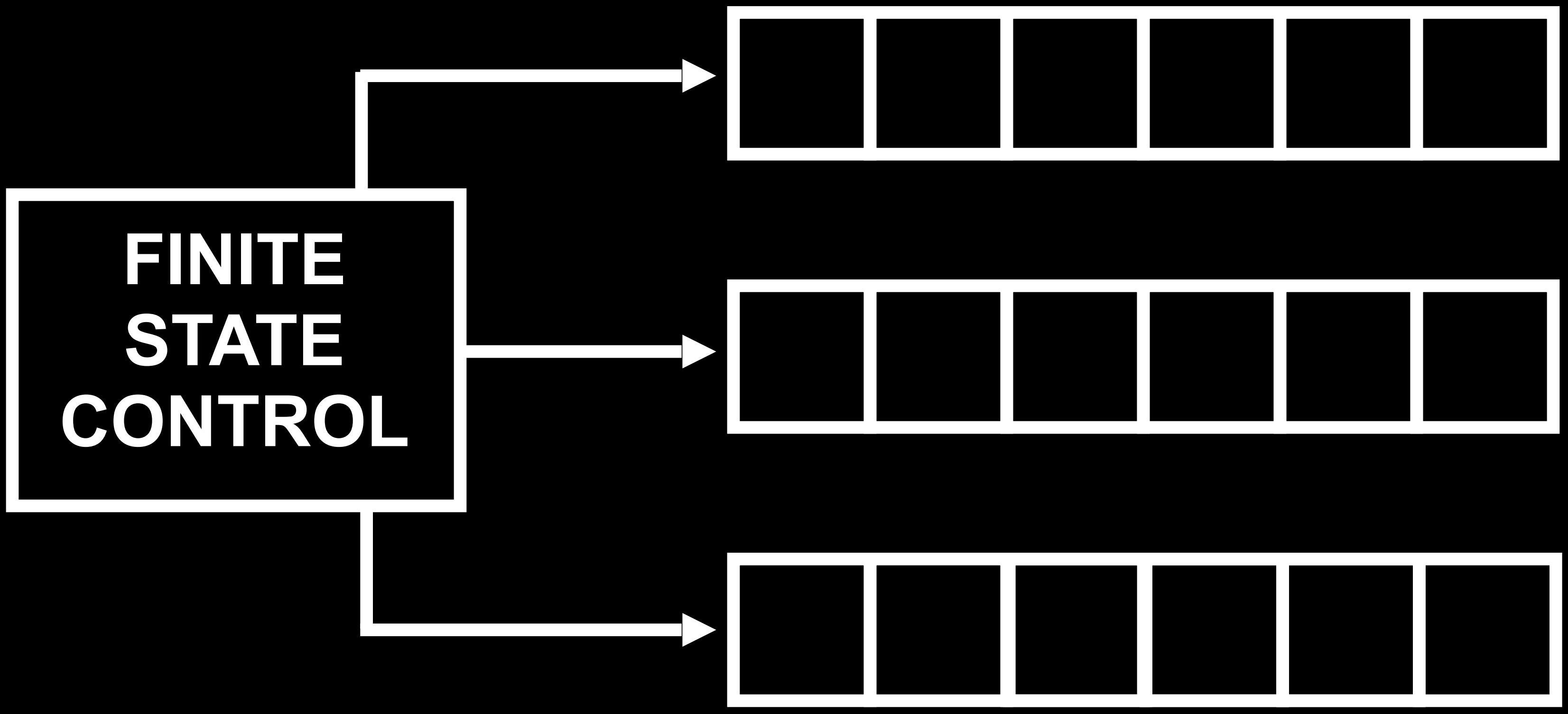
A_{TM} is r.e. :

Define a TM U as follows:

On input (M, w), U runs M on w. If M ever accepts, accept. If M ever rejects, reject.

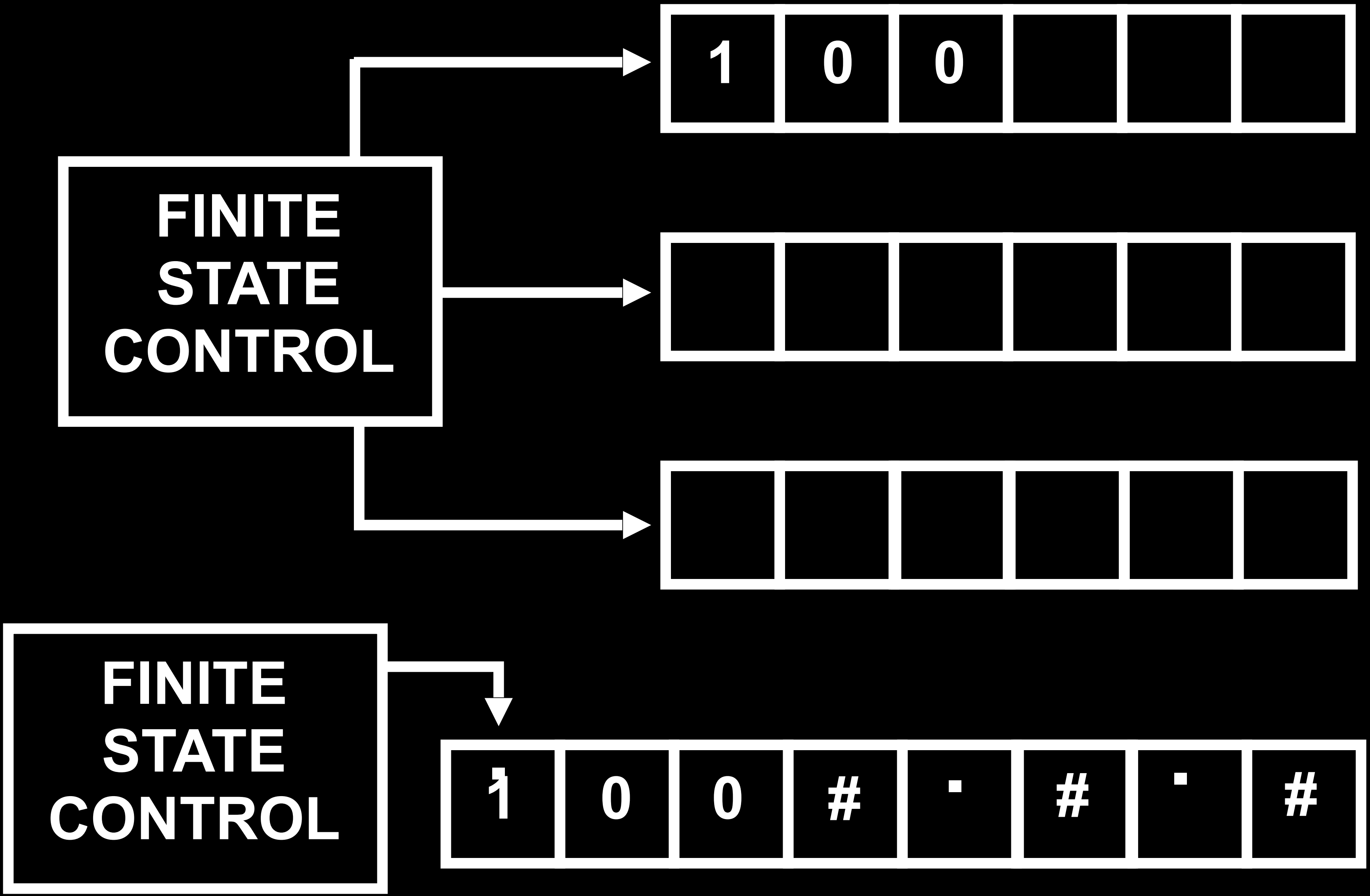
NB. When we write "input **(M, w)**" we really mean "input   code for (code for **M**, **w**)"

MULTITAPE TURING MACHINES

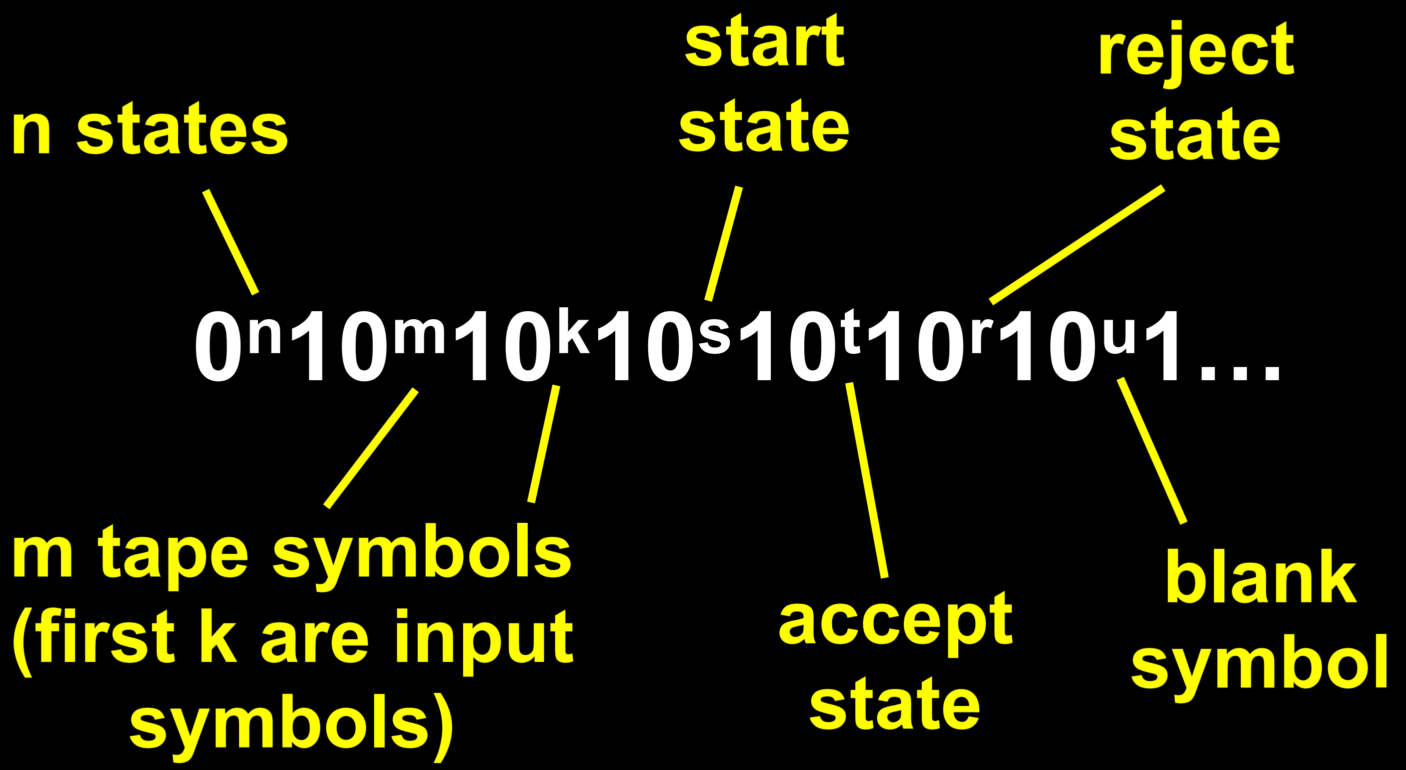


$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



We can encode a TM as a string of 0s and 1s



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

n states

start state

reject state

m tape symbols (first k are input symbols)

accept state

blank symbol

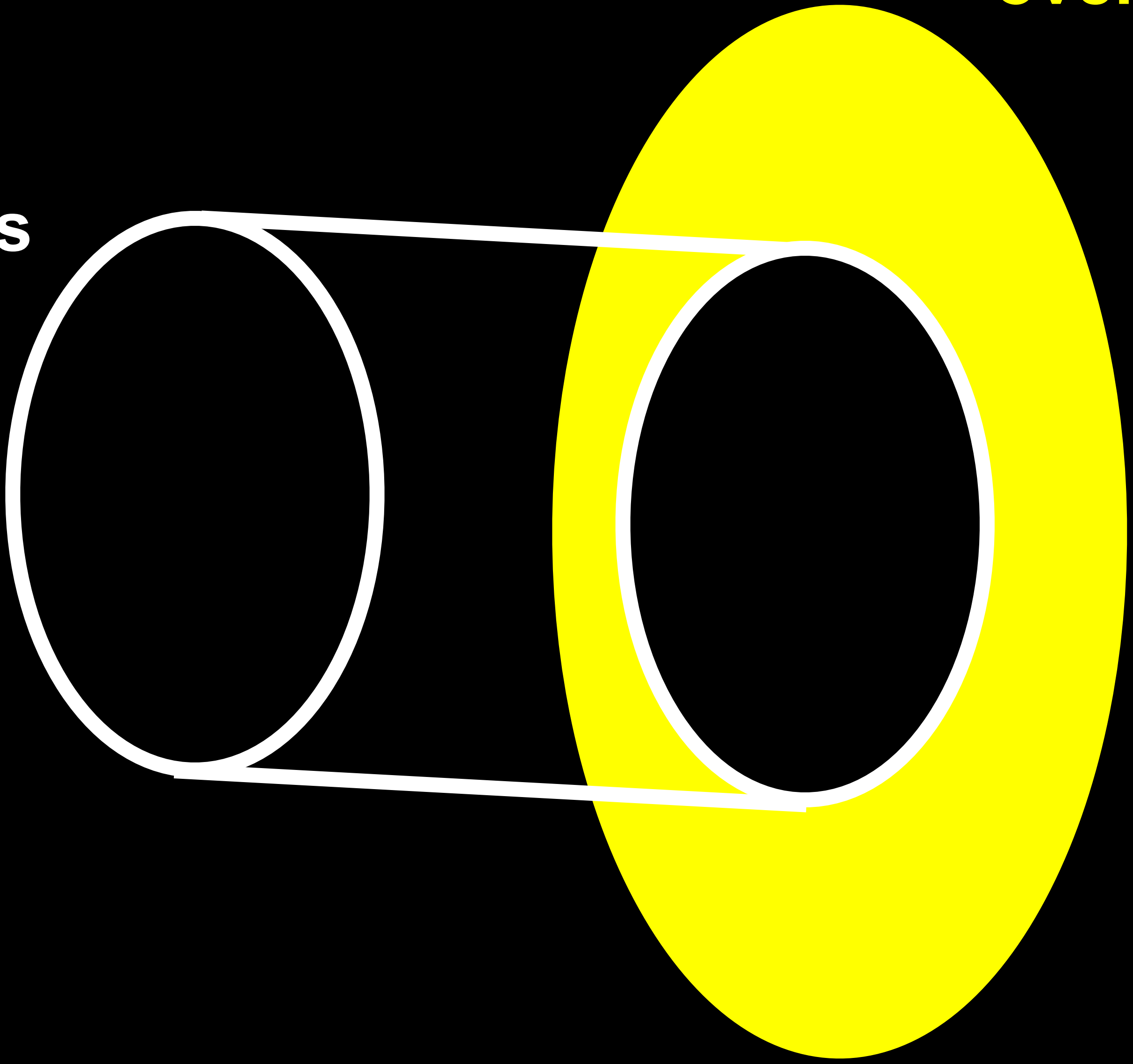
$$(\,(p,\, a),\, (q,\, b,\, L)\,) = 0^p 1 0^a 1 0^q 1 0^b 1 0$$

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

UNDECIDABLE PROBLEMS

THURSDAY Feb 13

There are languages over {0,1} that are not decidable



Languages over {0,1}

Turing Machines

Let L be any set and 2^L be the power set of L

Theorem: There is no onto map from L to 2^L

Proof: Assume, for a contradiction, that there is an onto map $f : L \rightarrow 2^L$

$$\text{Let } S = \{ x \in L \mid x \notin f(x) \}$$

If $S = f(y)$ then $y \in S$ if and only if $y \notin S$

Can give a more constructive argument!

Theorem: There is no onto function from the positive integers to the real numbers in (0, 1)

Proof: Suppose f is any function mapping the positive integers to the real numbers in (0,

1 → 0.28347279…

2 → 0.88388384…

3 → 0.77635284…

4 → 0.11111111…

5 → 0.12345678…

:　　　　:

$$[\text{ } n\text{-th digit of } r \text{ }] = \begin{cases} 1 & \text{if } [\text{ } n\text{-th digit of } f(n) \text{ }] \neq 1 \\ 2 & \text{otherwise} \end{cases}$$

$f(n) \neq r$ for all n (Here, r = 11121...)

THE MORAL:
No matter what L is,
2^L *always* has more elements than L

Not all languages over {0,1} are decidable, in fact: not all languages over {0,1} are semi-decidable

{decidable languages over {0,1}}

{semi-decidable languages over {0,1}}

{Turing Machines}

{Strings of 0s and 1s}

||

Set L

{Languages over {0,1}}

||

{Sets of strings of 0s and 1s}

||

Set of all subsets of L: 2^L

THE ACCEPTANCE PROBLEM

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

Theorem: A_{TM} is semi-decidable (r.e.)

but **NOT** decidable

A_{TM} is r.e. :

Define a TM U as follows:

On input (M, w), U runs M on w. If M ever accepts, accept. If M ever rejects, reject.

NB. When we write "input (M, w)" we really mean "input code for (code for M, w)"

THE ACCEPTANCE PROBLEM

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

Theorem: A_{TM} is semi-decidable (r.e.)

but NOT decidable

A_{TM} is r.e. :

Define a TM U as follows:

U is a *universal TM*

On input (M, w), U runs M on w. If M ever accepts, accept. If M ever rejects, reject.

Therefore,

U accepts (M,w) ⇔ M accepts w ⇔ (M,w) $\in A_{TM}$

Therefore, U *recognizes* A_{TM}

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

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

OUTPUT OF H

	M_1	M_2	M_3	M_4	…	D
M_1	accept	accept	accept	reject		accept
M_2	reject	accept	reject	reject		reject
M_3	accept	reject	reject	accept		accept
M_4	accept	reject	reject	reject		accept
:						
D	reject	reject	accept	accept		?

Theorem: A_{TM} is r.e. but NOT decidable

Cor: $\neg A_{TM}$ is not even r.e.!

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

A_{TM} is undecidable: A constructive proof:

Let machine H semi-decides A_{TM} (Such ∃ , why?)

$$H(\,(M,w)\,) = \begin{cases} \text{Accept} & \text{if } M \text{ accepts } w \\ \text{Reject or} & \\ \text{No output} & \text{if } M \text{ does not accept } w \end{cases}$$

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

$$D(\,D\,) = \begin{cases} \text{Reject} & \text{if H (D, D) Accepts} \\ \text{Accept} & \text{if H (D, D) Rejects} \\ \text{No output} & \text{if H D,|, D) has No output} \end{cases}$$

H((D,D)) = No output No Contradictions !

We have shown:

Given any machine H for semi-deciding A_{TM}, we can *effectively construct* a TM D such that (D,D) $\notin$ A_{TM} but H fails to tell us that.

That is, H *fails* to be a decider on instance (D,D).

In other words,

Given any "good" candidate for deciding the *Acceptance Problem*, we can effectively construct an instance where the candidate fails.

THE classical HALTING PROBLEM

$\mathrm{HALT}_{\mathrm{TM}}$ = { (M,w) | M is a TM that halts on string w }

Theorem: $\mathrm{HALT}_{\mathrm{TM}}$ is undecidable

Proof: Assume, for a contradiction, that TM H decides $\mathrm{HALT}_{\mathrm{TM}}$

We use H to construct a TM D that decides A_{TM}

```
On input (M,w), D runs H on (M,w):
    If H rejects then reject
    If H accepts, run M on w until it halts:
        Accept if M accepts, ie halts in an accept state
        Otherwise reject
```

MAPPING REDUCIBILITY

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

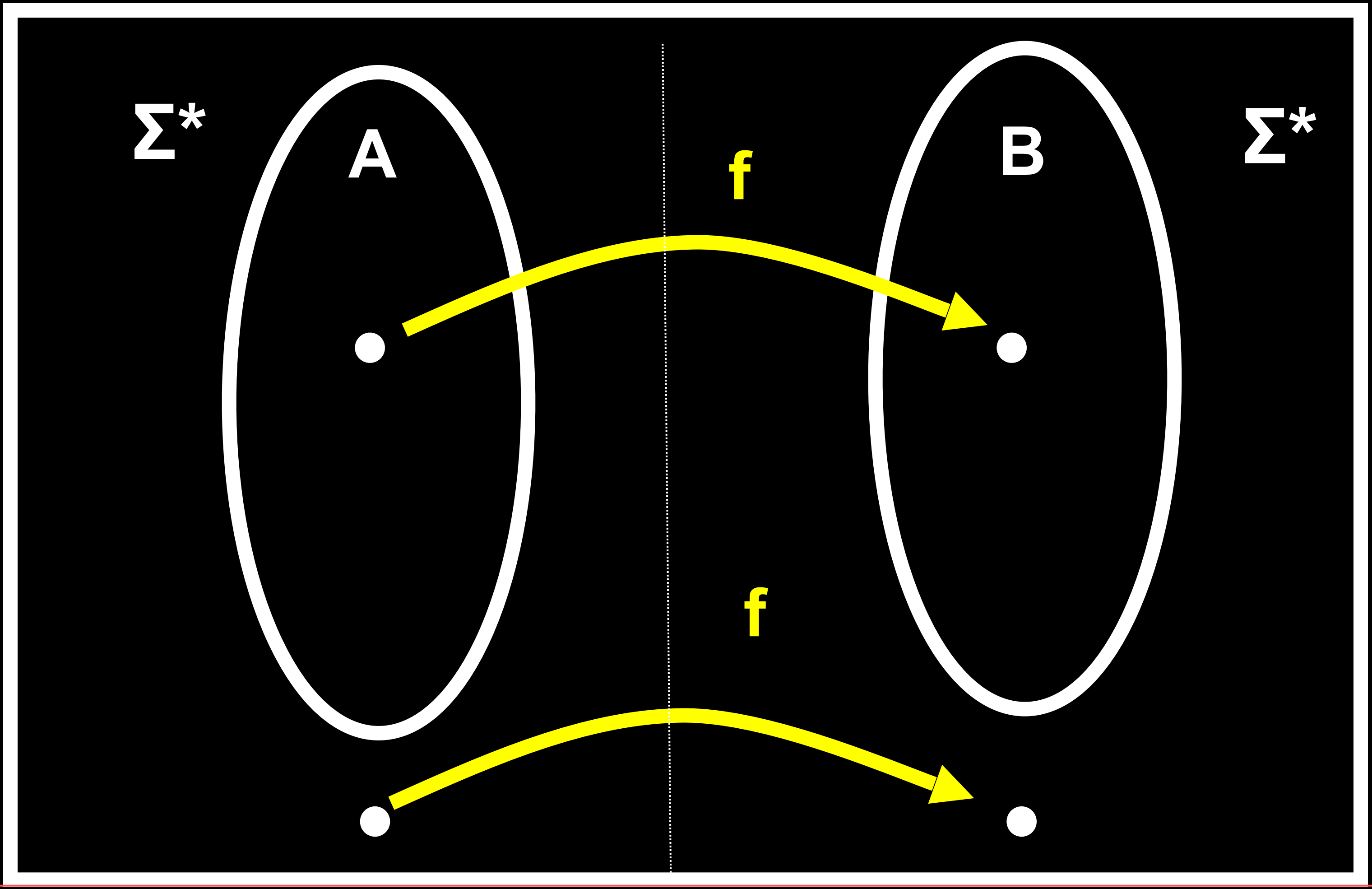
A language A is ***mapping reducible*** to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every w,

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

f is called a *reduction* from A to B

Think of f as a "computable coding"

A is mapping reducible to B, $A \leq_m B$, if there is a computable $f : \Sigma^* \to \Sigma^*$ such that $w \in A \Leftrightarrow f(w) \in B$



Also, $\neg A \leq_m \neg B$, why?

