CS 154

Lecture 8: Recognizability, Decidability, and Diagonalization

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 $\Sigma \subseteq \Gamma$

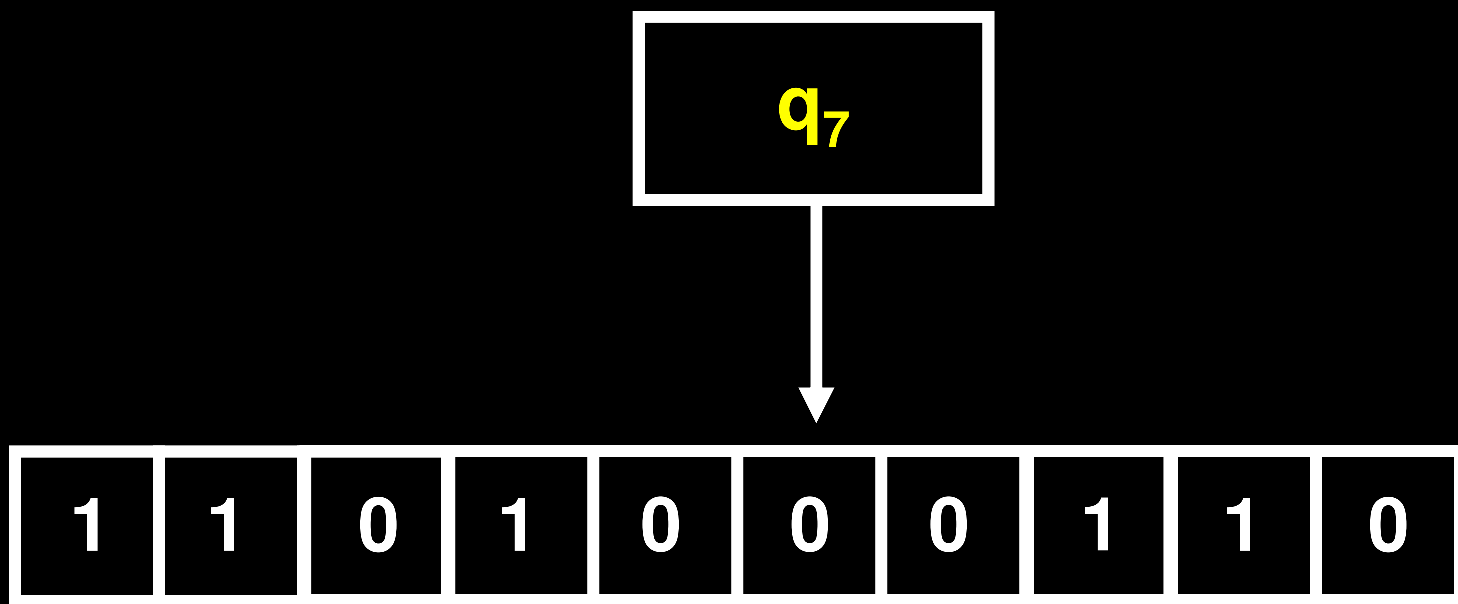
$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

$q_0 \in Q$ 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



corresponds to the *configuration*:

$$11010\,q_7\,00110 \in \{Q \cup \Gamma\}^*$$

Defining Acceptance and Rejection for TMs

Let C_1 and C_2 be configurations of M

Definition. C_1 *yields* C_2 if M is in configuration C_2 after running M in configuration C_1 for one step

Suppose $\delta(q_1, b) = (q_2, c, L)$

Then aaq_1bb yields aq_2acb

Suppose $\delta(q_1, a) = (q_2, c, R)$

Then $cabq_1a$ yields $cabcq_2\square$

Let $w \in \Sigma^*$ and M be a Turing machine

M *accepts* w if there are configs $C_0, C_1, \ldots, C_k$, s.t.