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

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

We build a machine N that decides A as follows:

```
On input w:
    1. Compute f(w)
    2. Run M on f(w)
```

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

Proof: Let M (semi) decide B and let f be a reduction from A to B

We build a machine N that (semi) decides A as follows:

```
On input w:
    1. Compute f(w)
    2. Run M on f(w)
```

RICE'S THEOREM

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

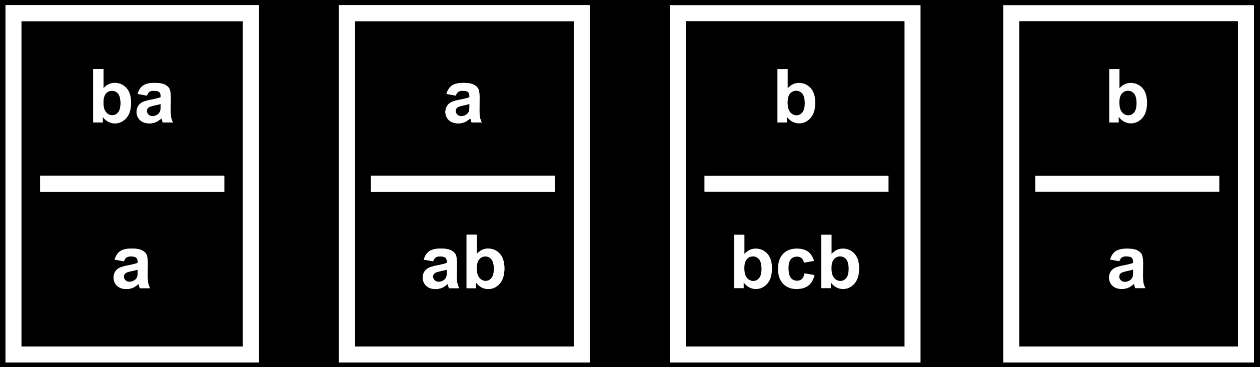
1. **For TMs M_1 and M_2, if $M_1 \equiv M_2$ then**

$$M_1 \in L \Leftrightarrow M_2 \in L$$

2. **There are TMs M_1 and M_2, such that $M_1 \in L$ and $M_2 \notin L$**

Then L is undecidable

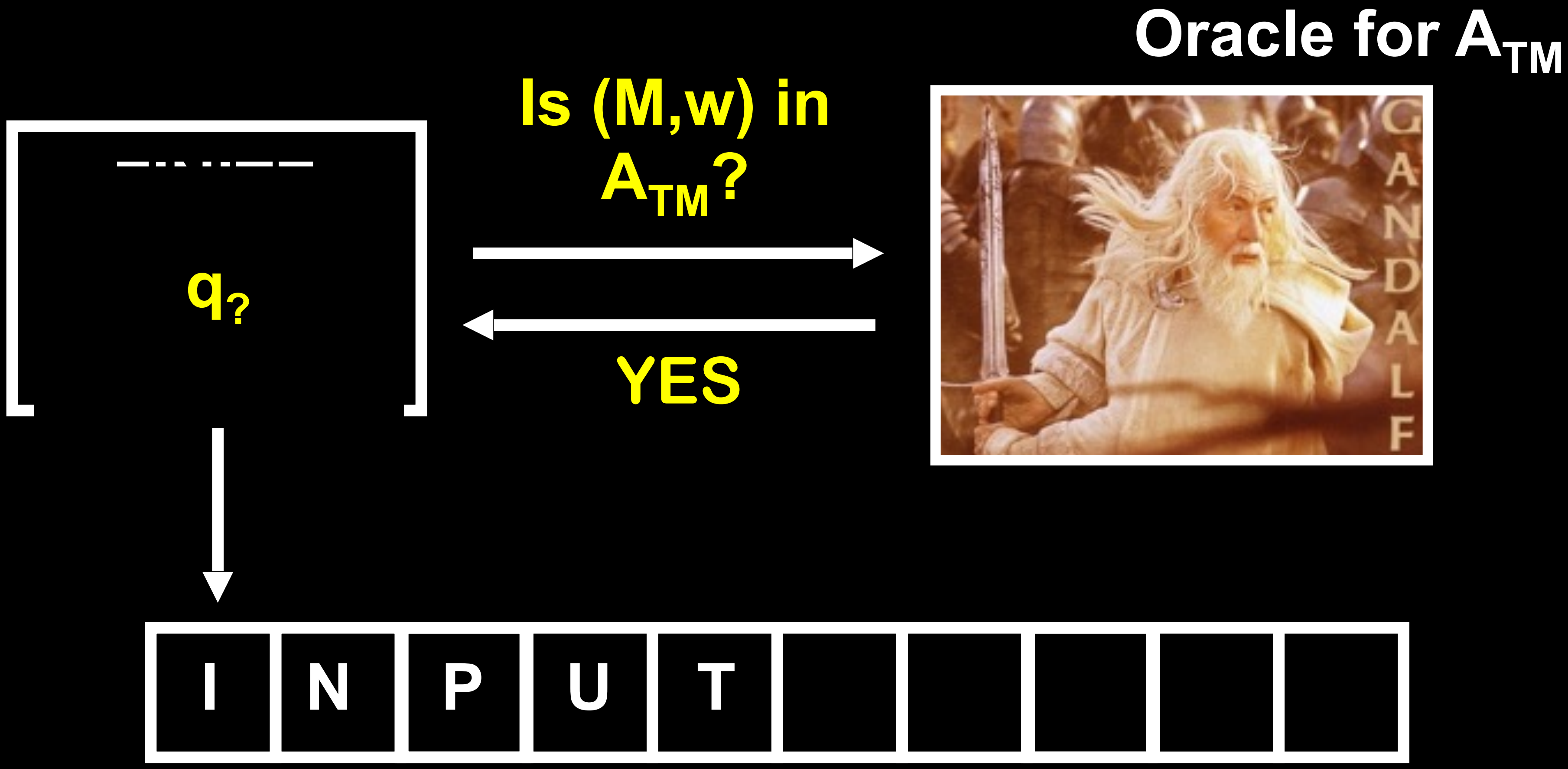
THE PCP **GAME**



ba	a	b	b
a	ab	bcb	a

THE ARITHMETIC HIERARCHY

ORACLE TMs



Oracle for A_{TM}

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

$q_?$

YES

I N P U T

INFINITE TAPE

ORACLE MACHINES

An ORACLE is a set B to which the TM may pose membership questions “Is w in B?”
(formally: TM enters state $q_?$)
and the TM always receives a correct answer in one step
(formally: if the string on the “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 do not assume that the oracle B is a computable set!)

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

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

Language A “Turing Reduces” to Language B

if A is decidable in B, ie if there is an oracle TM M with oracle B that decides A

$$A \leq_T B$$

$\leq_T$ 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: $\neg AT_{TM} \leq_T AT_{TM}$

Theorem: $\neg AT_{TM} \not\leq_m AT_{TM}$

THE ARITHMETIC **HIERARCHY**

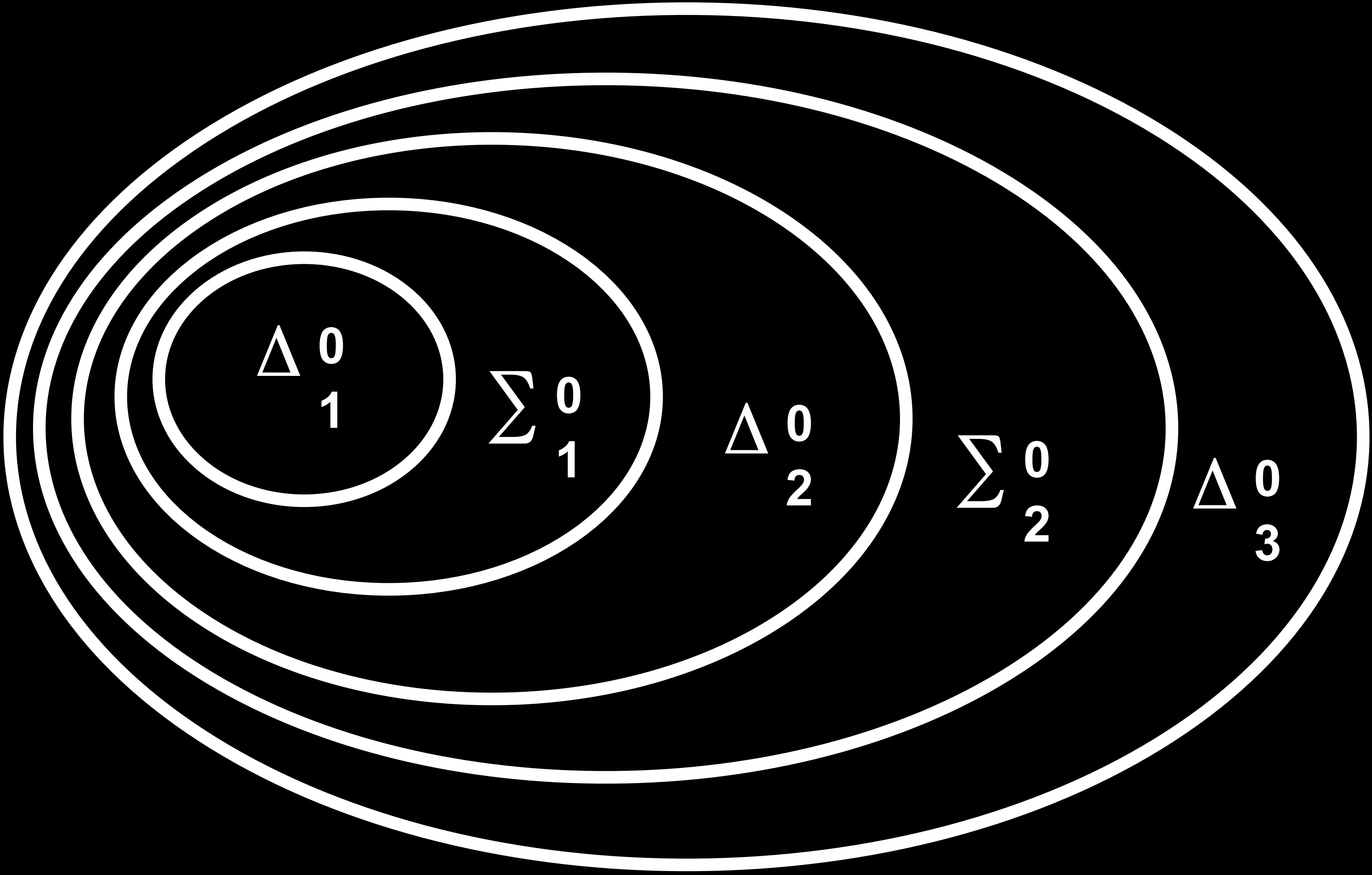
Δ^0_1 = { decidable sets } (sets = languages)

Σ^0_1 = { semi-decidable sets }

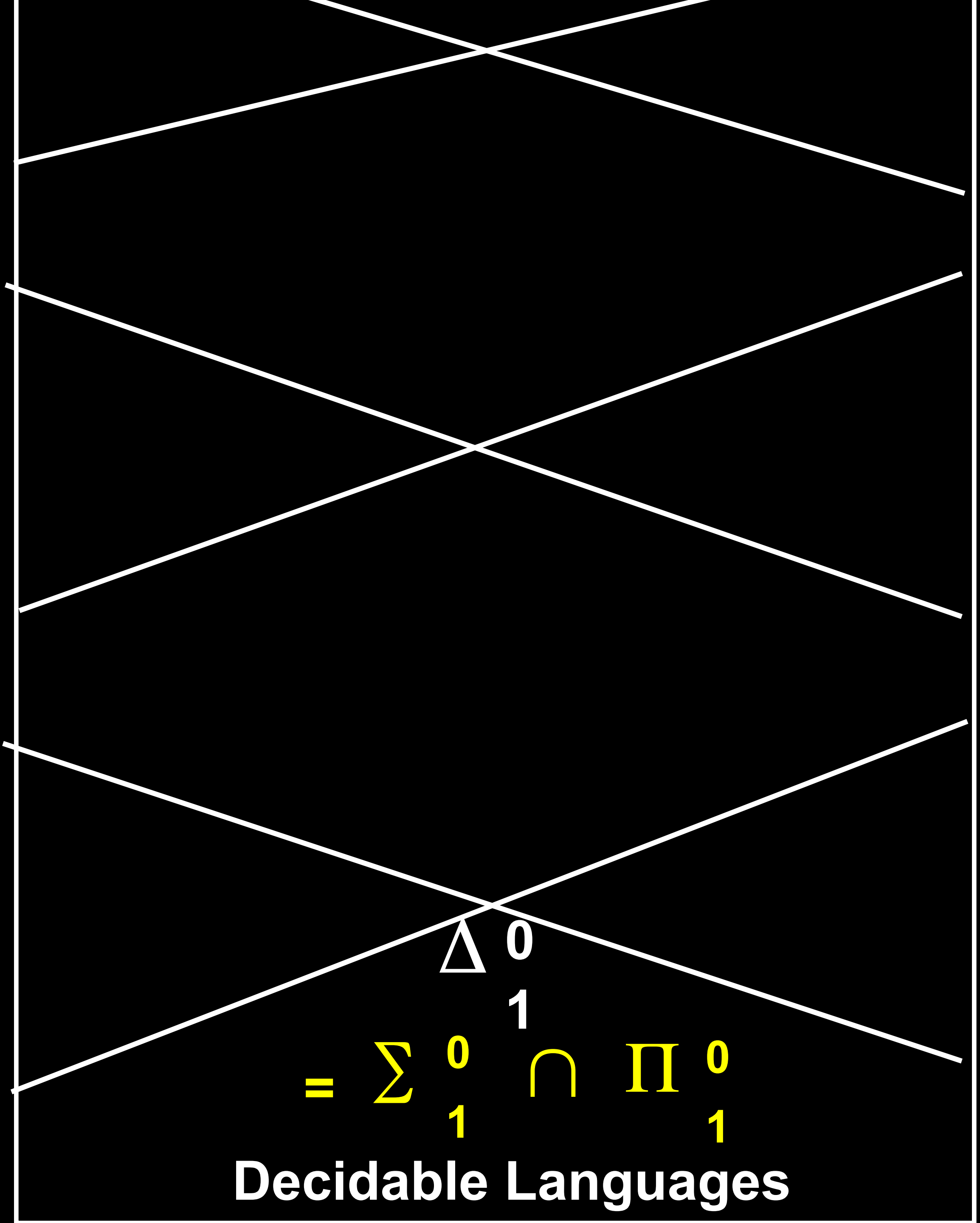
Σ^0_{n+1} = { sets semi-decidable in some B $\in \Sigma^0_n$ }

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_1

Semi-decidable Languages

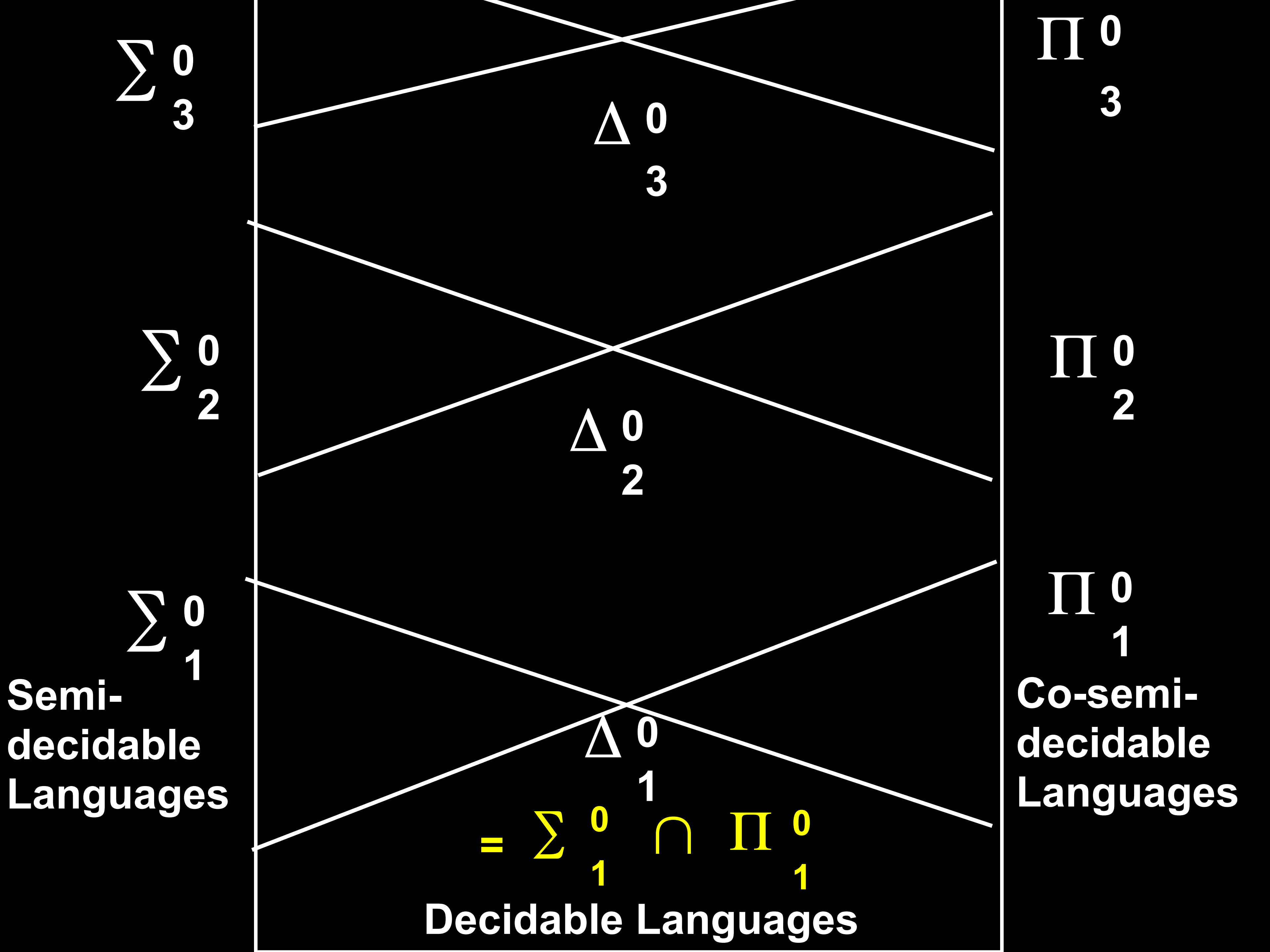
Π^0_1

Co-semi-decidable Languages

Δ^0_1

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

Decidable Languages



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

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}

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 ⇒ M(x,y) accepts
R(x,y) is FALSE ⇒ 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^*

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 ⇒ M(x,y) accepts
R(x,y) is FALSE ⇒ 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

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

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

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

Proof:

(1) If $A = \{ x \mid \exists y\, R(x,y) \}$ then A is semi-decidable

Because we can enumerate over all y's

(2) If A is semi-decidable, then $A = \{ x \mid \exists y\, R(x,y) \}$

Let M semi-decide A

Then, $A = \{ x \mid \exists y\, T(\langle M\rangle, x, y) \}$ (Here M is fixed.)

where

Kleene's T predicate, T(<M>, x, y): M accepts x in y steps.

THE PAIRING FUNCTION

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

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

Proof: Let $w = w_1 \ldots w_n \in \Sigma^*$, $t \in \Sigma^*$.

Let $a, b \in \Sigma$, $a \neq b$.

$$<w, t> := a\, w_1 \ldots a\, w_n\, b\, t$$

$\pi_1 (z)$:= "if z has the form $a\, w_1 \ldots a\, w_n\, b\, t$, then output $w_1 \ldots w_n$, else output ε"

$\pi_2(z)$:= "if z has the form $a\, w_1 \ldots a\, w_n\, b\, t$, then output t, else output ε"

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

Example

Decidable predicate

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

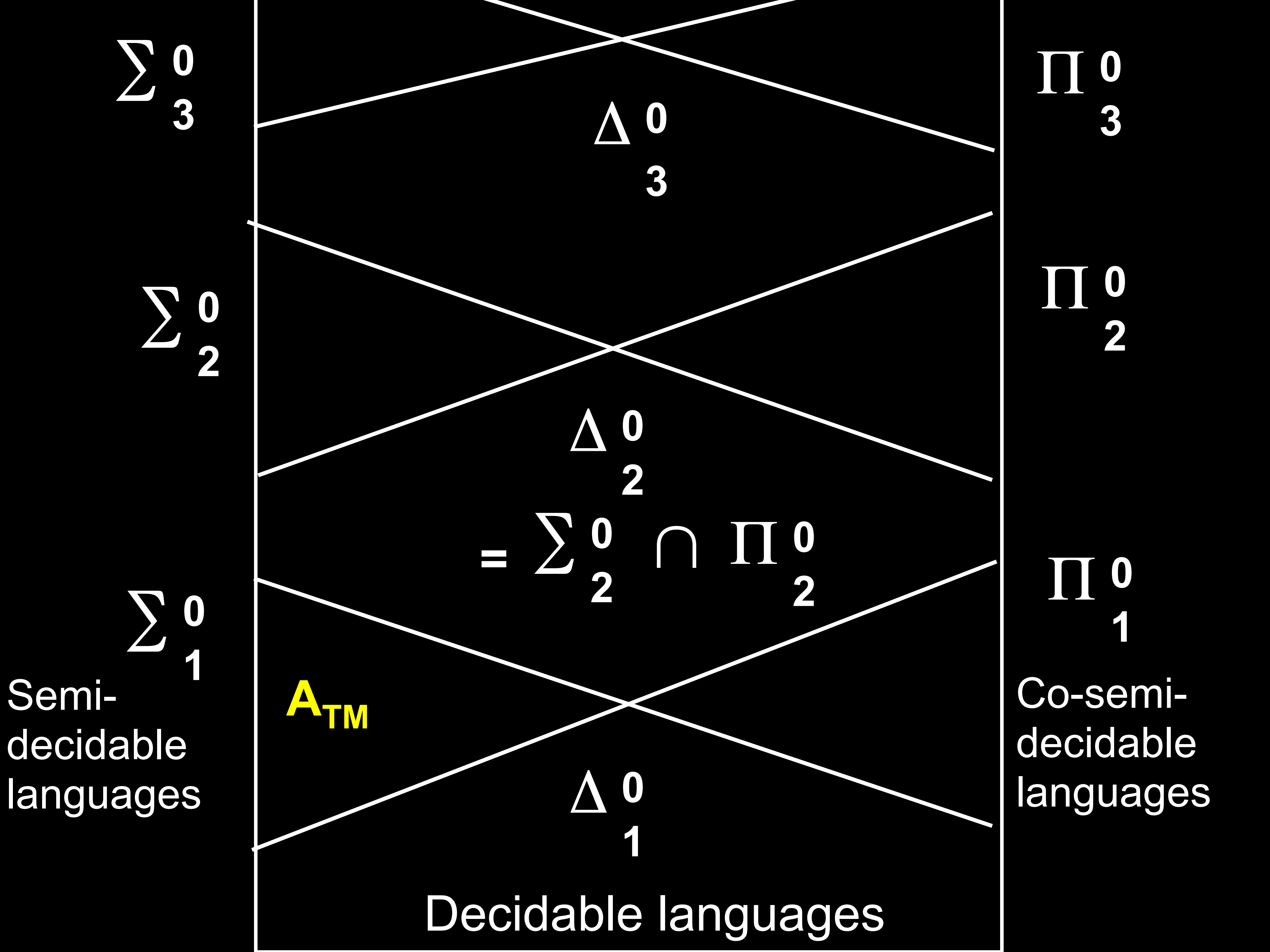
We know that A_{TM} is in Σ^0_1 Why?

Show it can be described in this form:

A_{TM} = { <(M,w)> | ∃t $\underbrace{[\text{M accepts w in t steps}]}_{\text{decidable predicate}}$ }

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^0_3

Δ^0_3

Π^0_3

Σ^0_2

Π^0_2

Δ^0_2

$= \Sigma^0_2 \cap \Pi^0_2$

Σ^0_1

Π^0_1

Semi-decidable languages

A_{TM}

Co-semi-decidable languages

Δ^0_1

Decidable languages

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

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

$$\text{EMPTY} = \{ M \mid \underbrace{\forall w \forall t}_{\text{two quantifiers??}} [\underbrace{\text{M doesn't accept w in t steps}}_{\text{decidable predicate}}] \}$$

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

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

$$\text{EMPTY} = \{ M \mid \underbrace{\forall w \forall t}_{\text{two quantifiers??}} [\underbrace{\neg T(\langle M\rangle, w, t)\ ]}_{\text{decidable predicate}} \}$$

THE PAIRING FUNCTION

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

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

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

THE PAIRING FUNCTION

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

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

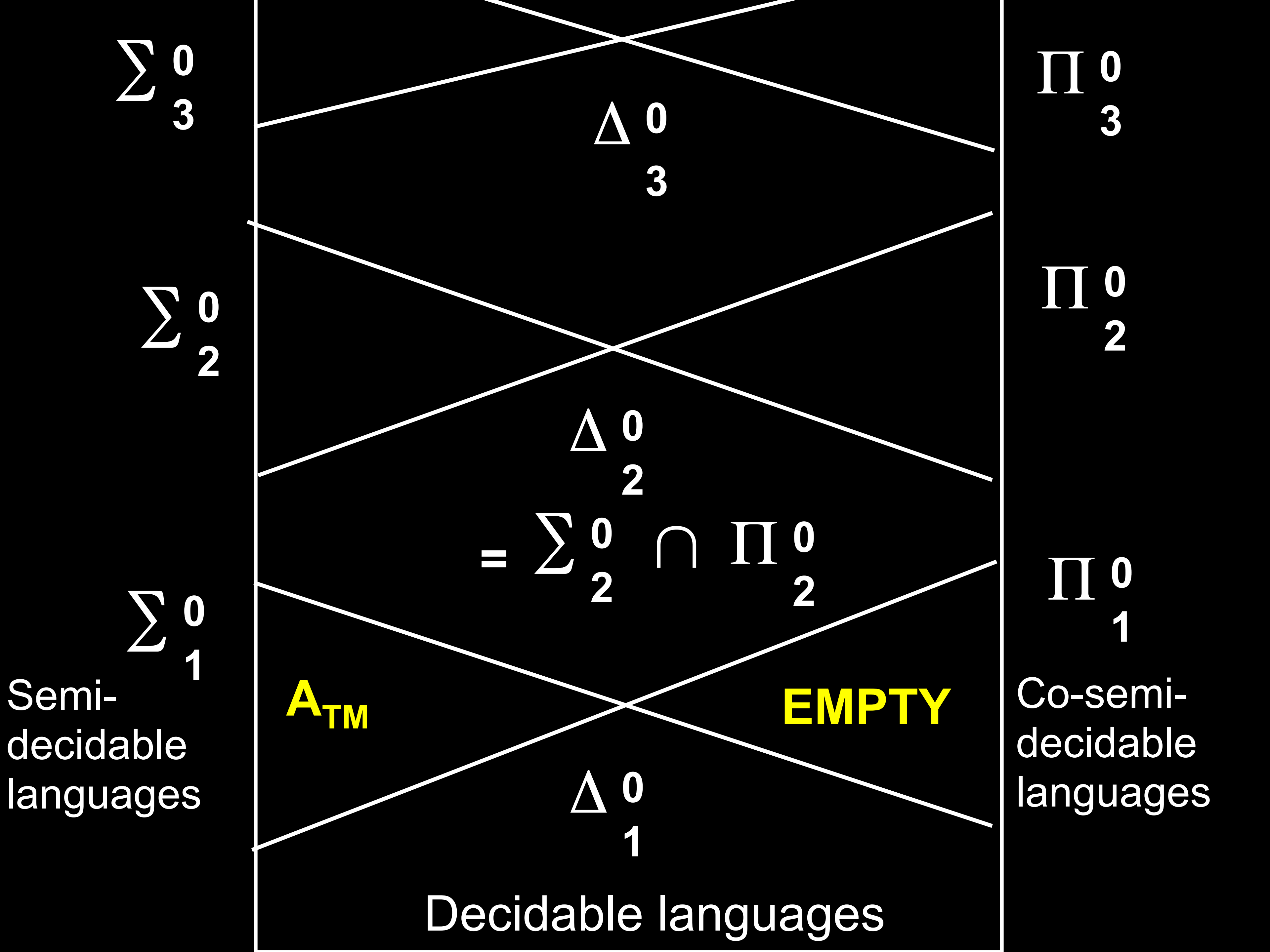
Proof: Let $w = w_1 \ldots w_n \in \Sigma^*$, $t \in \Sigma^*$.

Let $a, b \in \Sigma$, $a \neq b$.

$$<w, t> := a\, w_1 \ldots a\, w_n\, b\, t$$

$\pi_1(z)$:= "if z has the form $a\, w_1 \ldots a\, w_n\, b\, t$, then output $w_1 \ldots w_n$, else output ε"

$\pi_2(z)$:= "if z has the form $a\, w_1 \ldots a\, w_n\, b\, t$, then output t, else output ε"



Σ^0_3

Π^0_3

Δ^0_3

Σ^0_2

Π^0_2

$\Delta^0_2 = \Sigma^0_2 \cap \Pi^0_2$

Σ^0_1

Π^0_1

Semi-decidable languages

A_{TM}

EMPTY

Co-semi-decidable languages

Δ^0_1

Decidable languages

Π^0_2 = languages of the form $\{ x \mid \forall y \exists z\ R(x,y,z) \}$

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

is in Π^0_2

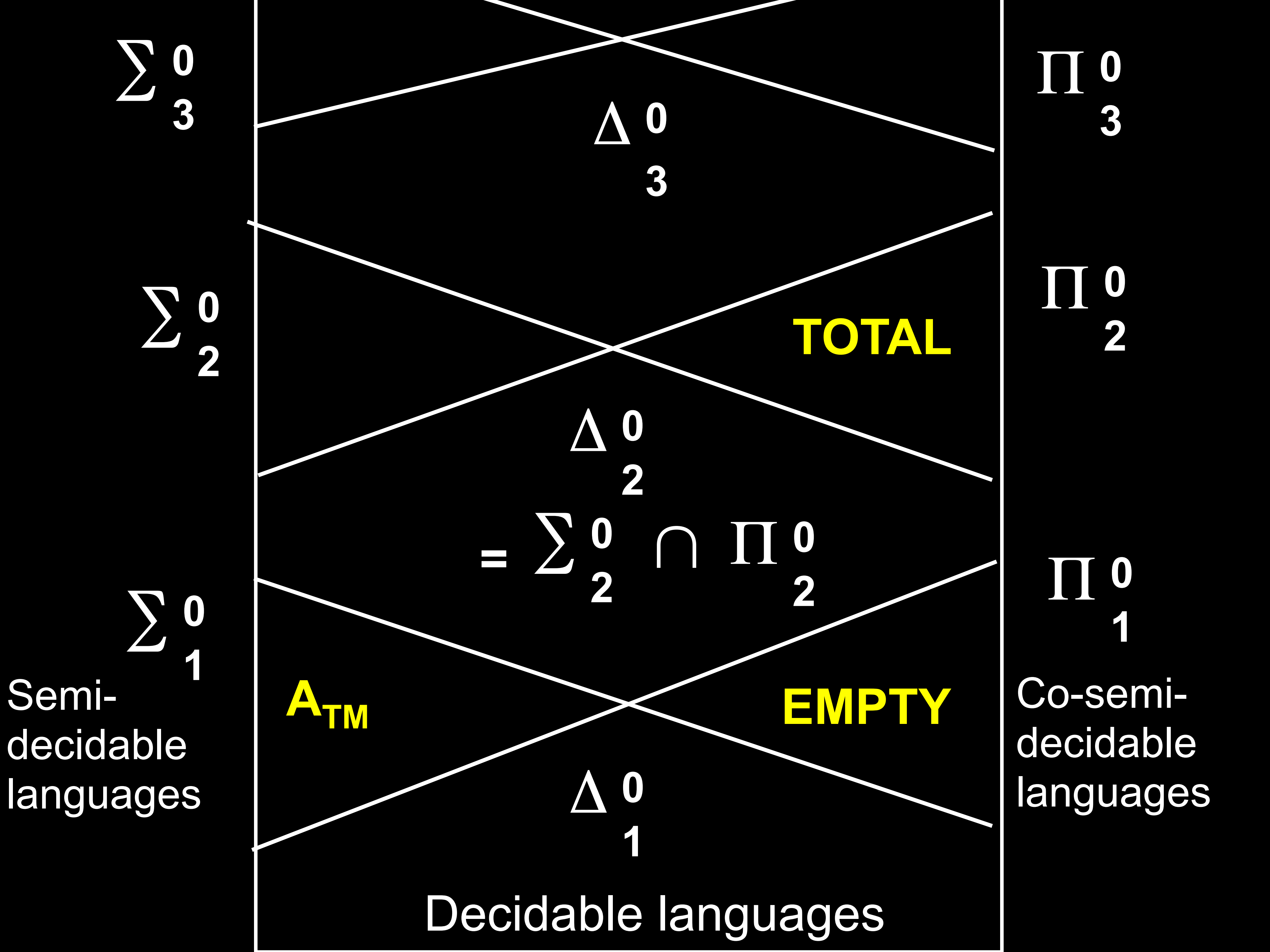
$$\text{TOTAL} = \{ M \mid \forall w\ \exists t\ \underbrace{[\text{M halts on w in t steps}]}_{\text{decidable predicate}} \}$$

Π^0_2 = languages of the form $\{ x \mid \forall y \exists z \; R(x,y,z) \}$

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

is in Π^0_2

$$\text{TOTAL} = \{ M \mid \forall w \; \exists t \; [\underbrace{T(\langle M\rangle, w, t) \;] \;\}}_{\text{decidable predicate}}$$



Σ^0_3

Δ^0_3

Π^0_3

Σ^0_2

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

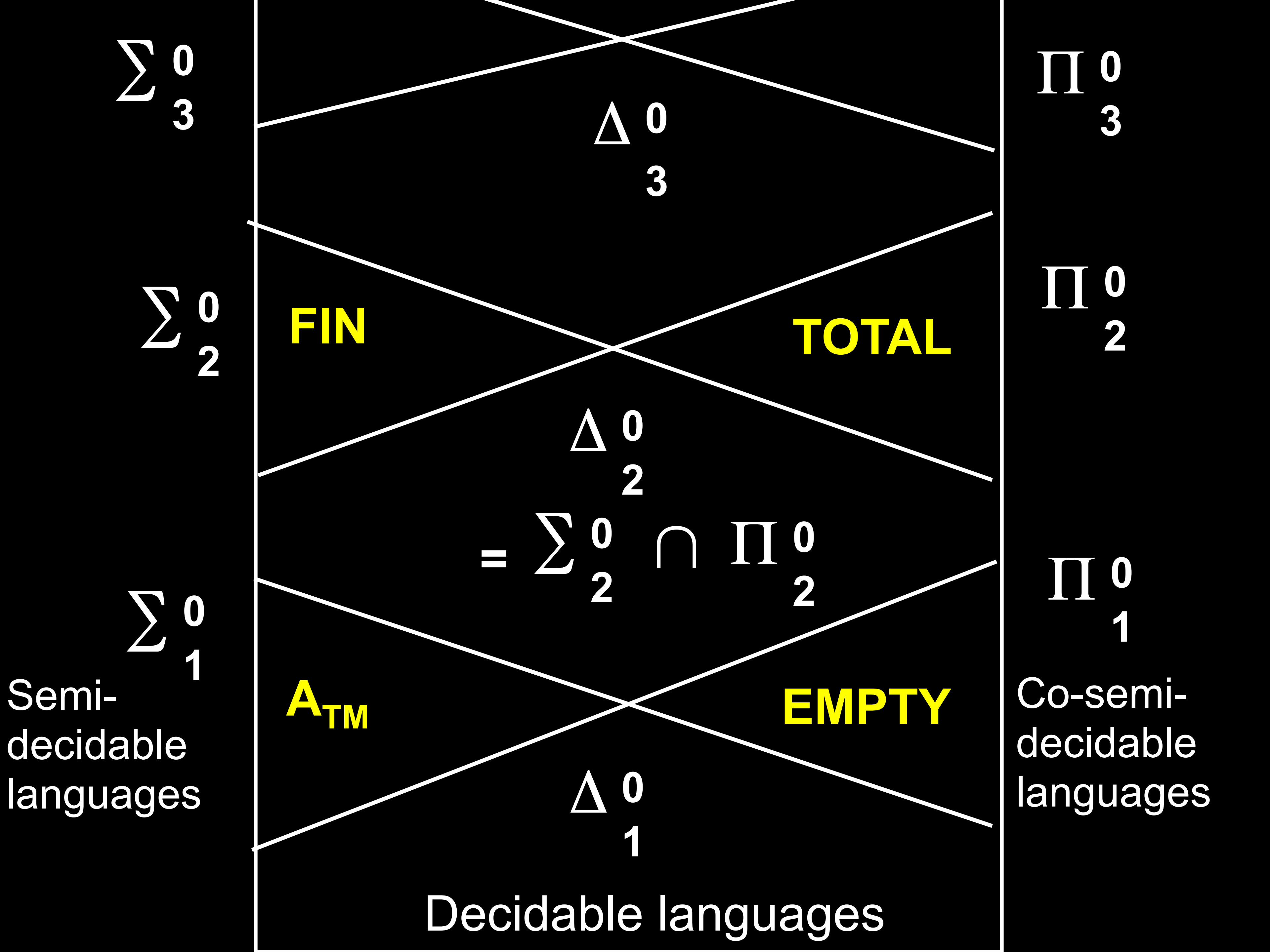
Decidable languages

Σ^0_2 = languages of the form $\{ x \mid \exists y \forall z\, R(x,y,z) \}$

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] }

$$\text{FIN} = \{ M \mid \exists n \forall w \forall t\ (\underbrace{|w| < n \vee \neg\, T(\langle M \rangle, w, t)}_{\text{decidable predicate}})\}$$



Σ^0_3

Δ^0_3

Π^0_3

Σ^0_2

FIN

TOTAL

Π^0_2

$\Delta^0_2 = \Sigma^0_2 \cap \Pi^0_2$

Σ^0_1

Π^0_1

Semi-decidable languages

A_{TM}

EMPTY

Co-semi-decidable languages

Δ^0_1

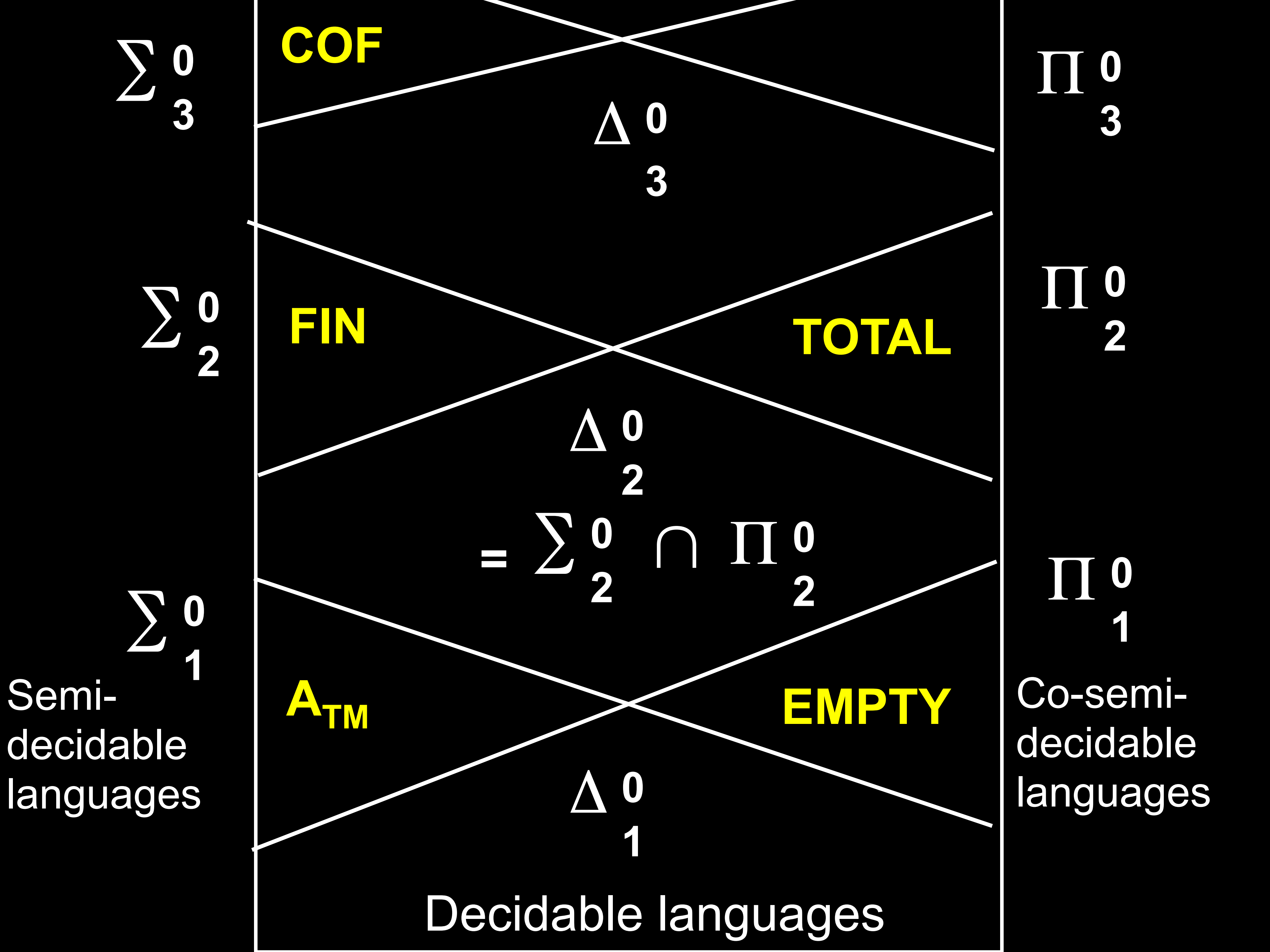
Decidable languages

Σ^0_3 = languages of the form $\{\ x \mid \exists y \forall z \exists u\ R(x,y,z,u)\ \}$