- $C_0 = q_0w$
- C_i yields C_{i+1} for $i = 0, \ldots, k-1$, and
- C_k contains the accept state q_{accept}

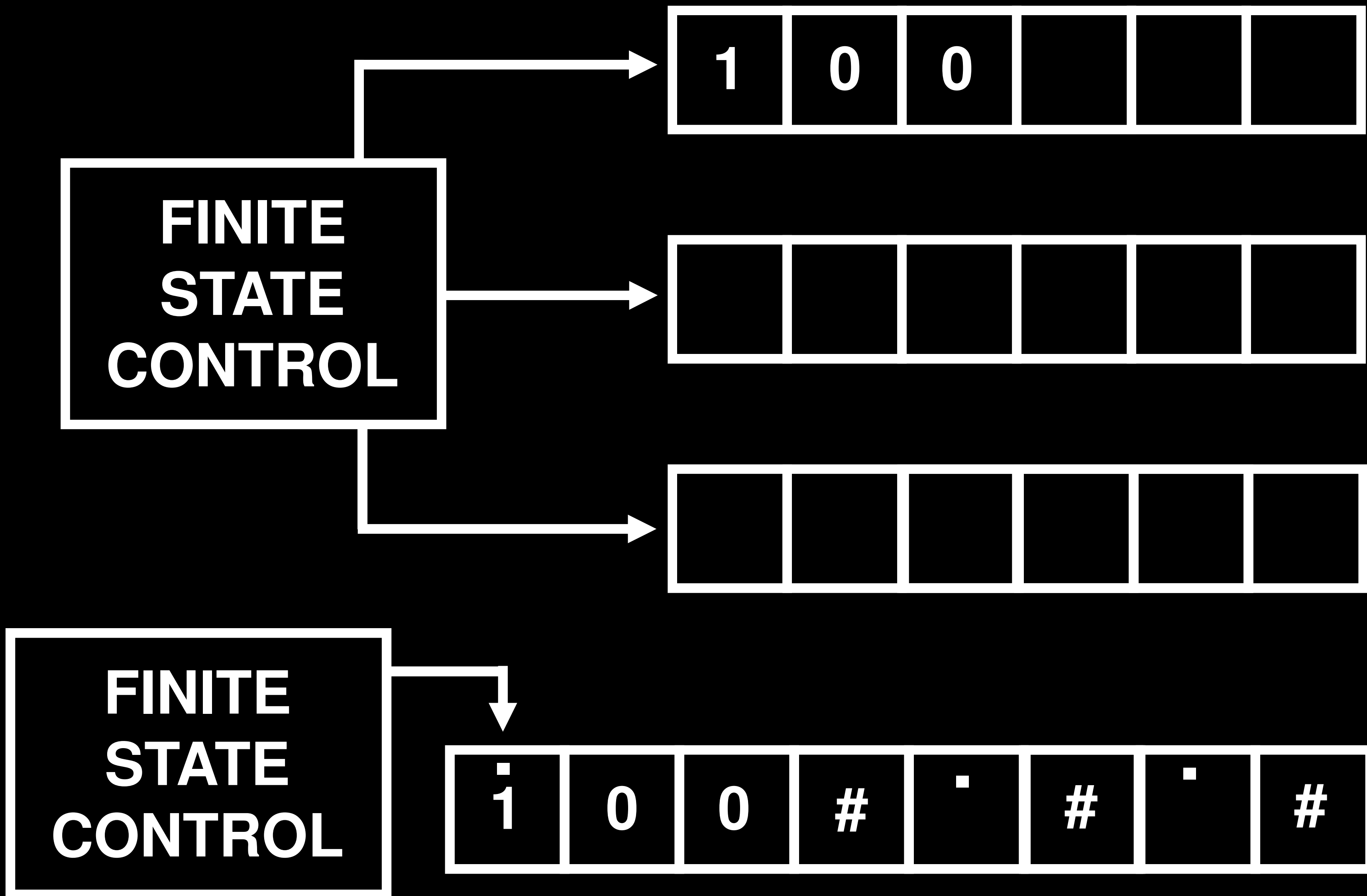
A TM *M recognizes* a language L
if *M* accepts exactly those strings in L

A language L is called recognizable or recursively enumerable (r.e.) if some TM recognizes L

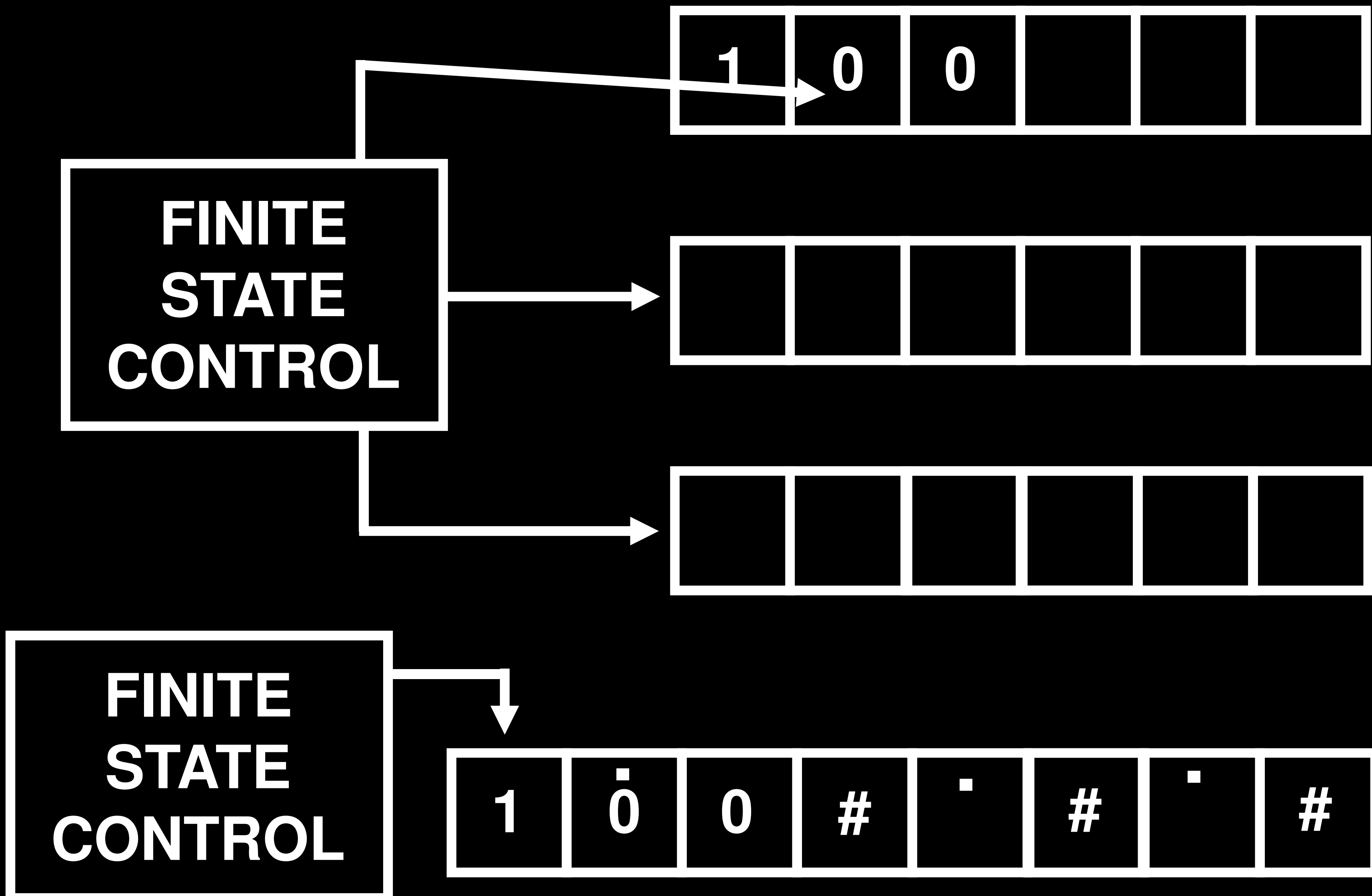
A TM *M decides* a language L if *M* accepts all strings in L and rejects all strings not in L