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

COF = $\{\ M \mid \exists n \forall w \exists\ t\ [\ |w| > n \Rightarrow$ M accept w in t steps]

$$\text{COF} = \{\ M \mid \exists n \forall w \exists\ t\ (\ \underbrace{|w| \leq n \vee T(<M>,w,\ t)}_{\textbf{decidable predicate}}\ )\}$$



Σ_3^0 COF Π_3^0

Δ_3^0

Σ_2^0 FIN TOTAL Π_2^0

$\Delta_2^0 = \Sigma_2^0 \cap \Pi_2^0$

Σ_1^0 Π_1^0

Semi-decidable languages

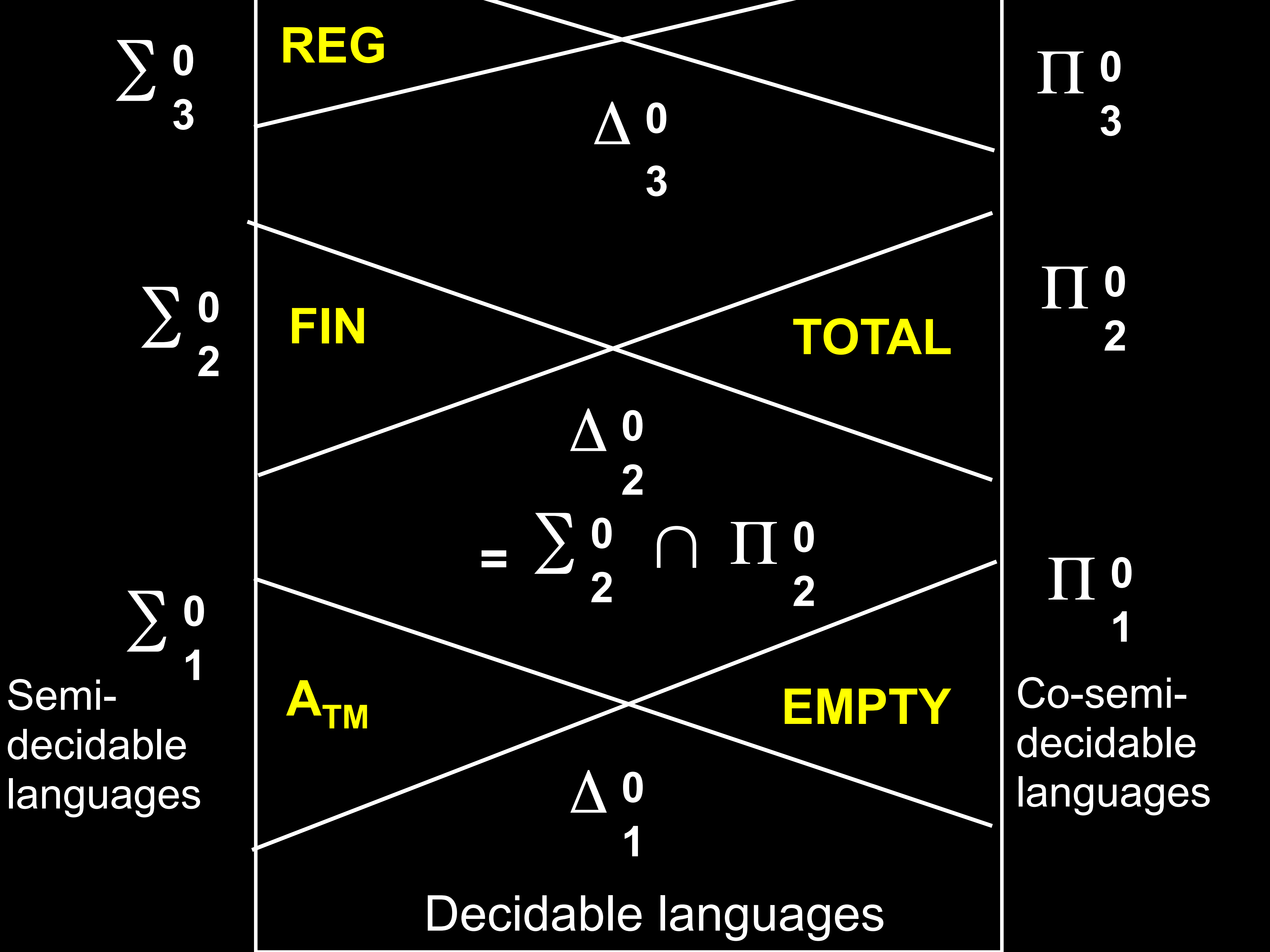
A_{TM}

EMPTY

Co-semi-decidable languages

Δ_1^0

Decidable languages



Σ^0_3

REG

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

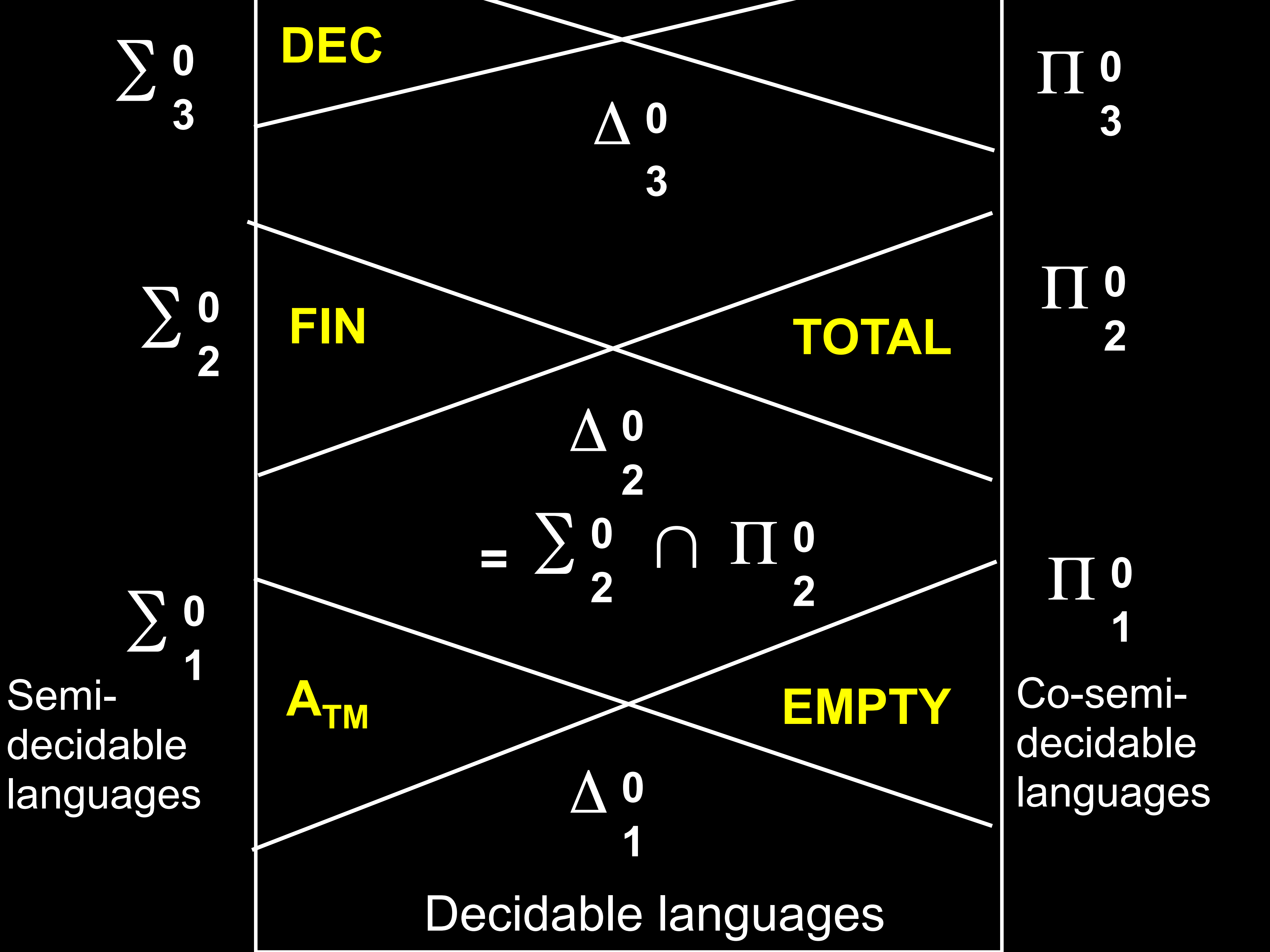
A_{TM}

EMPTY

Co-semi-decidable languages

Δ^0_1

Decidable languages



Σ^0_3

DEC

Π^0_3

Δ^0_3

Σ^0_2

FIN

TOTAL

Π^0_2

$\Delta^0_2 = \Sigma^0_2 \cap \Pi^0_2$

Σ^0_1

Π^0_1

A_{TM}

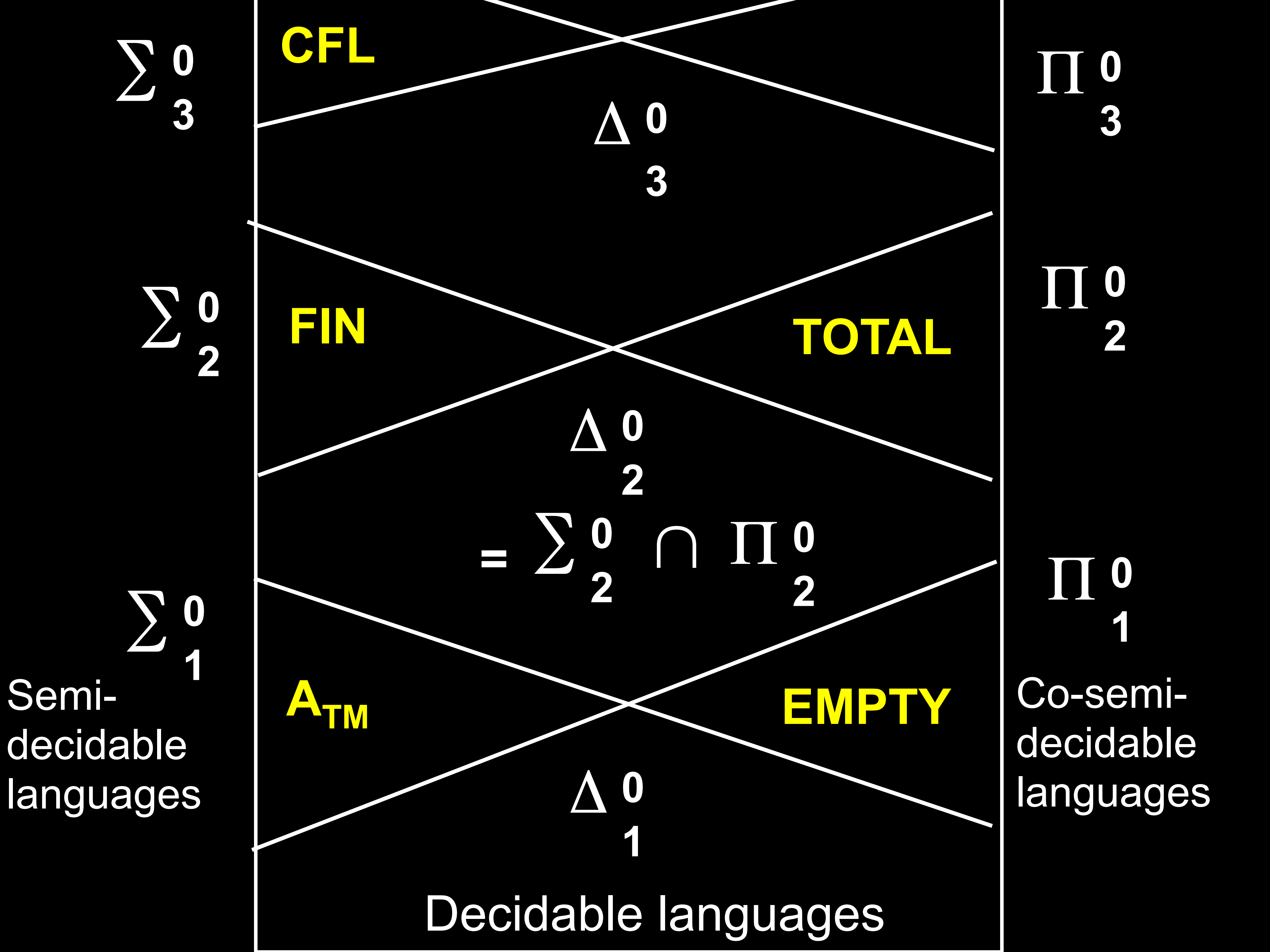
EMPTY

Semi-decidable languages

Co-semi-decidable languages

Δ^0_1

Decidable languages



Σ_3^0

CFL

Π_3^0

Δ_3^0

Σ_2^0

FIN

TOTAL

Π_2^0

$\Delta_2^0 = \Sigma_2^0 \cap \Pi_2^0$

Σ_1^0

Π_1^0

Semi-decidable languages

A_{TM}

EMPTY

Co-semi-decidable languages

Δ_1^0

Decidable languages

Each is m-complete for its level in hierarchy and cannot go lower (by next Theorem, which shows the hierarchy does not collapse).

L is m-complete for class C if

i) L $\in$ C and

ii) L is m-hard for C,

ie, for all L’ $\in$ C , L’ $\leq_m$ L

A_{TM} is m-complete for class $C = \Sigma^0_1$

i) $A_{TM} \in C$

ii) A_{TM} is m-hard for C,

Suppose $L \in C$. Show: $L \leq_m A_{TM}$

Let M semi-decide L. Then Map $\Sigma^* \to \Sigma^*$

where w → (M, w).

Then, $w \in L \Leftrightarrow (M,w) \in A_{TM}$ QED

FIN is m-complete for class $C = \Sigma_2^0$

i) FIN $\in$ C

ii) FIN is m-hard for C,

Suppose L $\in$ C . Show: L $\leq_m$ FIN

Suppose L= { w | ∃y∀z R(w,y,z) }
where R is decided by some TM D

Map Σ^* → Σ^*
where w → $N_{D,w}$

Supose $L \in \Sigma^0_2$ ie $L = \{ w \mid \exists y \forall z\, R(w,y,z) \}$
where R is decided by some TM D

Show: $L \leq_m \mathrm{FIN}$

Map $\Sigma^* \rightarrow \Sigma^*$
where $w \rightarrow N_{D,w}$

Define $N_{D,w}$ On input s:

1. Write down all strings y of length $|s|$
2. For each y, try to find a z such that $\neg R(w, y, z)$ and accept if all are successful (here use D and w)

So, $w \in L \Leftrightarrow N_{D,w} \in \mathrm{FIN}$

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}

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.

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

How to code <M,w>?

Assume w in {0,1}* and we have a binary encoding of M

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 input w, halt."

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

This implies

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

(Note: M is fixed for all x. So |M| is constant)

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)

≤ (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

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

Theorem: COMPRESS is undecidable!