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

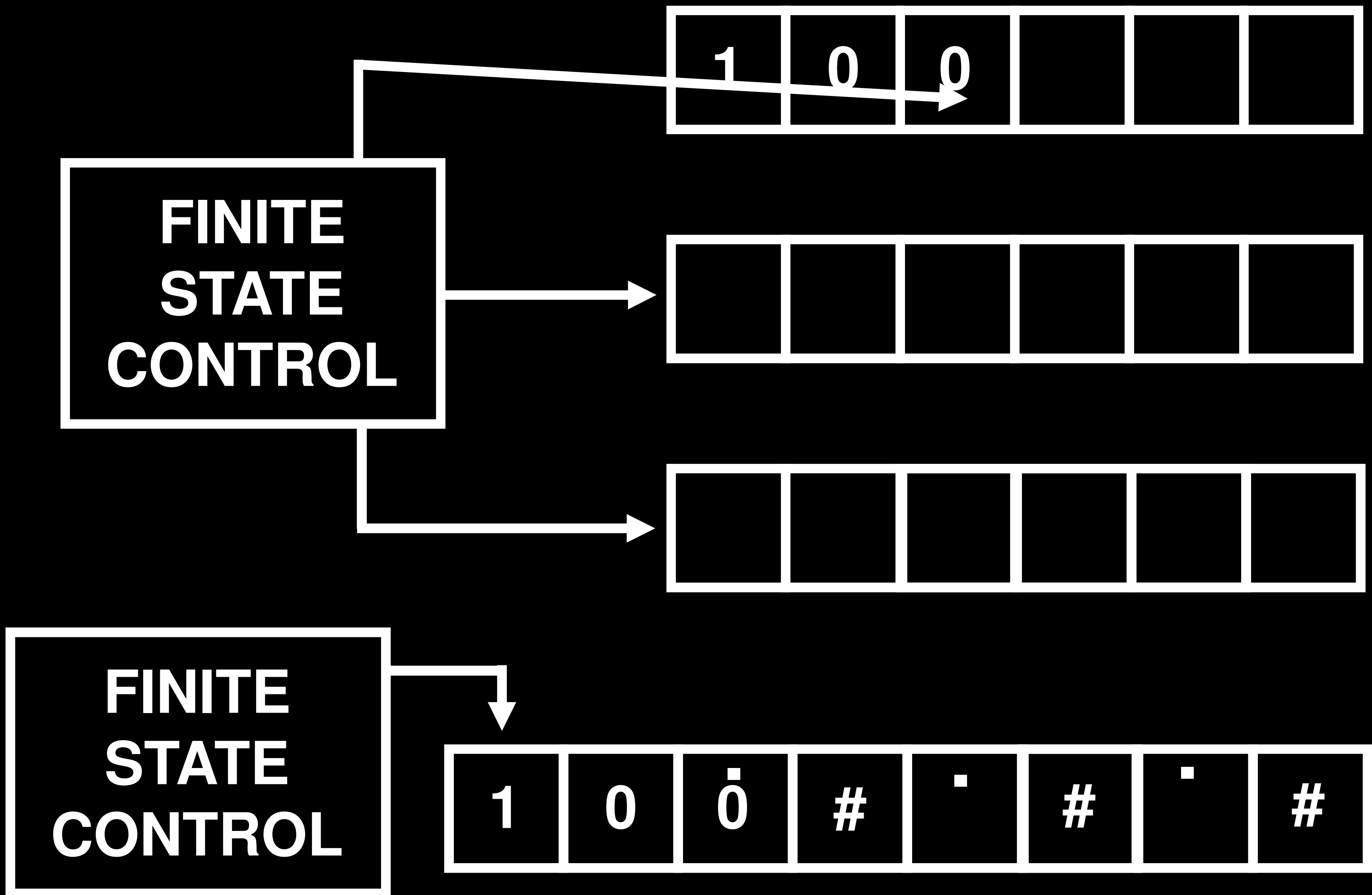
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



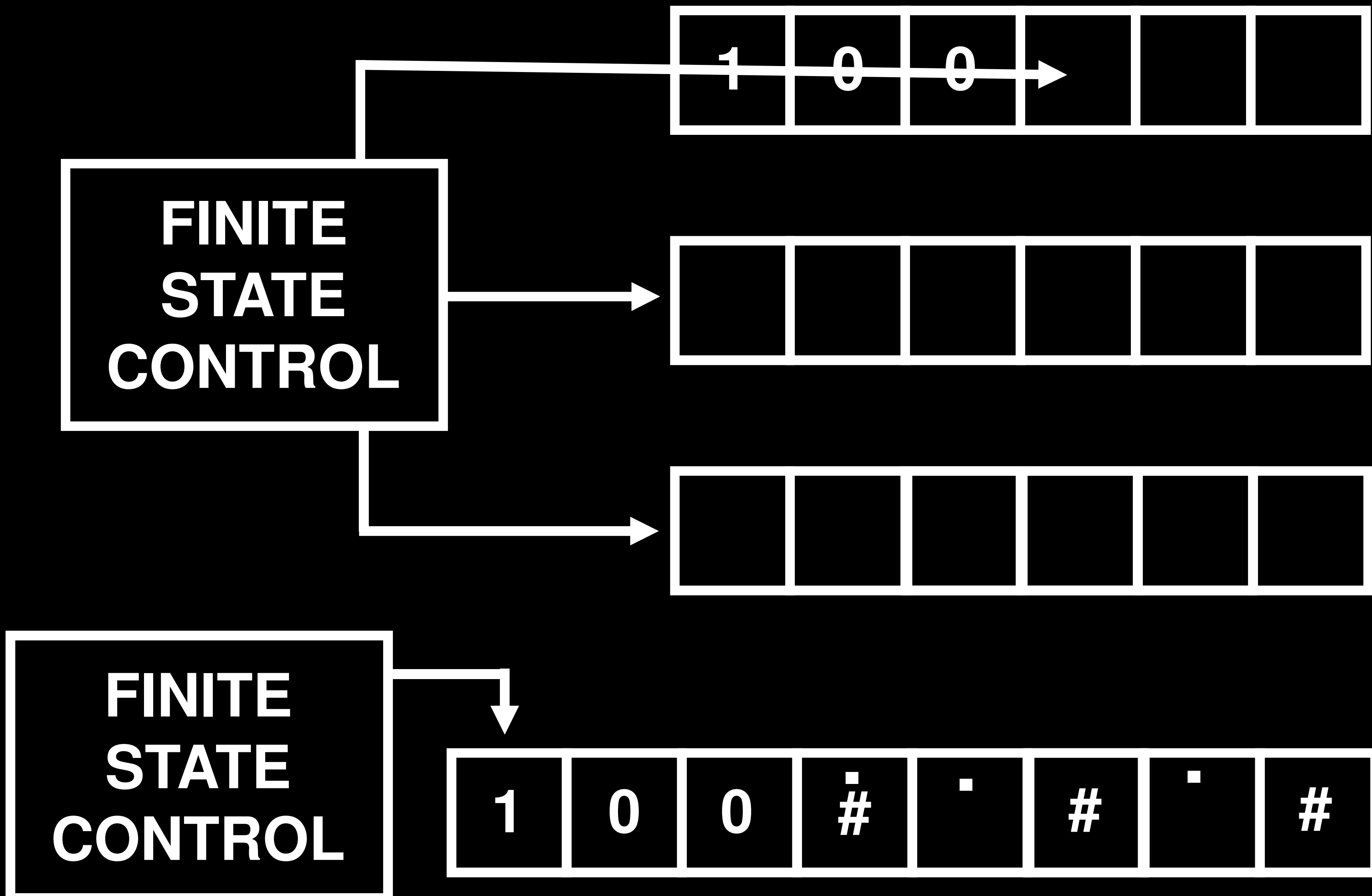
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