Proof:

```
M = "On input x ∈ {0,1}*, let x' = 1x
        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 $n < K(y^*) \leq c + \log n$. CONTRADICTION!

DETERMINING COMPRESSIBILITY

Theorem: K is not computable

Proof:

M = "On input $x \in \{0,1\}^*$, let $x' = 1x$

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 $\langle M,x \rangle$ describes y^*, and $|\langle M,x \rangle| \leq c + \log n$

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

TIME COMPLEXITY AND POLYNOMIAL TIME; NON DETERMINISTIC TURING MACHINES AND NP

THURSDAY Mar 20

COMPLEXITY THEORY

Studies what can and can't be computed under limited resources such as time, space, etc

Today: Time complexity

MEASURING TIME COMPLEXITY

We measure time complexity by counting the elementary steps required for a machine to halt

Consider the language $A = \{ 0^k1^k \mid k \geq 0 \}$

On input of length **n**:

~n **1. Scan across the tape and reject if the string is not of the form 0^i1^j**

~n^2 **2. Repeat the following if both 0s and 1s remain on the tape:**

Scan across the tape, crossing off a single 0 and a single 1

~n **3. If 0s remain after all 1s have been crossed off, or vice-versa, reject. Otherwise accept.**

Definition:

Suppose M is a TM that halts on all inputs.

The running time or time-complexity of M is the function f : N → N, where f(n) is the maximum number of steps that M uses *on any input of length n.*

ASYMPTOTIC ANALYSIS

$$5n^3 + 2n^2 + 22n + 6 = O(n^3)$$

BIG-O

Let f and g be two functions f, g : $N \to R^+$. We say that $f(n) = O(g(n))$ if there exist positive integers c and n_0 so that for every integer $n \geq n_0$

$$f(n) \leq cg(n)$$

When $f(n) = O(g(n))$, we say that g(n) is an asymptotic upper bound for f(n)

f asymptotically NO MORE THAN g

$$5n^3 + 2n^2 + 22n + 6 = O(n^3)$$

If $c = 6$ and $n_0 = 10$, then $5n^3 + 2n^2 + 22n + 6 \leq cn^3$

$$2n^{4.1} + 200283n^4 + 2 \;=\; O(n^{4.1})$$

$$3n\log_2 n + 5n\log_2\log_2 n \;=\; O(n\log_2 n)$$

$$n\log_{10} n^{78} \;=\; O(n\log_{10} n)$$

$$\log_{10} n = \log_2 n \boxed{/ \log_2 10}$$

$$O(n\log_{10} n) = O(n\log_2 n) = O(n\log n)$$

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

$A = \{ 0^k 1^k \mid k \geq 0 \} \in \mathrm{TIME}(n^2)$

$A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n\text{log } n)$

Cross off every other 0 and every other 1. If the # of 0s and 1s left on the tape is odd, reject

0000000000000111111111111

x0x0x0x0x0x0xx1x1x1x1x1x1x

xxx0xxx0xxx0xxxx1xxx1xxx1x

xxxxxxx0xxxxxxxxxxxx1xxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxx

We can prove that a TM cannot decide A in less time than O(nlog n)

*7.49 Extra Credit. Let $f(n) = o(n\log n)$. Then Time(f(n)) contains only regular languages.

where $f(n) = o(g(n))$ iff $\lim_{n\to\infty} f(n)/g(n) = 0$

ie, for all $c > 0$, $\exists\, n_0$ such that $f(n) < cg(n)$ for all $n \geq n_0$

f asymptotically LESS THAN g

Can A = { 0^k1^k | $k \geq 0$ } be decided in time O(n) with a two-tape TM?

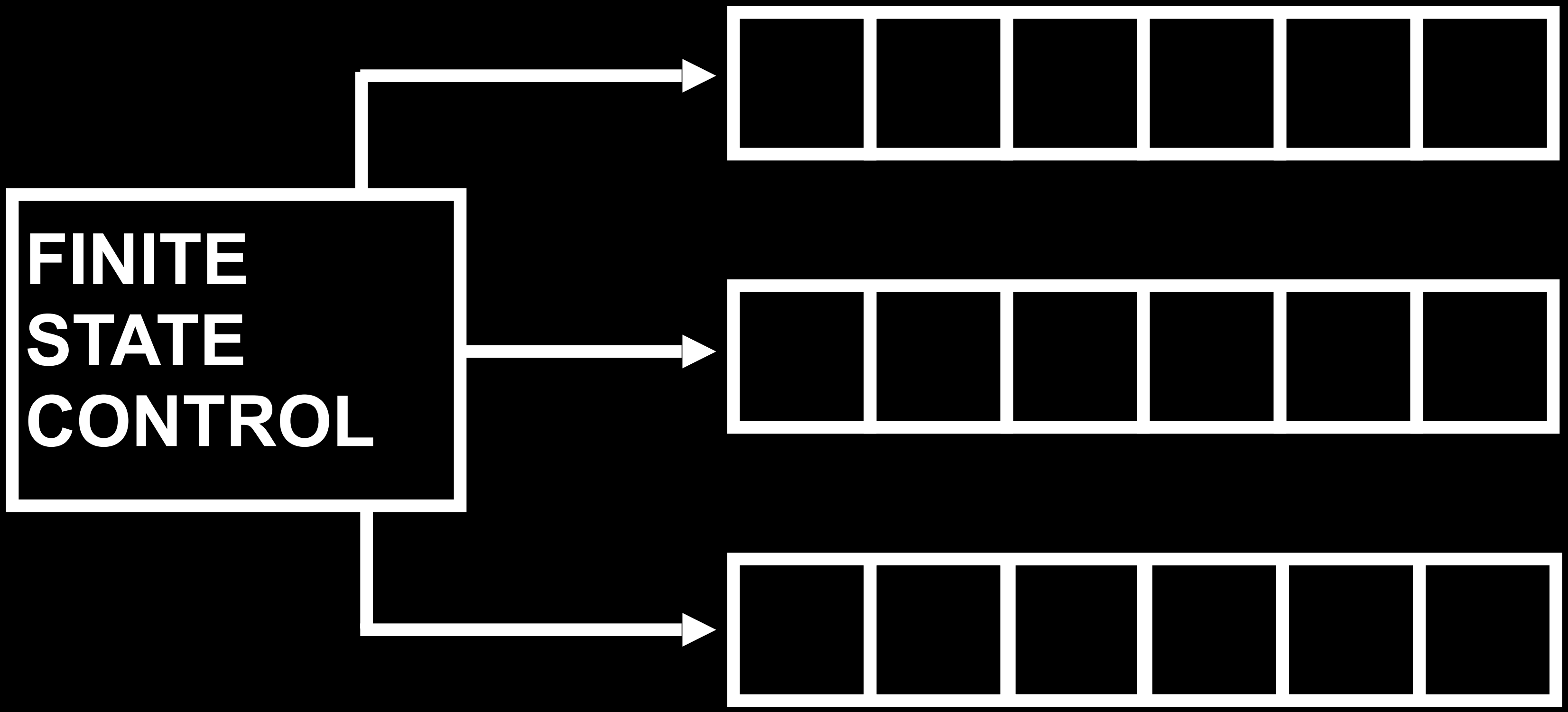
Scan all 0s and copy them to the second tape. Scan all 1s, crossing off a 0 from the second tape for each 1.

Different models of computation yield different running times for the same language!

Theorem: Let t(n) be a function such that t(n) ≥ n. Then every t(n)-time multi-tape TM has an equivalent $O(t(n)^2)$ single tape TM

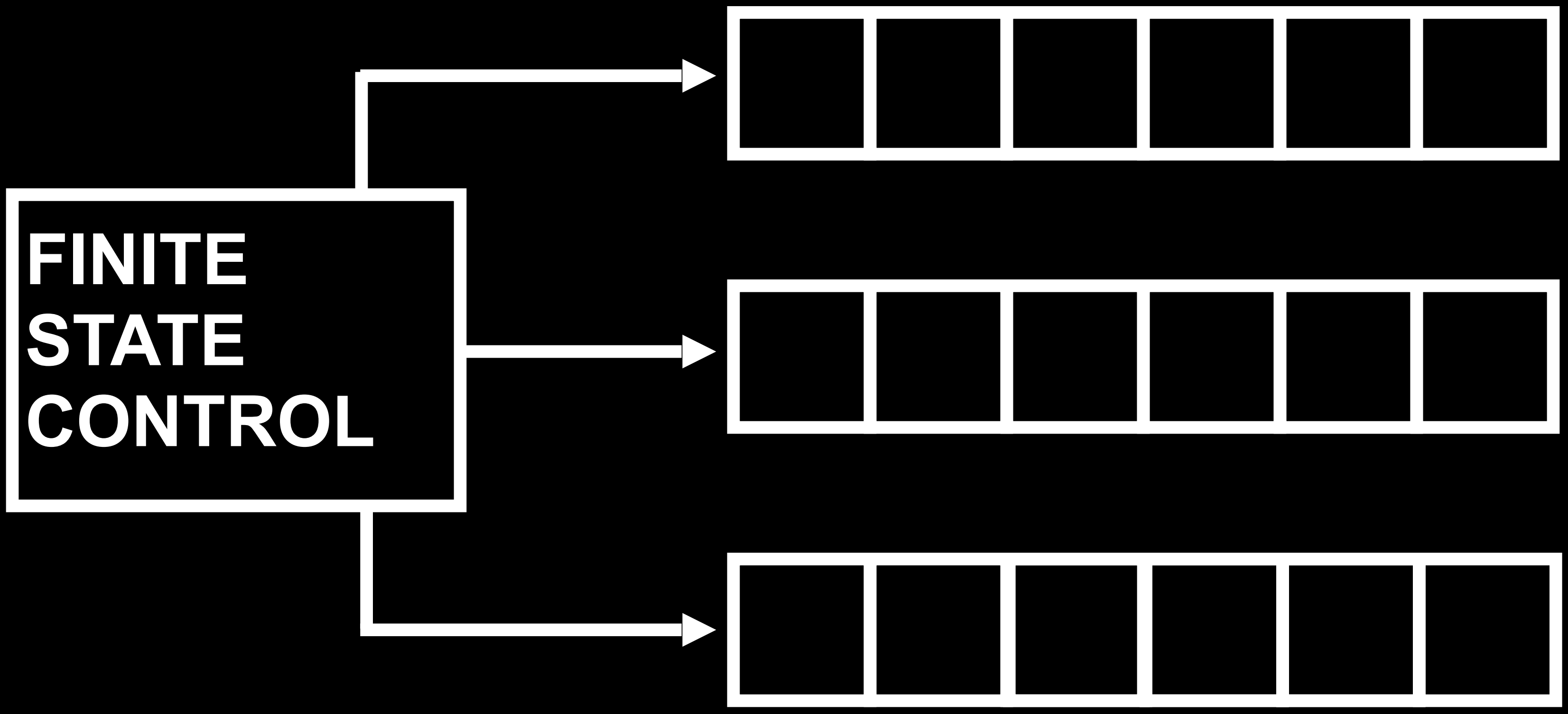
> **Claim:** Simulating each step in the multi-tape machine uses at most O(t(n)) steps on a single-tape machine.
> Hence total time of simulation is $O(t(n)^2)$.

MULTITAPE TURING MACHINES



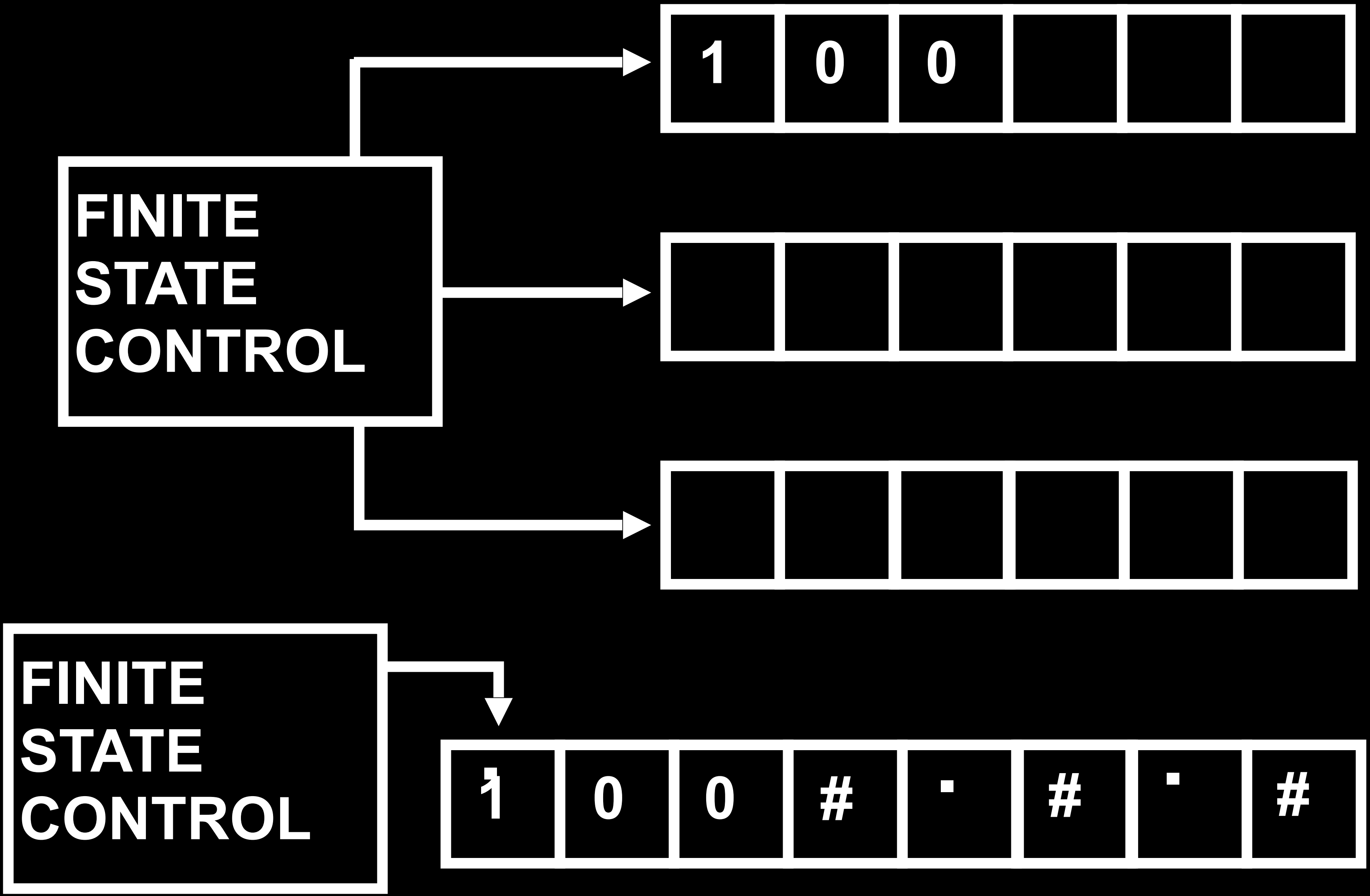
$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine

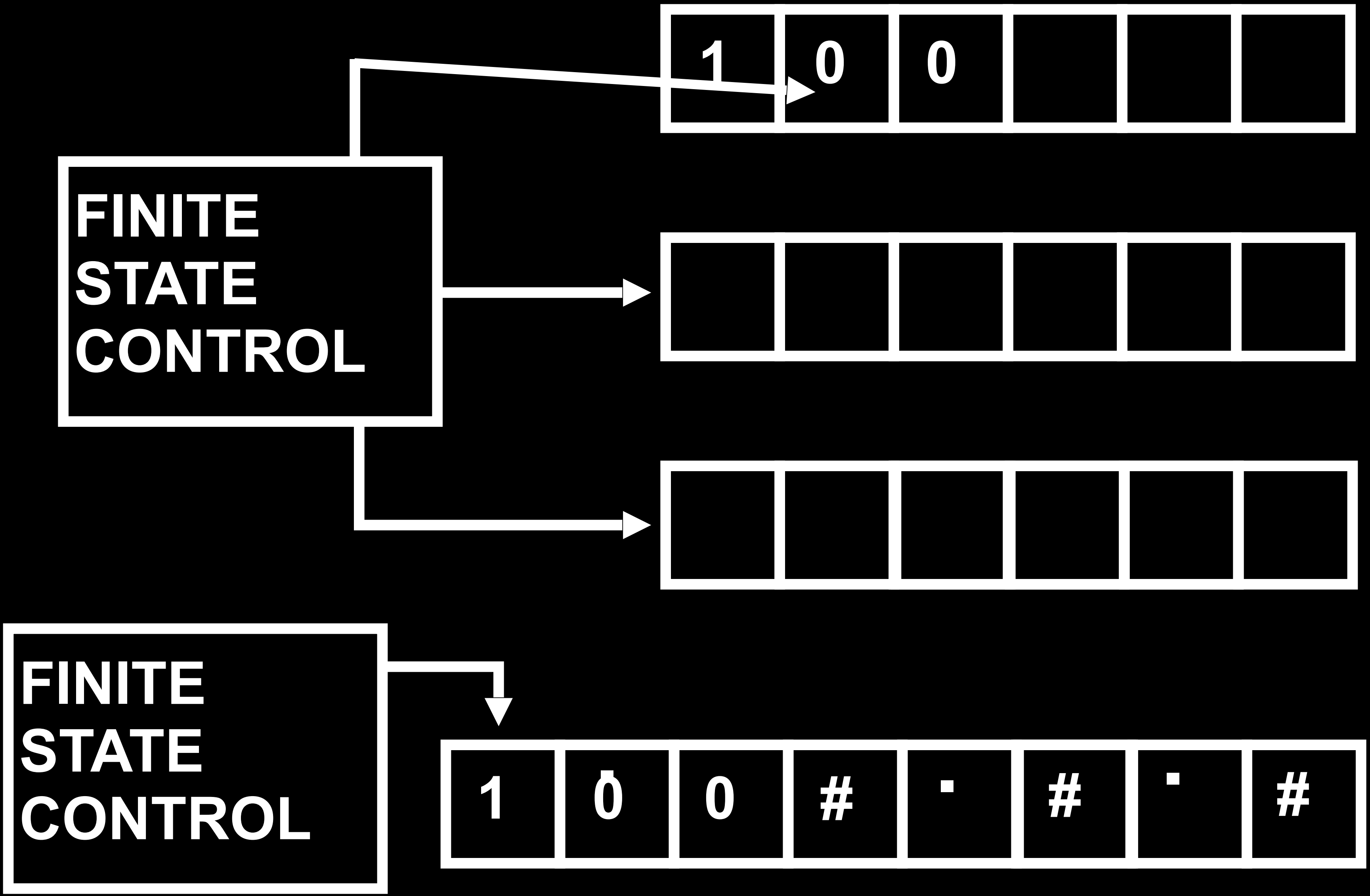


$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



Analysis: (Note, k, the # of tapes, is fixed.)

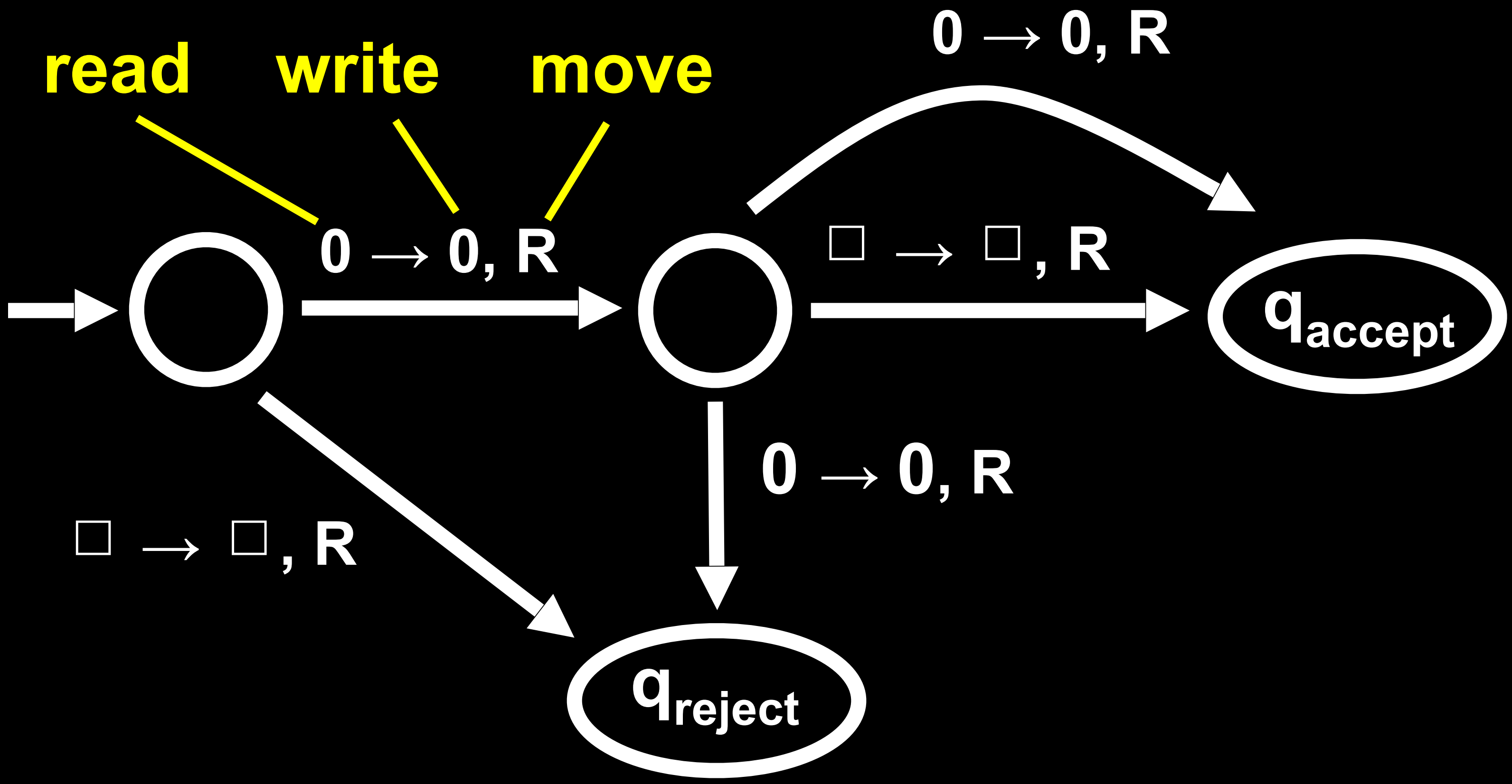
Let S be simulator

- Put S's tape in proper format: O(n) steps
- Two scans to simulate one step,

  1. to obtain info for next move O(t(n)) steps, why?
  2. to simulate it (may need to shift everything over to right possibly k times): O(t(n)) steps, why?

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

NON-DETERMINISTIC TURING MACHINES AND NP



read
write
move
0 → 0, R
0 → 0, R
□ → □, R
q_{accept}
0 → 0, R
□ → □, R
q_{reject}

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 $q_{reject} \neq q_{accept}$

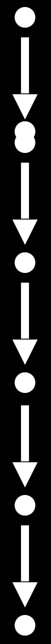
NON-DETERMINISTIC TMs

…are just like standard TMs, except:

1. The machine may proceed according to several possibilities

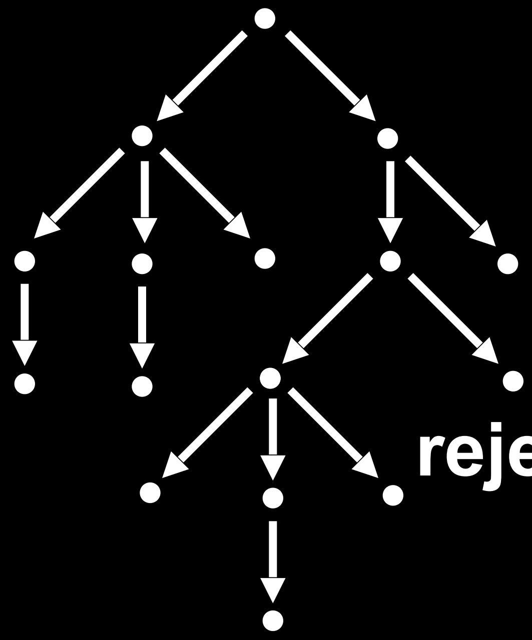
2. The machine accepts a string if there exists a path from start configuration to an accepting configuration

Deterministic Computation



accept or reject

Non-Deterministic Computation

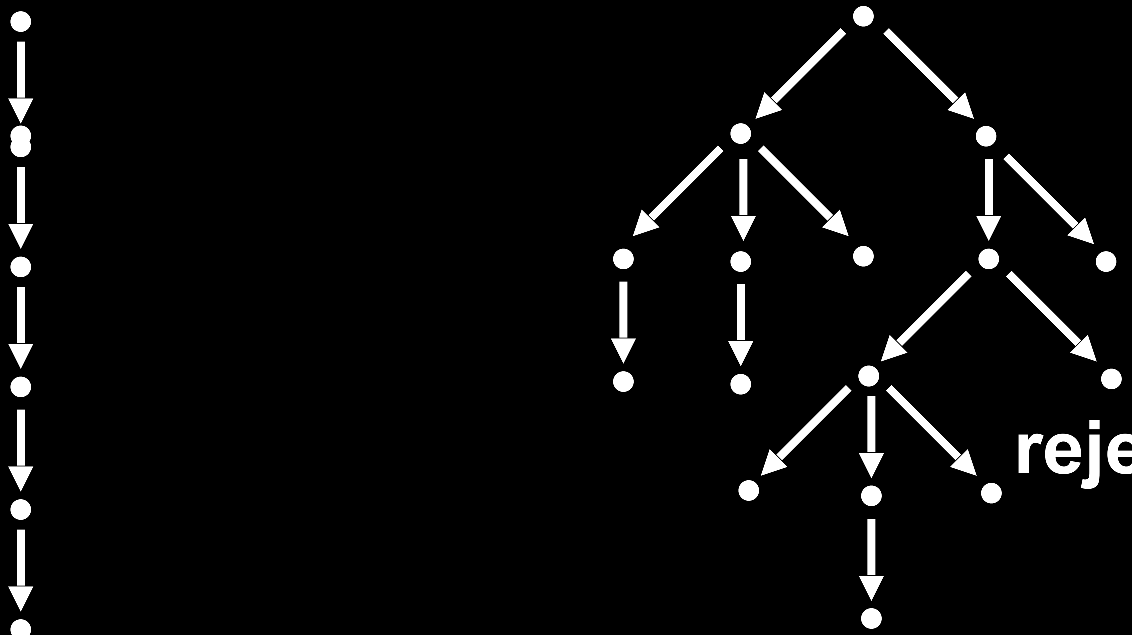


reject

accept

Deterministic Computation

Non-Deterministic Computation



reject

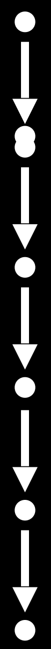
accept or reject

accept

Definition: **Let M be a NTM that is a decider** (Ie all branches halt on all inputs).

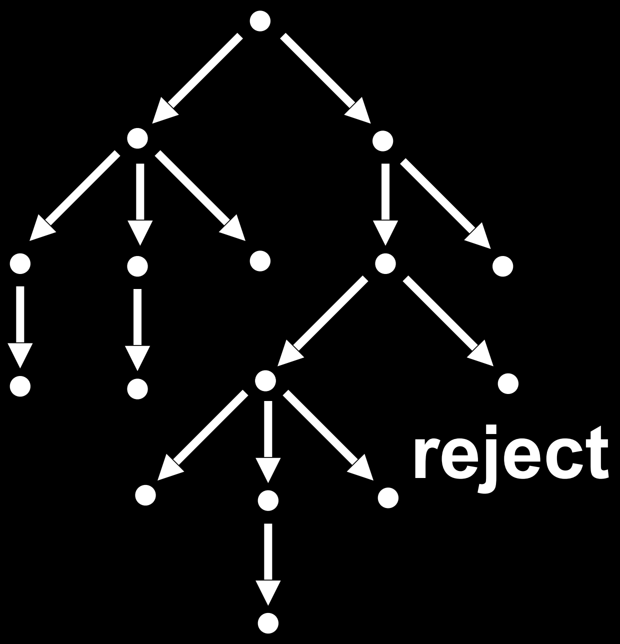
The **running time** or **time-complexity** of **M** is the function f : N → N, where f(n) is the maximum number of steps that **M** uses ***on any branch of its computation on any input of length n.***

Deterministic Computation



accept or reject

Non-Deterministic Computation