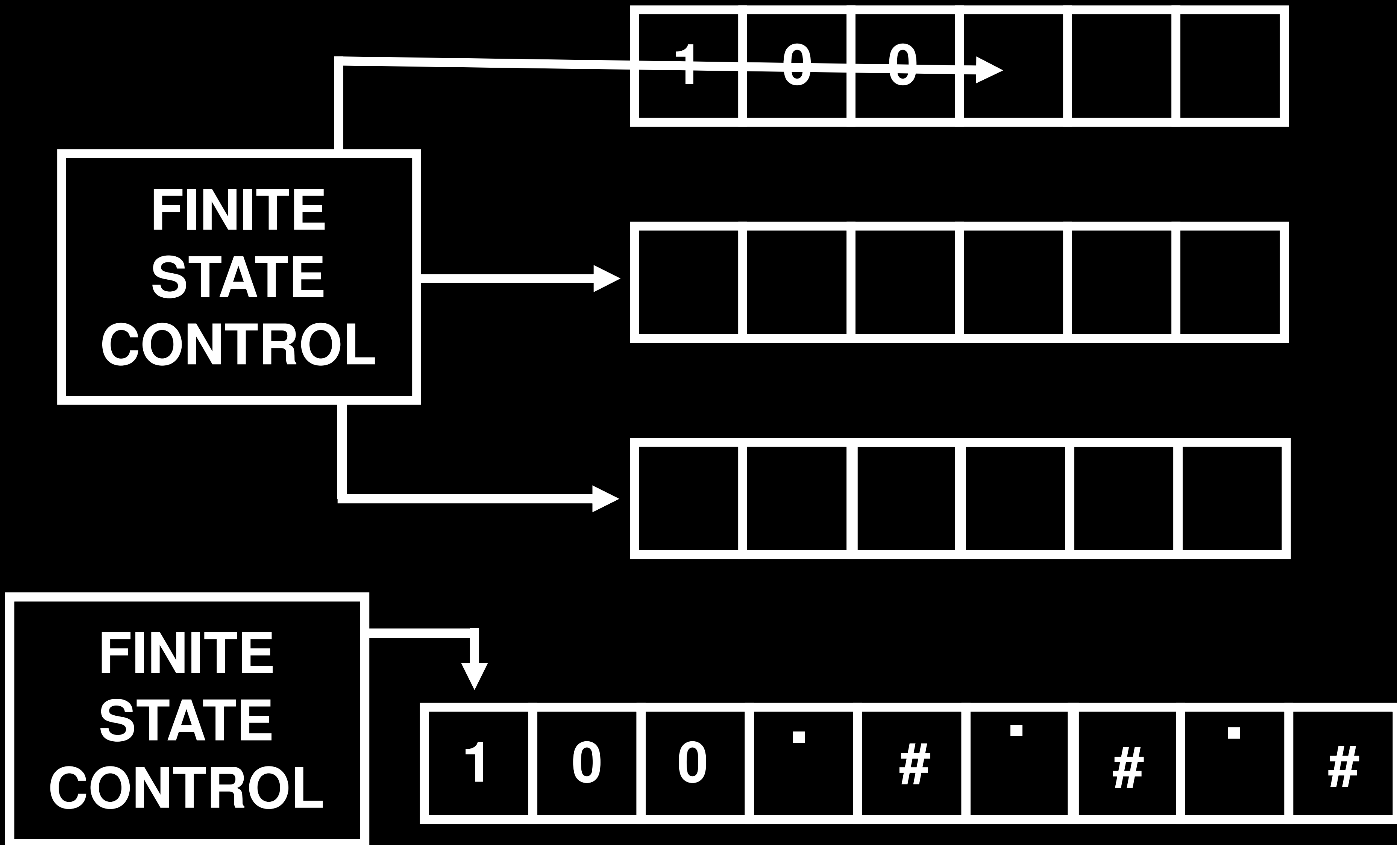
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine

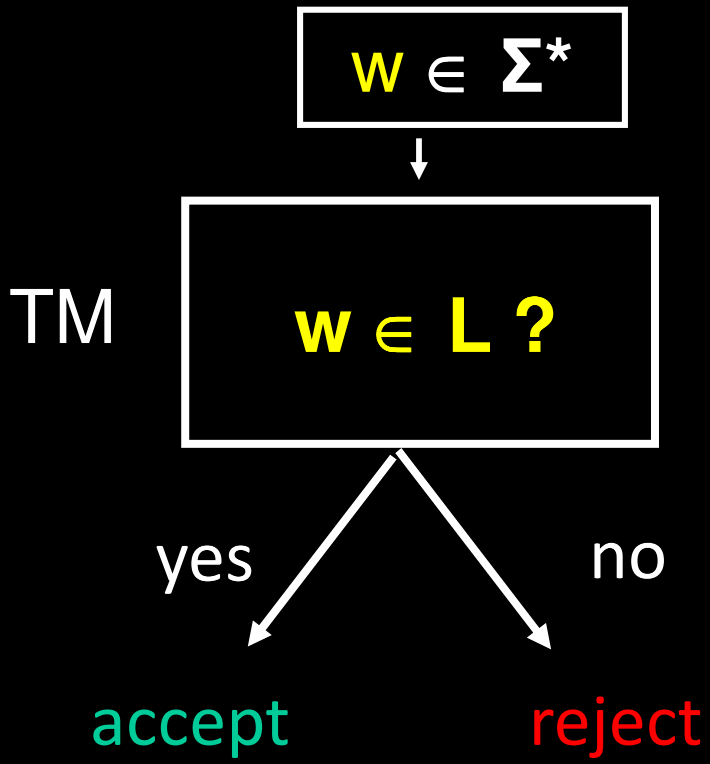


Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine

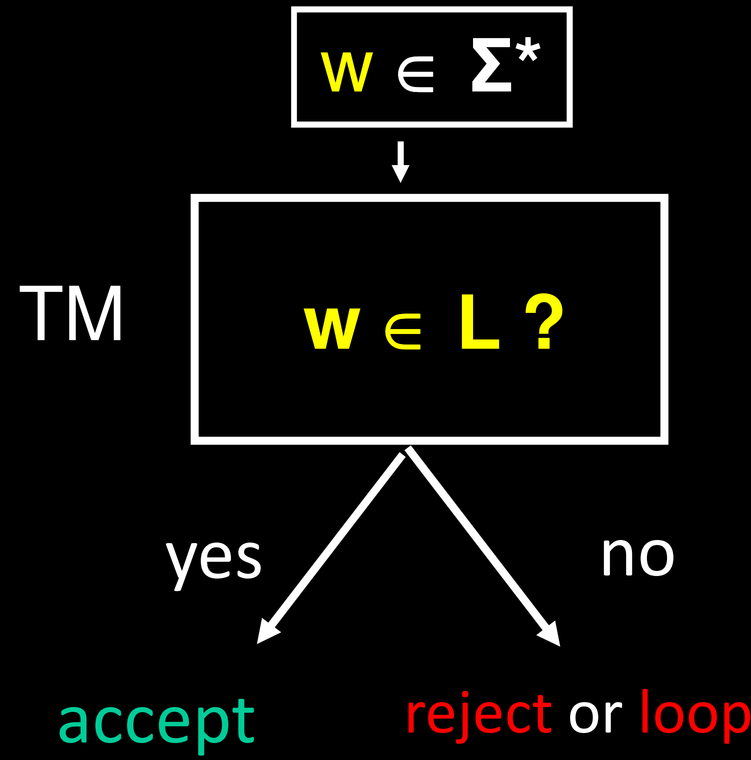


Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine





L is decidable
(recursive)



L is recognizable
(recursively enumerable)

Theorem: L is decidable
iff both L and $\neg$L are recognizable

Recall: Given $L \subseteq \Sigma^*$, define $\neg L := \Sigma^* \setminus L$

Theorem: L is decidable iff both L and $\neg$L are recognizable

Given: a TM M_1 that recognizes L and a TM M_2 that recognizes $\neg$L, we want to build a new machine M that *decides* L

How? Any ideas?