accept

Theorem: Let t(n) be a function such that t(n) ≥ n. Then every t(n)-time nondeterministic single-tape TM has an equivalent $2^{O(t(n))}$ deterministic single tape TM

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

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

BOOLEAN FORMULAS

logical operations

parentheses

A **satisfying assignment** is a setting of the variables that makes the formula true

$$\phi = (\neg x \land y) \lor z$$

$x = 1, y = 1, z = 1$ is a satisfying assignment for ϕ

variables

$$\neg(x \lor y) \land (z \land \neg x)$$

0 0 1 0

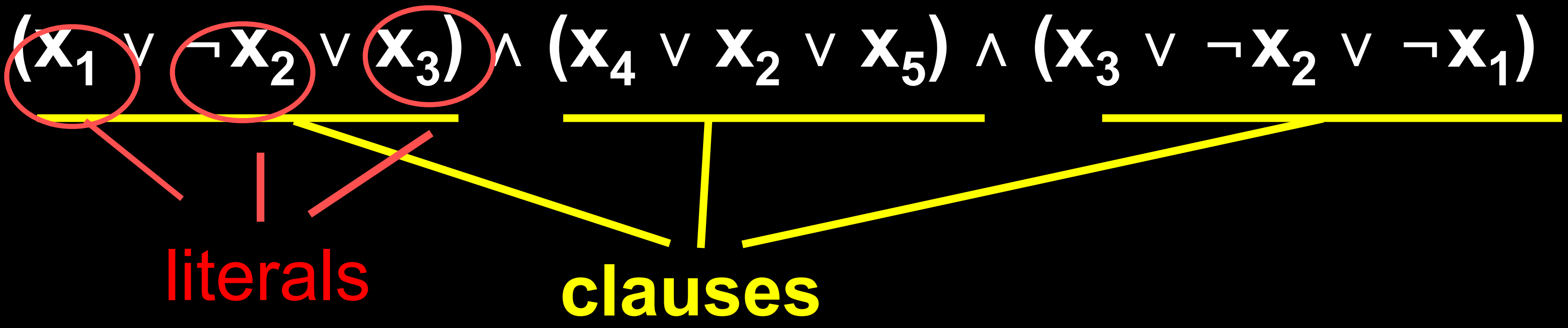
A Boolean formula is **satisfiable** if there exists a satisfying assignment for it

YES $a \wedge b \wedge c \wedge \neg d$

NO $\neg(x \vee y) \wedge x$

SAT = { ϕ | ϕ is a satisfiable Boolean formula }

A **3cnf-formula** is of the form:



$$(x_1 \lor \neg x_2 \lor x_3) \land (x_4 \lor x_2 \lor x_5) \land (x_3 \lor \neg x_2 \lor \neg x_1)$$

literals

clauses

YES $(x_1 \lor \neg x_2 \lor x_1)$

NO $(x_3 \lor x_1) \land (x_3 \lor \neg x_2 \lor \neg x_1)$

NO $(x_1 \lor x_2 \lor x_3) \land (\neg x_4 \lor x_2 \lor x_1) \lor (x_3 \lor x_1 \lor \neg x_1)$

NO $(x_1 \lor \neg x_2 \lor x_3) \land (x_3 \land \neg x_2 \land \neg x_1)$

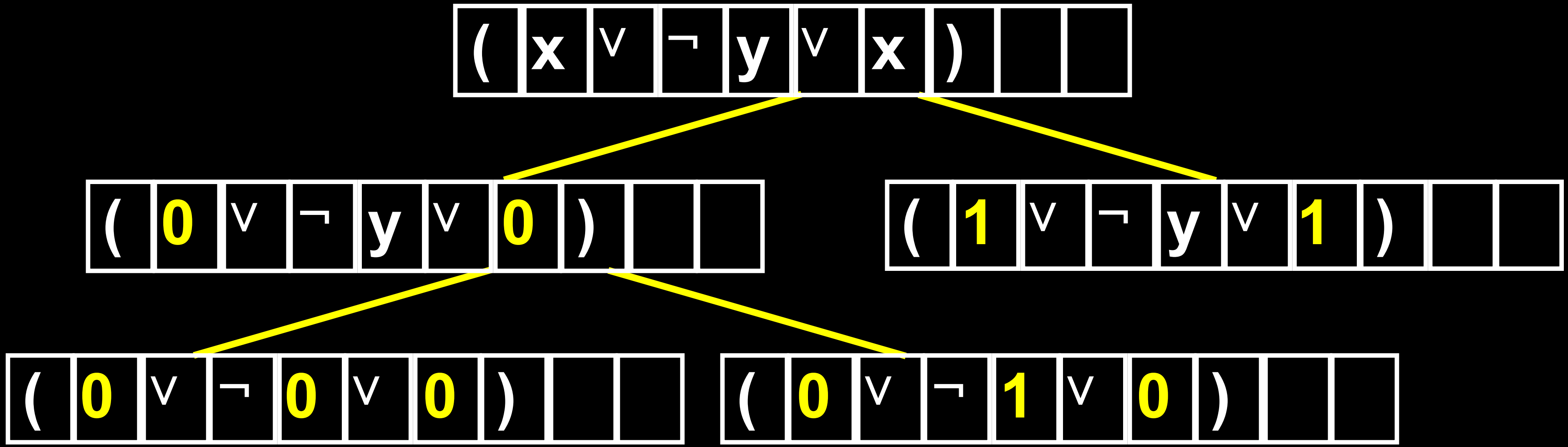
3SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

3SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

Theorem: 3SAT $\in$ NTIME(n^2)

On input ϕ:

1. Check if the formula is in 3cnf
2. For each variable, non-deterministically substitute it with 0 or 1



(x ∨ ¬ y ∨ x)

(0 ∨ ¬ y ∨ 0)

(1 ∨ ¬ y ∨ 1)

(0 ∨ ¬ 0 ∨ 0)

(0 ∨ ¬ 1 ∨ 0)

3. Test if the assignment satisfies ϕ

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

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

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

Proof:

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

then $L \in NP$

Because we can guess y and then run V

(2) If $L \in NP$ then

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

Let N be a non-deterministic poly-time TM that decides L and define V(x,y) to accept if y is an accepting computation history of N on x

3SAT = { ϕ | $\exists$y such that y is a satisfying assignment to ϕ and ϕ is in 3cnf }

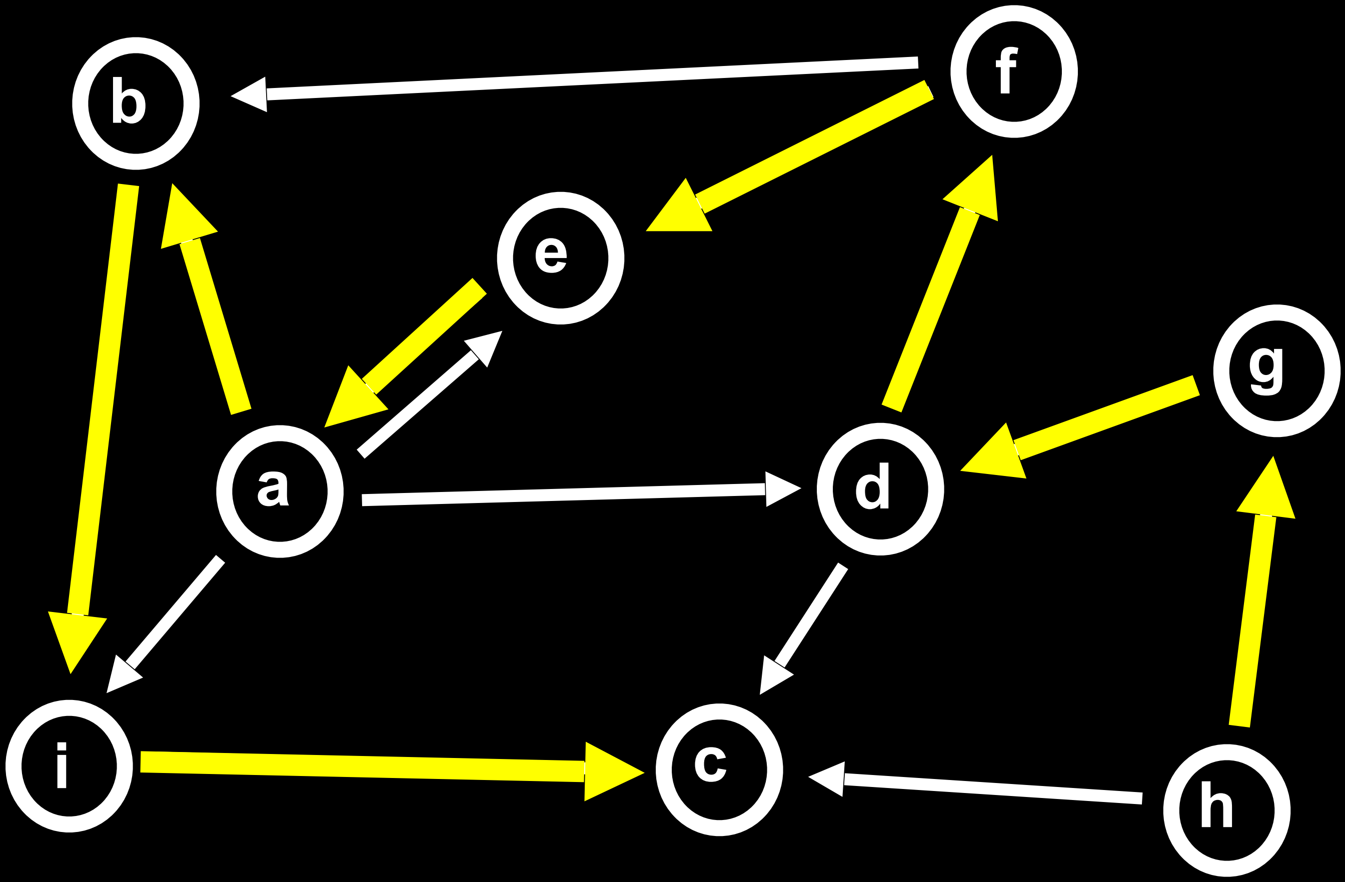
SAT = { ϕ | $\exists$y such that y is a satisfying assignment to ϕ }

A language is in NP if and only if there exist polynomial-length certificates* for membership to the language

SAT is in NP because a satisfying assignment is a polynomial-length certificate that a formula is satisfiable

* that can be verified in poly-time

HAMILTONIAN PATHS

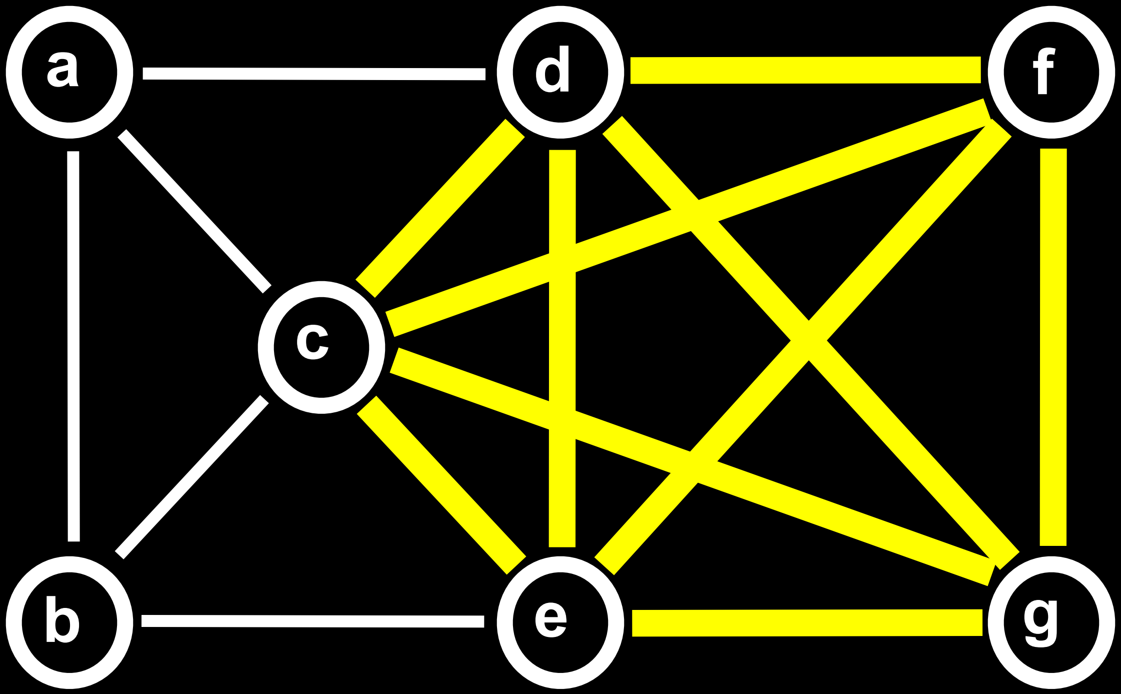


HAMPATH = { (G,s,t) | G is a directed graph with a Hamiltonian path from s to t }

Theorem: HAMPATH $\in$ NP

The Hamilton path itself is a certificate

K-CLIQUES



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

Theorem: CLIQUE $\in$ NP

The k-clique itself is a certificate

NP = all the problems for which once you have the answer it is easy (i.e. efficient) to verify

\$\$\$

P = NP ?

\$\$\$

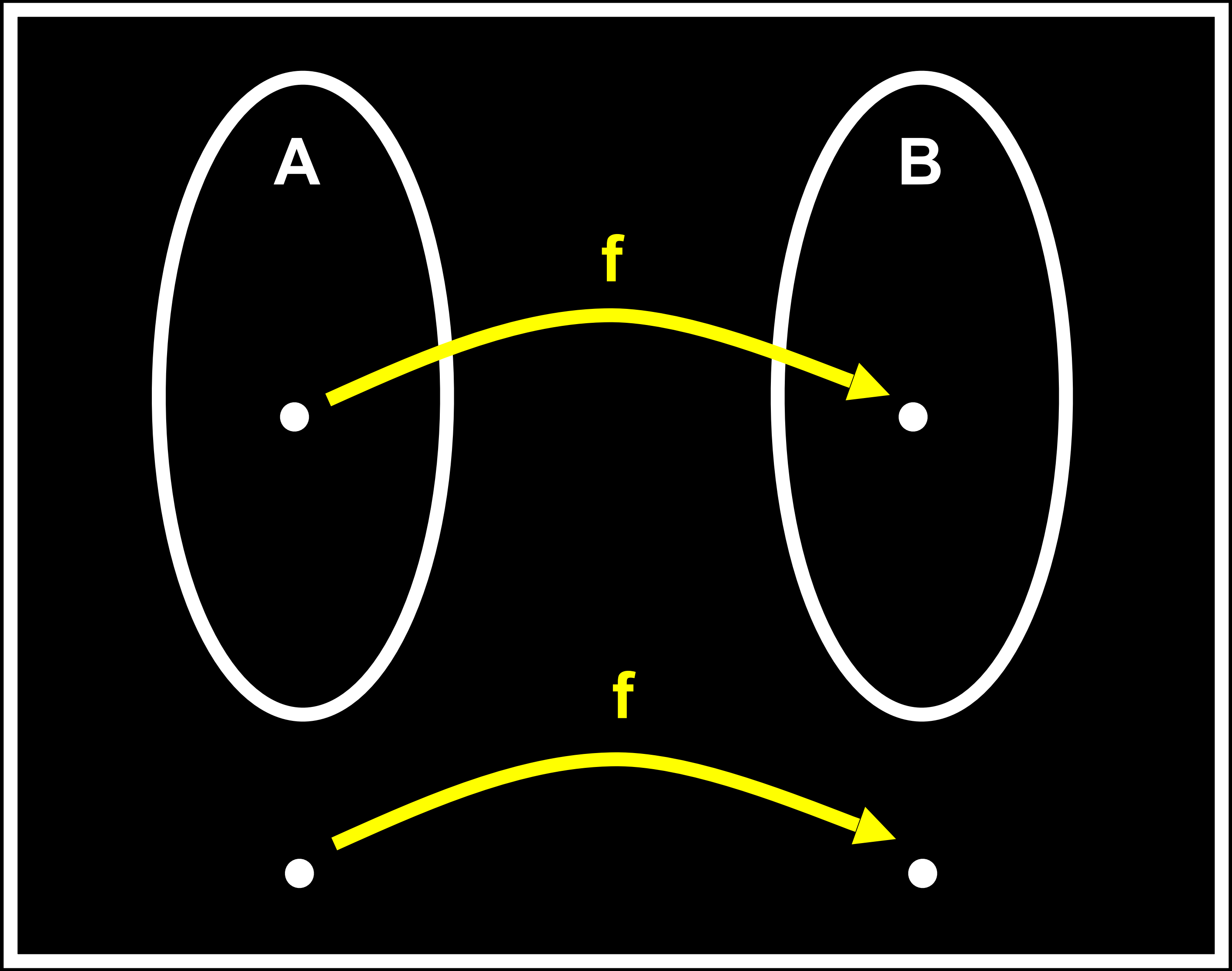
POLY-TIME REDUCIBILITY

$f : \Sigma^* \rightarrow \Sigma^*$ 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


A
B
f
f

Theorem: If $A \leq_P B$ and $B \in P$, then $A \in P$

Proof: Let M_B be a poly-time (deterministic) TM that decides B and let f be a poly-time reduction from A to B

We build a machine M_A that decides A as follows:

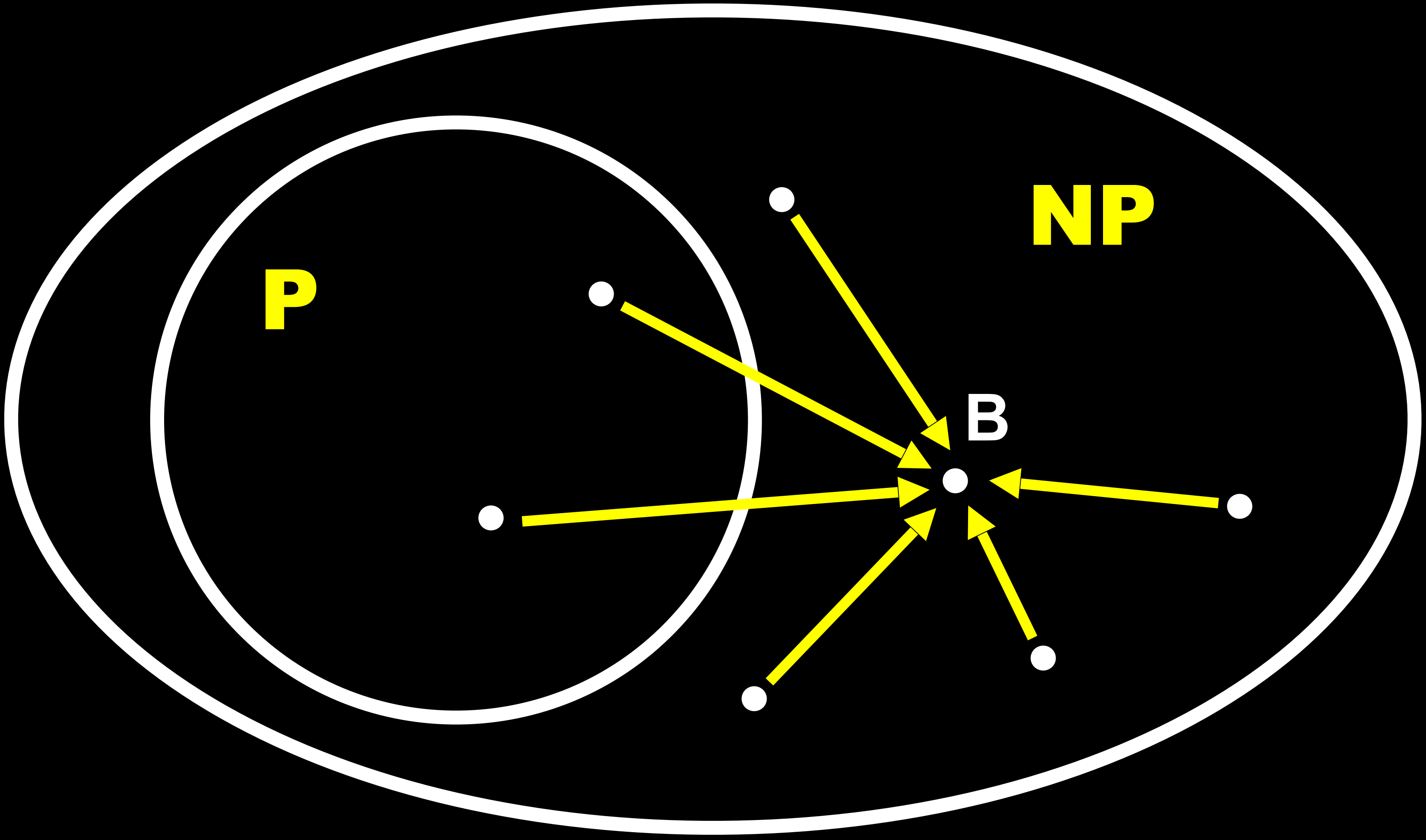
On input w:

1. Compute f(w)
2. Run M_B on f(w)

Definition: A language B is NP-complete if:

1. $B \in NP$
2. Every A in NP is poly-time reducible to B (i.e. B is NP-hard)

Suppose B is NP-Complete



So, if B is NP-Complete and B ∈ P then NP = P. Why?

Theorem (Cook-Levin): SAT is NP-complete

Corollary: SAT $\in$ P if and only if P = NP

WWW.FLAC.WS

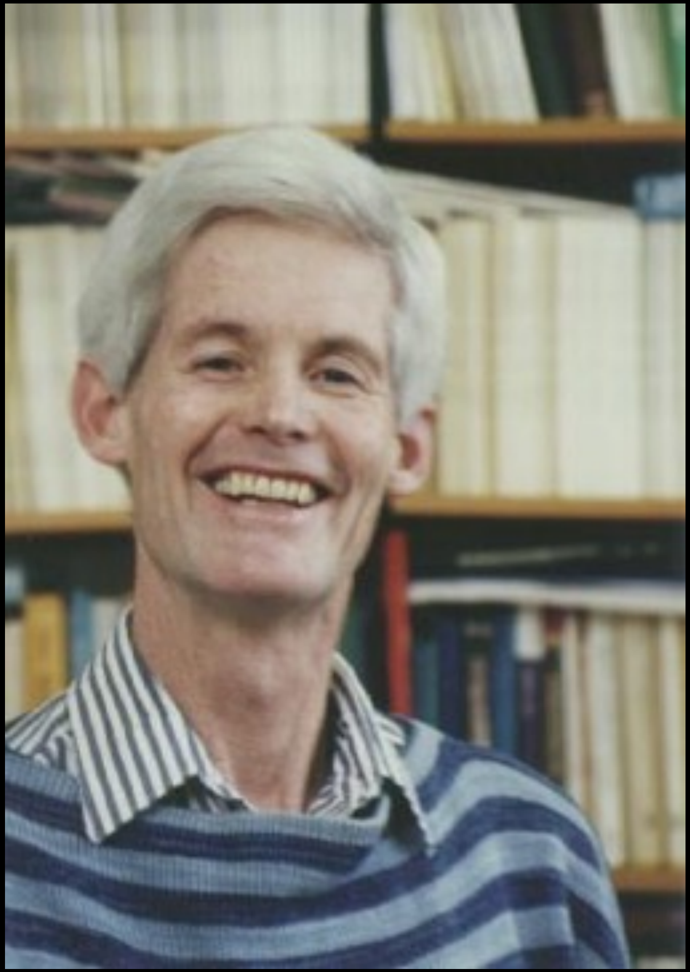
Read Chapter 7.3 of the book for next time

NP-COMPLETENESS:
THE COOK-LEVIN THEOREM

Theorem (Cook-Levin.'71): SAT is NP-complete

Corollary: SAT $\in$ P if and only if P = NP

Leonid Levin



Steve Cook

Theorem (Cook-Levin): SAT is NP-complete

Proof:

(1) SAT $\in$ NP

(2) Every language A in NP is polynomial time reducible to SAT

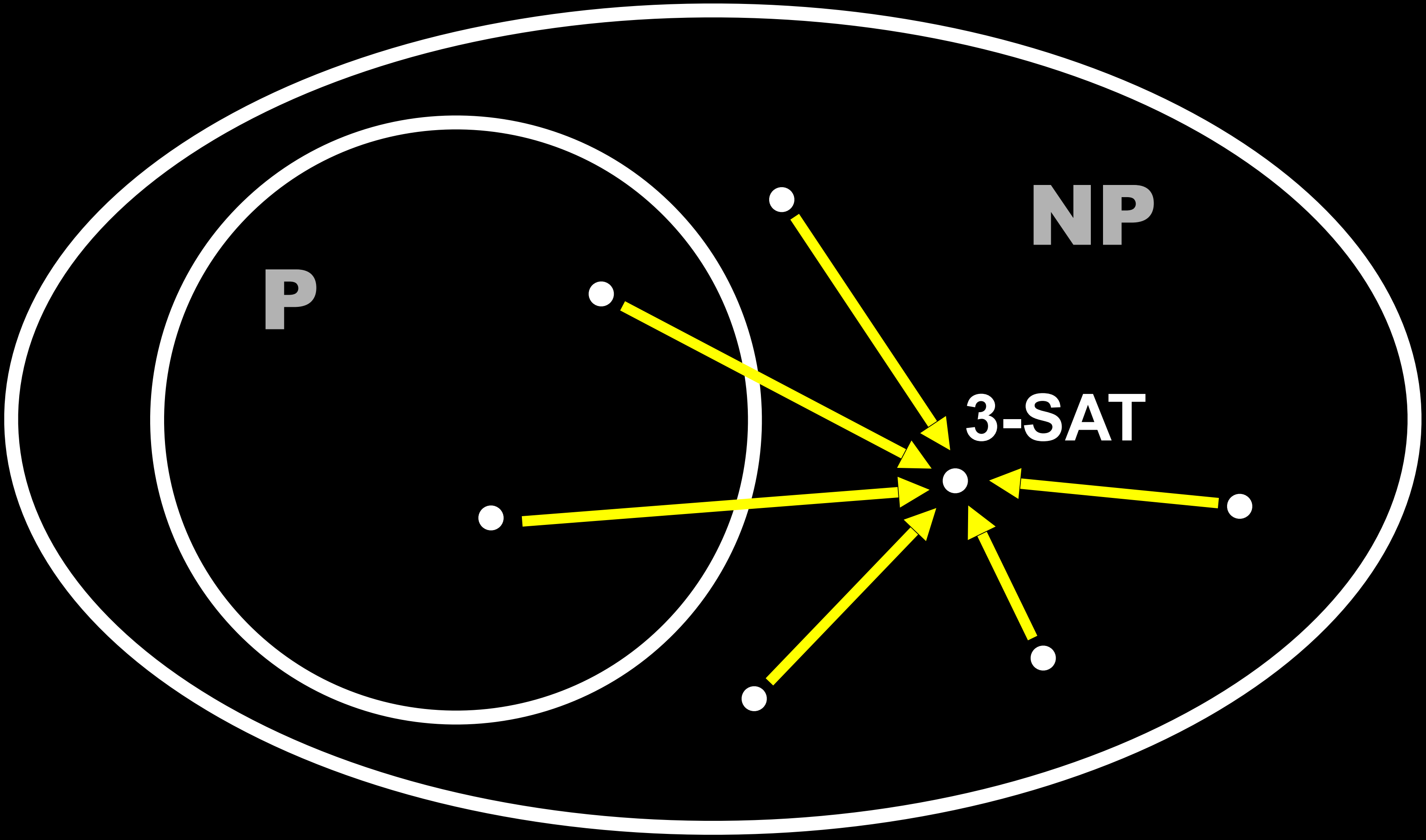
We build a poly-time reduction from A to SAT

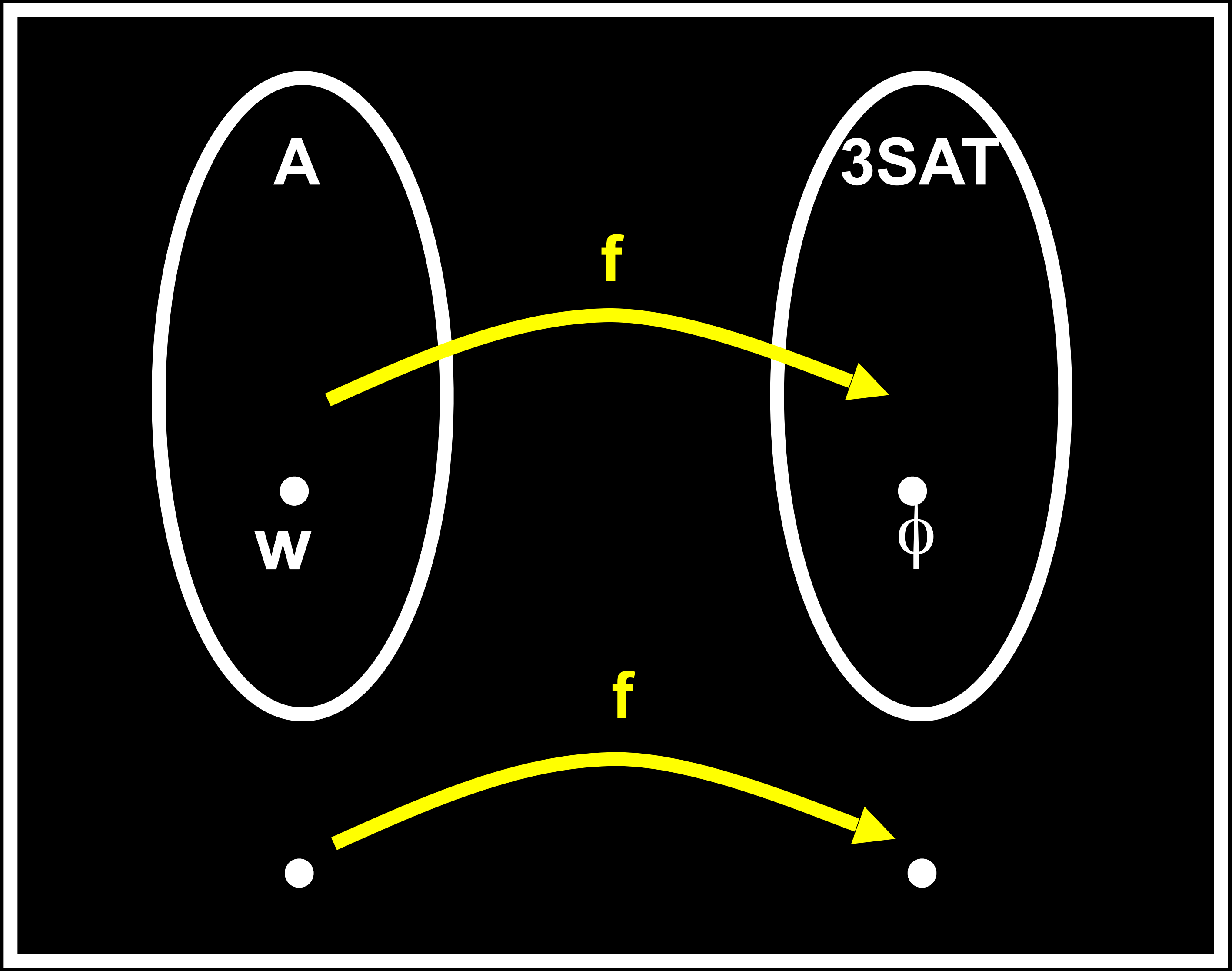
The reduction turns a string w into a 3-cnf formula ϕ such that w $\in$ A iff $\phi \in$ 3-SAT.

ϕ will *simulate* the NP machine N for A on w.

Let N be a non-deterministic TM that decides A in time n^k How do we know N exists?

So proof will also show:
3-SAT is NP-Complete

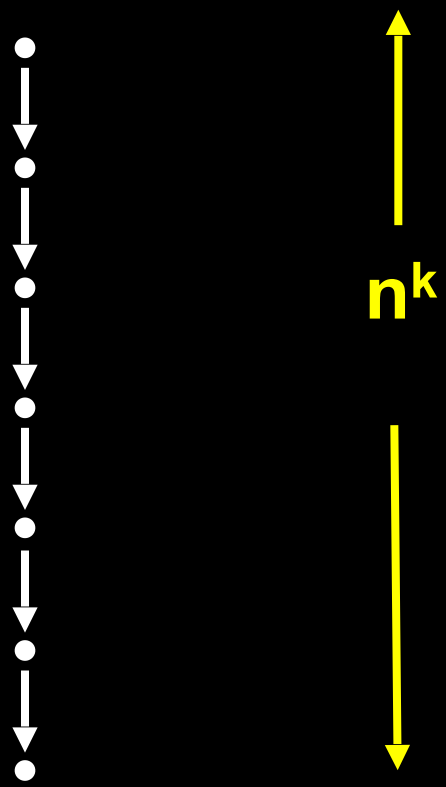




The reduction f turns a string w into a 3-cnf formula ϕ such that: w $\in$ A $\Leftrightarrow$ $\phi \in$ 3SAT.

ϕ will “simulate” the NP machine N for A on w.

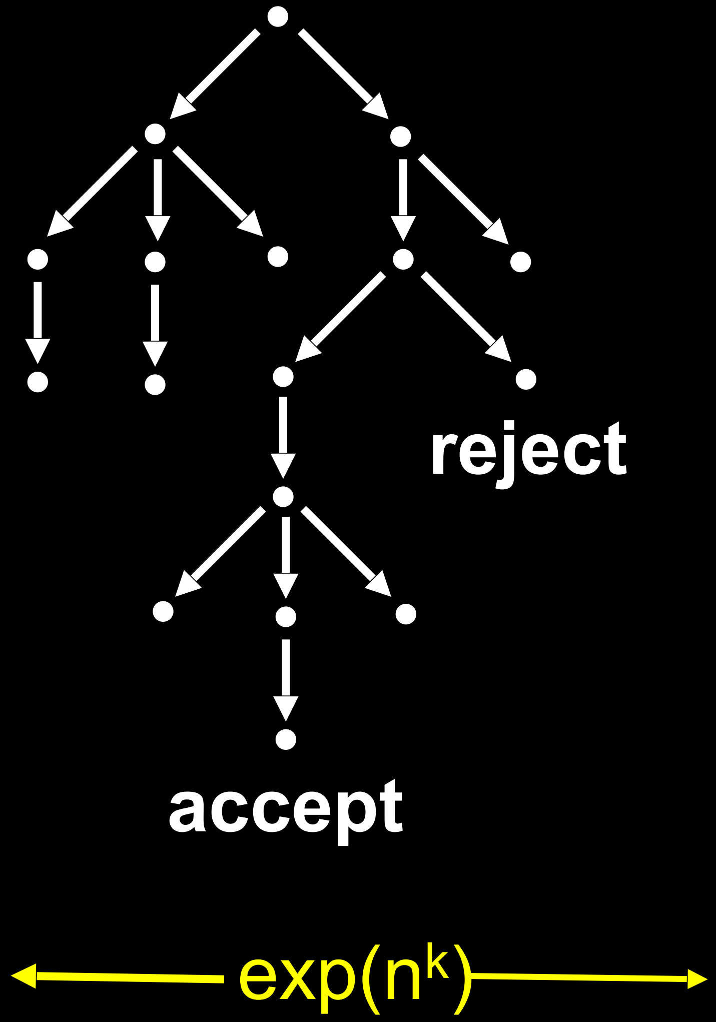
Deterministic Computation



n^k

accept or reject

Non-Deterministic Computation



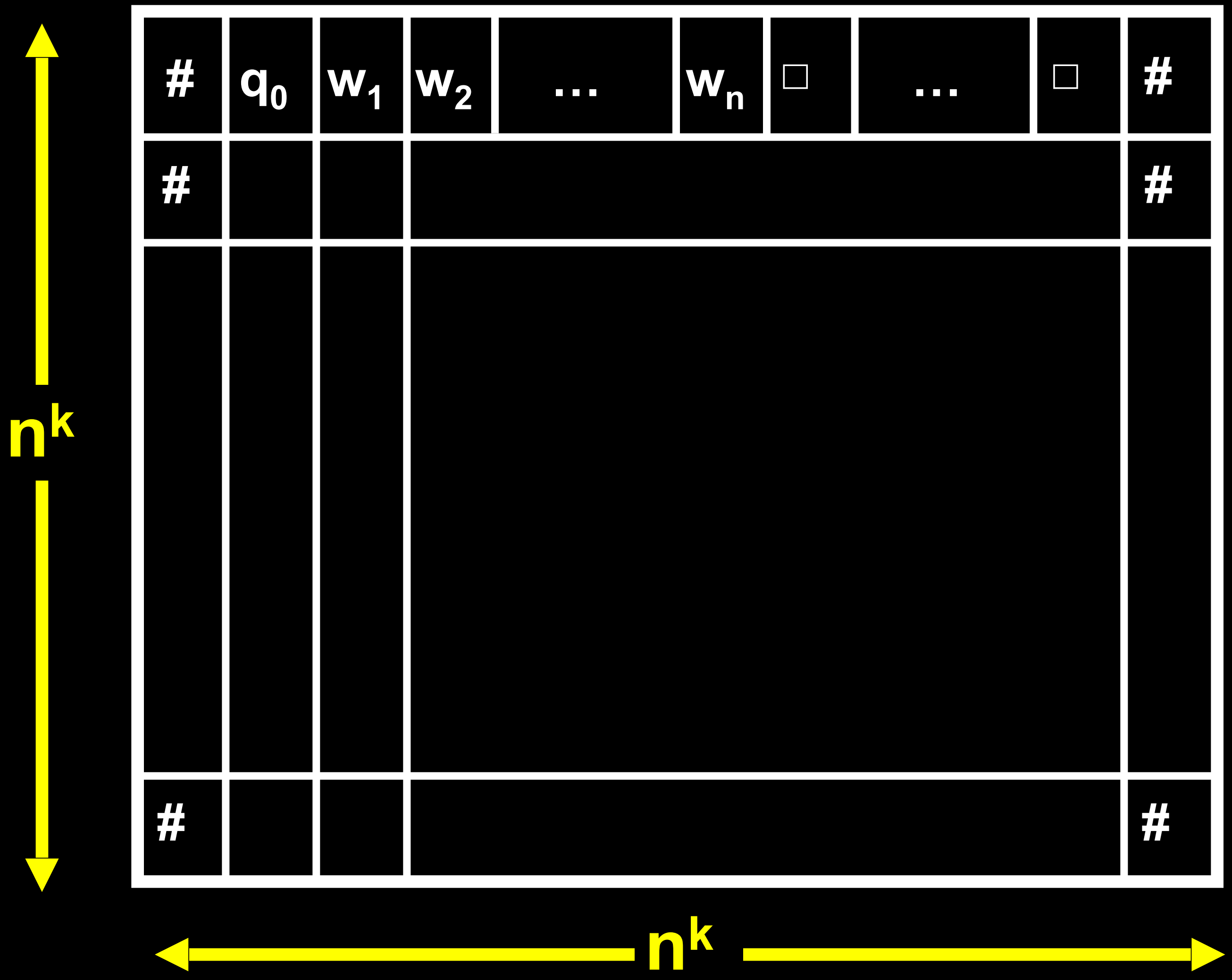
reject

accept

$\exp(n^k)$

Suppose A ∈ NTIME(n^k) and let N be an NP machine for A.

A tableau for N on w is an $n^k \times n^k$ table whose rows are the configurations of *some* possible computation of N on input w.



#	q_0	w_1	w_2	…	w_n	□	…	□	#
#									#
#									#

A tableau is accepting if any row of the tableau is an accepting configuration

Determining whether N accepts w is equivalent to determining whether there is an accepting tableau for N on w

Given w, our 3cnf-formula ϕ will describe a *generic* tableau for N on w (in fact, essentially *generic* for N on any string w of length n).

The 3cnf formula ϕ will be satisfiable *if and only if* there is an accepting tableau for N on w.

VARIABLES of ϕ

Let $C = Q \cup \Gamma \cup \{ \# \}$

Each of the $(n^k)^2$ entries of a tableau is a cell

cell[i,j] = the cell at row i and column j

For each i and j ($1 \le i, j \le n^k$) and for each $s \in C$ we have a variable $x_{i,j,s}$

# variables = $|C|n^{2k}$, ie $O(n^{2k})$, since $|C|$ only depends on N

These are the variables of ϕ and represent the contents of the cells

We will have: $x_{i,j,s} = 1 \Leftrightarrow$ cell[i,j] = s

$$x_{i,j,s} = 1$$

means

$$\text{cell}[\ i,\ j\ ] = s$$

We now design ϕ so that a satisfying assignment to the variables $x_{i,j,s}$ corresponds to an accepting tableau for N on w

The formula ϕ will be the AND of four parts:

$$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$$

ϕ_{cell} ensures that for each i,j, exactly one $x_{i,j,s} = 1$

ϕ_{start} ensures that the first row of the table is the *starting (initial)* **configuration of N on w**

ϕ_{accept} ensures* that an accepting configuration occurs somewhere in the table

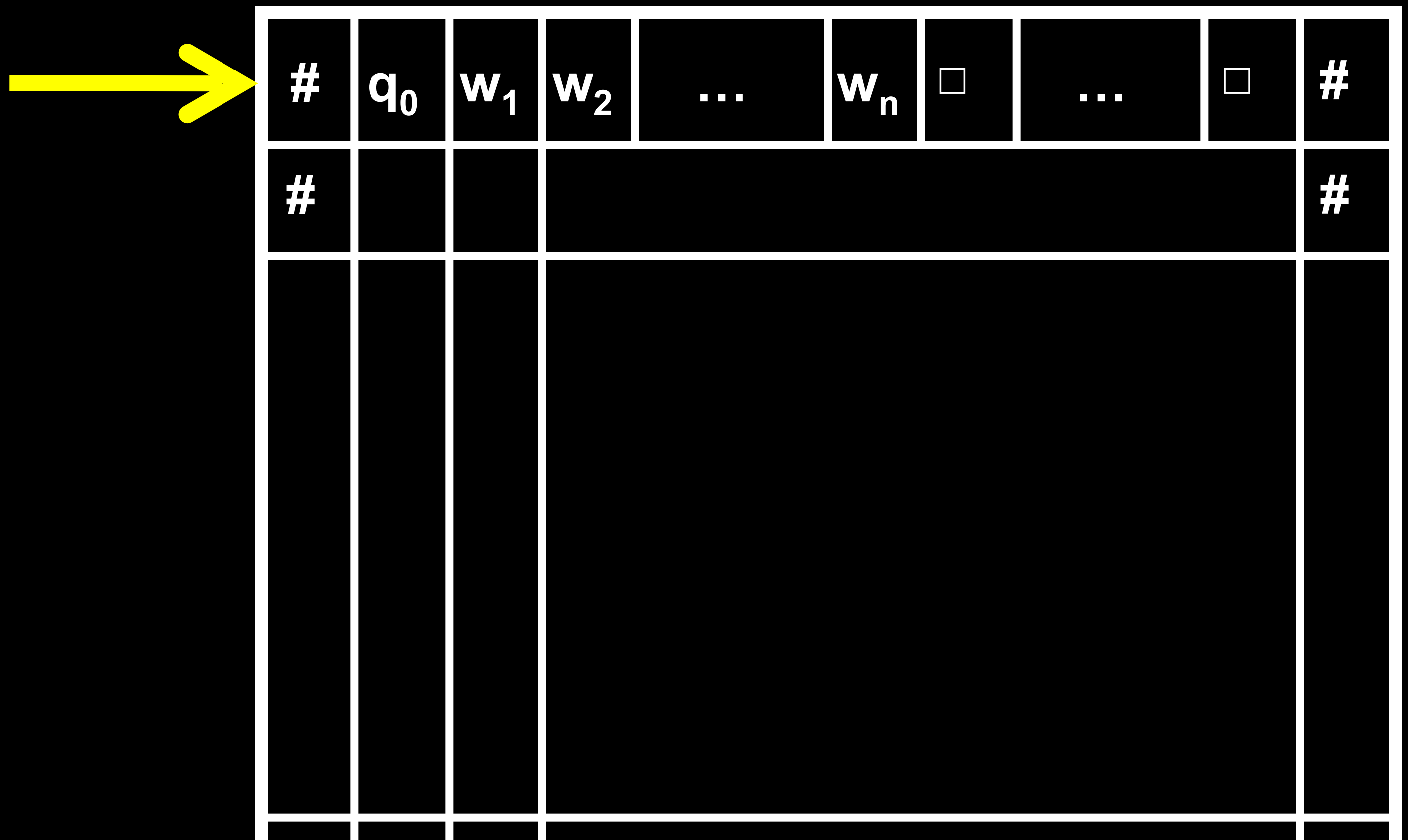
ϕ_{move} ensures* that every row is a configuration that legally follows from the previous config

*if the other components of ϕ **hold**

ϕ_{cell} ensures that for each i,j, exactly one $x_{i,j,s} = 1$

$$\phi_{cell} = \bigwedge_{1 \leq i,\, j \leq n^k} \left[\underbrace{\left(\bigvee_{s \in C} x_{i,j,s} \right)}_{\text{at least one variable is turned on}} \wedge \underbrace{\left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right)}_{\text{at most one variable is turned on}} \right]$$

$$\phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge$$
$$x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \wedge x_{1,n+2,w_n} \wedge$$
$$x_{1,n+3,\square} \wedge \dots \wedge x_{1,n^k-1,\square} \wedge x_{1,n^k,\#}$$



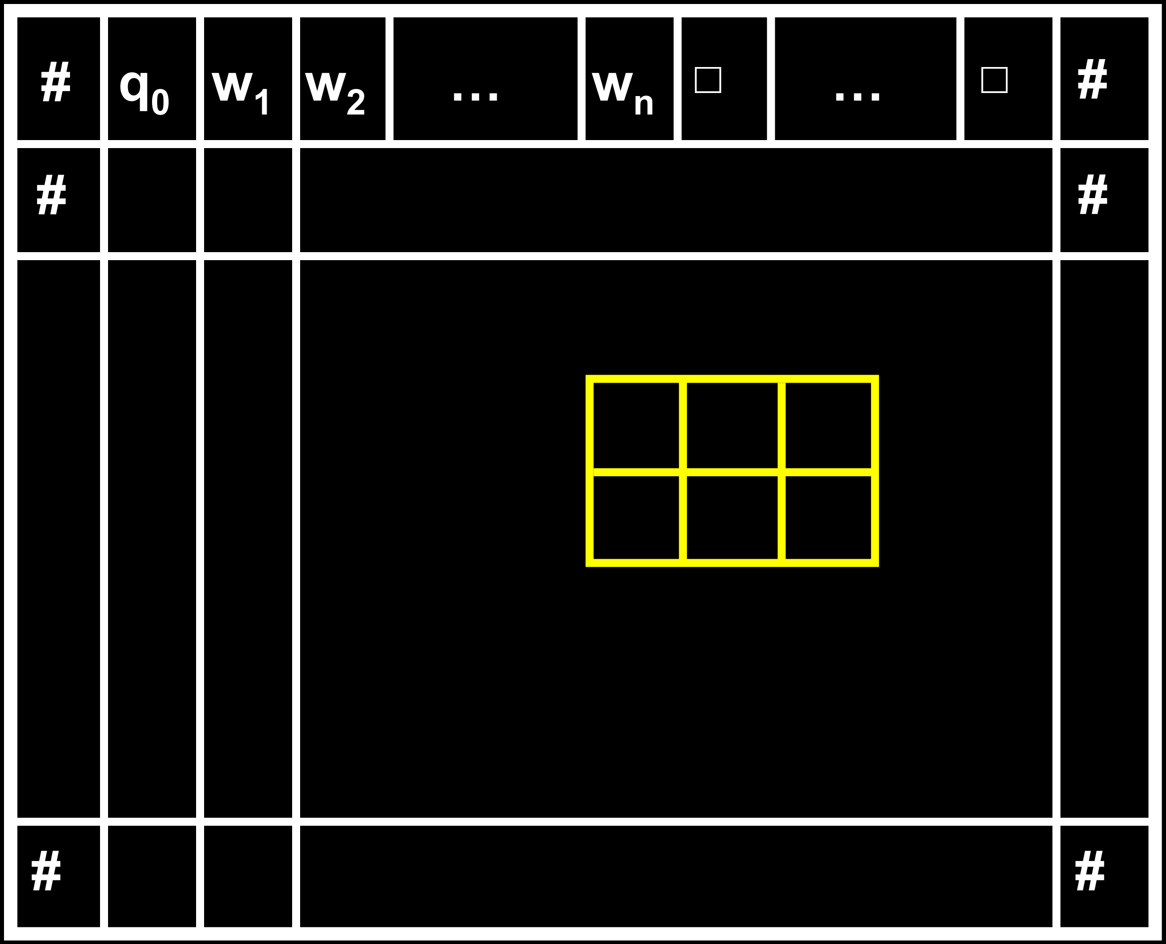
#	q_0	w_1	w_2	…	w_n	□	…	□	#
#									#

ϕ_{accept} ensures that an accepting configuration occurs somewhere in the table

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,\, j \leq n^k} x_{i,j,q_{\text{accept}}}$$

ϕ_{move} ensures that every row is a configuration that legally follows from the previous

It works by ensuring that each 2 × 3 "window" of cells is legal (does not violate N's rules)



#	q_0	w_1	w_2	…	w_n	□	…	□	#
#									#
#									#

If $\delta(q_1,a) = \{(q_1,b,R)\}$ and $\delta(q_1,b) = \{(q_2,c,L), (q_2,a,R)\}$

Which of the following windows are legal:

a	q_1	b
q_2	a	c

a	q_1	b
q_1	a	a

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

b	q_1	b
q_2	b	$_2$

a	b	a
a	a	a

a	q_1	b
a	a	q_2

b	b	b
c	b	b

If $\delta(q_1,a) = \{(q_1,b,R)\}$ and $\delta(q_1,b) = \{(q_2,c,L), (q_2,a,R)\}$

Which of the following windows are legal:

a	q_1	b
q_2	a	c

~~a~~	~~q_1~~	~~b~~
~~q_1~~	~~a~~	~~a~~

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

~~b~~	~~q_1~~	~~b~~
~~q_2~~	~~b~~	~~q_2~~

~~a~~	~~b~~	~~a~~
~~a~~	~~a~~	~~a~~

a	q_1	b
a	a	q_2

b	b	b
c	b	b

CLAIM:

If

- the top row of the tableau is the start configuration, and
- and every window is legal,

Then

each row of the tableau is a configuration that legally follows the preceding one.

CLAIM:

If

- the top row of the tableau is the start configuration, and
- and every window is legal,

Then

each row of the tableau is a configuration that legally follows the preceding one.

	s	

Proof:

In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.

CLAIM:

If

- the top row of the tableau is the start configuration, and
- and every window is legal,

Then

each row of the tableau is a configuration that legally follows the preceding one.

	a	
	a	

Proof:

In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.

Case 1. center cell of window is a non-state symbol and not adjacent to a state symbol

CLAIM:

If

- the top row of the tableau is the start configuration, and
- and every window is legal,

Then

each row of the tableau is a configuration that legally follows the preceding one.

	a	
	a	

	q	
ok	ok	ok

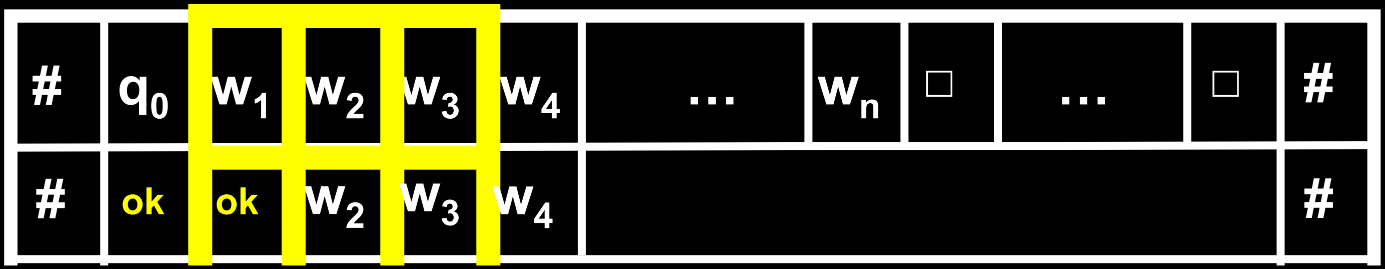
Proof:

In upper configuration, every cell that doesn't contain the boundary symbol #, is the center top cell of a window.

Case 1. center cell of window is a non-state symbol and not adjacent to a state symbol

Case 2. center cell of window is a state symbol

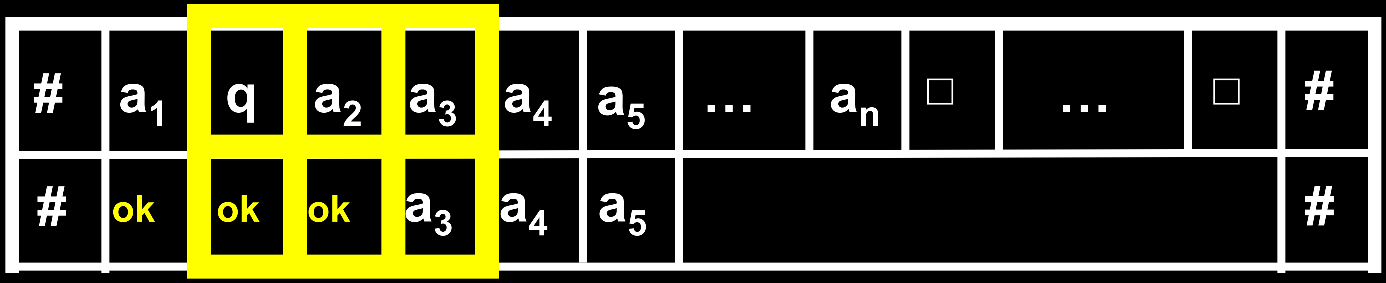
#	q_0	w_1	w_2	w_3	w_4	…	w_n	□	…	□	#
#	ok	ok	w_2	w_3	w_4						#



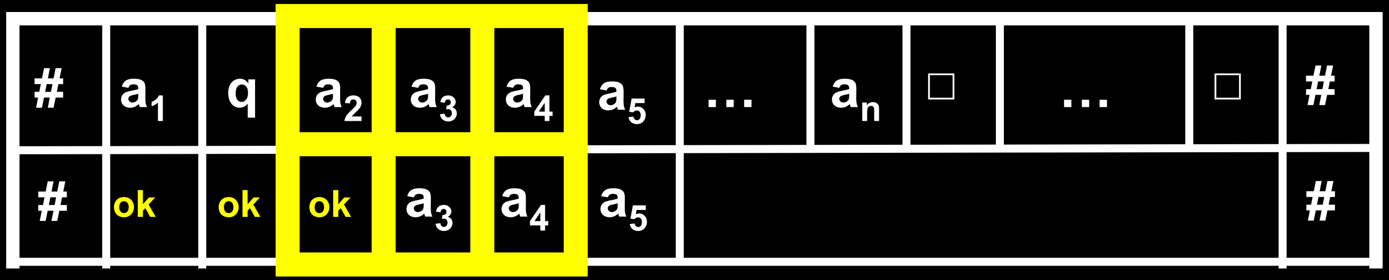
#	q_0	w_1	w_2	w_3	w_4	…	w_n	□	…	□	#
#	ok	ok	w_2	w_3	w_4						#

#	q_0	w_1	w_2	w_3	w_4	…	w_n	□	…	□	#
#	ok	ok	w_2	w_3	w_4						#

#	a_1	q	a_2	a_3	a_4	a_5	…	a_n	□	…	□	#
#	ok	ok	ok	a_3	a_4	a_5						#



#	a_1	q	a_2	a_3	a_4	a_5	…	a_n	□	…	□	#
#	ok	ok	ok	a_3	a_4	a_5						#



#	a_1	q	a_2	a_3	a_4	a_5	…	a_n	□	…	□	#
#	ok	ok	ok	a_3	a_4	a_5						#

So the lower configuration follows from the upper!!!

	col. j-1	col. j	col. j+1
row i	**(i,j-1)** a_1	**(i,j)** a_2	**(i,j+1)** a_3
row i+1	**(i+1,j-1)** a_4	**(i+1,j)** a_5	**(i+1,j+ 1)** a_6

The (i,j) Window

$$\phi_{\text{move}} = \bigwedge_{1 \leq i,\, j \leq n^k} (\text{ the } (i, j) \text{ window is legal })$$

the (i, j) window is legal =

$$\bigvee_{\substack{a_1,\, \ldots,\, a_6 \\ \text{is a legal window}}} (\, x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j,+1,a} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a} \,)$$

This is a disjunct over all ($\leq |C|^6$) legal sequences (a_1, …, a_6).

$$\phi_{move} = \bigwedge_{1 \le i, j \le n^k} (\text{ the } (i, j) \text{ window is legal })$$

the (i, j) window is legal =

$$\bigvee_{\substack{a_1, \ldots, a_6 \\ \text{is a legal window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j,+1,a} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a})$$

This is a disjunct over all (≤ $|C|^6$) legal sequences ($a_1, \ldots, a_6$).

This disjunct is satisfiable

⇔

There is *some* assignment to the cells (ie variables) in the window (i,j) that makes the window legal

$$\phi_{move} = \bigwedge_{1 \le i, j \le n^k} (\text{ the } (i, j) \text{ window is legal })$$

the (i, j) window is legal =

$$\bigvee_{\substack{a_1, \ldots, a_6 \\ \text{is a legal window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j,+1,a} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a})$$

This is a disjunct over all ($\le |C|^6$) legal sequences ($a_1, \ldots, a_6$).

So ϕ_{move} is satisfiable

⟺

There is *some* assignment to each of the variables that makes *every* window legal.

$$\phi_{move} = \bigwedge_{1 \le i, j \le n^k} (\text{ the } (i, j) \text{ window is legal })$$

the (i, j) window is legal =

$$\bigvee_{\substack{a_1, \ldots, a_6 \\ \text{is a legal window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j,+1,a} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a})$$

This is a disjunct over all (≤ $|C|^6$) legal sequences ($a_1, \ldots, a_6$).

Can re-write as equivalent conjunct:

$$\equiv \bigwedge_{\substack{a_1, \ldots, a_6 \\ \text{ISN'T a legal window}}} (□_{i,j-1,a_1} \vee □_{i,j,a_2} \vee □_{i,j,+1,a_3} \vee □_{i+1,j-1,a_4} \vee □_{i+1,j,a} \vee □_{i+1,j+1,a_6})$$

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$$

ϕ is satisfiable (ie, there is some assignment to each of the varialbes s.t. ϕ evaluates to 1)

$\Leftrightarrow$

there is some assignment to each of the variables s.t. ϕ_{cell} and ϕ_{start} and ϕ_{accept} and ϕ_{move} each evaluates to 1

$\Leftrightarrow$

There is some assignment of symbols to cells in the tableau such that:

- The first row of the tableau is a start configuration and
- Every row of the tableau is a configuration that follows from the preceding by the rules of N and
- One row is an accepting configuration

$\Leftrightarrow$

There is some accepting computation for N with input w

3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals

3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs

We just need to make those ORs with 3 literals

If a clause has less than three variables:

$a \equiv (a \vee a \vee a)$, $(a \vee b) \equiv (a \vee b \vee b)$

3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals

If a clause has less than three variables:

$a \equiv (a \vee a \vee a)$, $(a \vee b) \equiv (a \vee b \vee b)$

If a clause has more than three variables:

$(a \vee b \vee c \vee d) \equiv (a \vee b \vee z) \wedge (\neg z \vee c \vee d)$

$(a_1 \vee a_2 \vee \ldots \vee a_t) \equiv$

$(a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \ldots$

$$\phi = \phi_{\mathbf{cell}} \wedge \phi_{\mathbf{start}} \wedge \phi_{\mathbf{accept}} \wedge \phi_{\mathbf{move}}$$

WHAT'S THE **LENGTH OF ϕ?**

$$\phi_{cell} = \bigwedge_{1 \leq i,\, j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right]$$

If a clause has less than three variables:

$(a \vee b) = (a \vee b \vee b)$

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,\, j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right]$$

$O(n^{2k})$ clauses

$$\text{Length}(\phi_{\text{cell}}) = O(n^{2k})\ \underbrace{O(\log\,(n))}_{\text{length(indices)}} = O(n^{2k} \log n)$$

$$
\begin{aligned}
\phi_{\mathrm{start}} = \; & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\
& x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\
& x_{1,n+3,\square} \wedge \ldots \wedge x_{1,n^k-1,\square} \wedge x_{1,n^k,\#} \\
= \; & (x_{1,1,\#} \vee x_{1,1,\#} \vee x_{1,1,\#}) \wedge \\
& (x_{1,2,q_0} \vee x_{1,2,q_0} \vee x_{1,2,q_0}) \\
& \wedge \ldots \wedge \\
& (x_{1,n^k,\#} \vee x_{1,n^k,\#} \vee x_{1,n^k,\#})
\end{aligned}
$$

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge$$

$$x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge$$

$$x_{1,n+3,\square} \wedge \ldots \wedge x_{1,n^k-1,\square} \wedge x_{1,n^k,\#}$$

$$O(n^k)$$

$$\phi_{accept} = \bigvee_{1 \le i,\, j \le n^k} x_{i,j,q_{accept}}$$

$$(a_1 \vee a_2 \vee \ldots \vee a_t) =$$

$$(a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \ldots$$

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,\, j \leq n^k} x_{i,j,q_{\text{accept}}}$$

$$O(n^{2k})$$

$$\phi_{move} = \bigwedge_{1 \le i,\, j \le n^k} (\text{ the (i, j) window is legal })$$

the (i, j) window is legal =

$$\bigwedge_{\substack{a_1, \ldots, a_6 \\ \text{ISN'T a legal window}}} (\overline{x}_{i,j-1,a_1} \vee \overline{x}_{i,j,a_2} \vee \overline{x}_{i,j,+1,a_3} \vee \overline{x}_{i+1,j-1,a_4} \vee \overline{x}_{i+1,j,a_5} \vee \overline{x}_{i+1,j+1,a})$$

This is a conjunct over all ($\le |C|^6$) illegal sequences ($a_1, \ldots, a_6$).

$$O(n^{2k})$$

Theorem (Cook-Levin): 3-SAT is NP-complete

Corollary: 3-SAT $\in$ P if and only if P = NP