M_1 always accepts x, when x is in L

M_2 always accepts x, when x isn't in L

Recall: Given $L \subseteq \Sigma^*$, define $\neg L := \Sigma^* \setminus L$

Theorem: L is decidable iff both L and $\neg$L are recognizable

Given: a TM M_1 that recognizes L and
a TM M_2 that recognizes $\neg$L,
we want to build a new machine M that *decides* L

M(x): Run M_1 (x) and M_2 (x) on separate tapes.
Alternate between simulating one step of M_1, and one step of M_2.
If M_1 ever accepts, then accept
If M_2 ever accepts, then reject

Nondeterministic Turing Machines

Have multiple transitions for a state, symbol pair

Theorem: Every nondeterministic Turing machine **N** can be transformed into a Turing Machine **M** that accepts precisely the same strings as **N**.

Proof Idea (more details in Sipser)

Pick a natural ordering on all strings in $\{Q \cup \Gamma \cup \#\}^*$

M(w): For all strings $D \in \{Q \cup \Gamma \cup \#\}^*$ in the ordering,

Check if $D = C_0\# \cdots \#C_k$ where $C_0, \ldots, C_k$ is *some* accepting computation history for **N** on **w**.

If so, *accept*.

Fact: We can encode Turing Machines as *bit strings*

n states

start state

reject state

$$0^n10^m10^k10^s10^t10^r10^u1 \ldots$$

m tape symbols (first k are input symbols)

accept state

blank symbol

$$(\,(p, i), (q, j, L)\,) = 0^p10^i10^q10^j10$$

$$(\,(p, i), (q, j, R)\,) = 0^p10^i10^q10^j100$$

Similarly, we can encode DFAs and NFAs as *bit strings*, and $w \in \Sigma^*$ as *bit strings*

For $x \in \Sigma^*$ define $b_\Sigma(x)$ to be its binary encoding

For $x, y \in \Sigma^*$, define the *pair of x and y* to be

$$(x, y) := 0^{|b_\Sigma(x)|} 1\, b_\Sigma(x)\, b_\Sigma(y)$$

Then we define the following languages over {0,1}:

A_{DFA} = { (B, w) | B encodes a DFA over some Σ, and B accepts $w \in \Sigma^*$ }

A_{NFA} = { (B, w) | B encodes an NFA, B accepts w }

A_{TM} = { (M, w) | M encodes a TM, M accepts w }

A_{TM} = { (M, w) | M encodes a TM over some Σ,
w encodes a string over Σ
and M accepts w}

Technical Note:

We'll use an decoding of pairs, TMs, and strings so that *every* binary string decodes to *some* pair (M, w)

If $z \in \{0,1\}^*$ doesn't decode to (M, w) in the usual way, then we *define* that z decodes to the pair (D, ε) where D is a "dummy" TM that accepts nothing.

$\neg A_{TM}$ = { z | z decodes to (M, w) and
M does not accept w }

Universal Turing Machines

Theorem: There is a Turing machine U
which takes as input:
- the code of an arbitrary TM M
- and an input string w

such that U accepts (M, w) ⇔ M accepts w.

This is a *fundamental* property of TMs:
There is a Turing Machine that
can run arbitrary Turing Machine code!

Note that DFAs/NFAs do *not* have this property.
That is, A_{DFA} and A_{NFA} are not regular.

A_{DFA} = { (D, w) | D is a DFA that accepts string w }

Theorem: A_{DFA} is decidable

Proof: A DFA is a special case of a TM.
Run the universal U on (D, w) and output its answer.

A_{NFA} = { (N, w) | N is an NFA that accepts string w }

Theorem: A_{NFA} is decidable. (Why?)

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

Theorem: A_{TM} is recognizable

The Church-Turing Thesis

Everyone's Intuitive Notion of Algorithms = Turing Machines

This is not a theorem – it is a falsifiable scientific hypothesis.

And it has been thoroughly tested!

CURIS about Theory?

Apply to work with me
(or with other theory folks)
this summer, at
http://curis.Stanford.edu

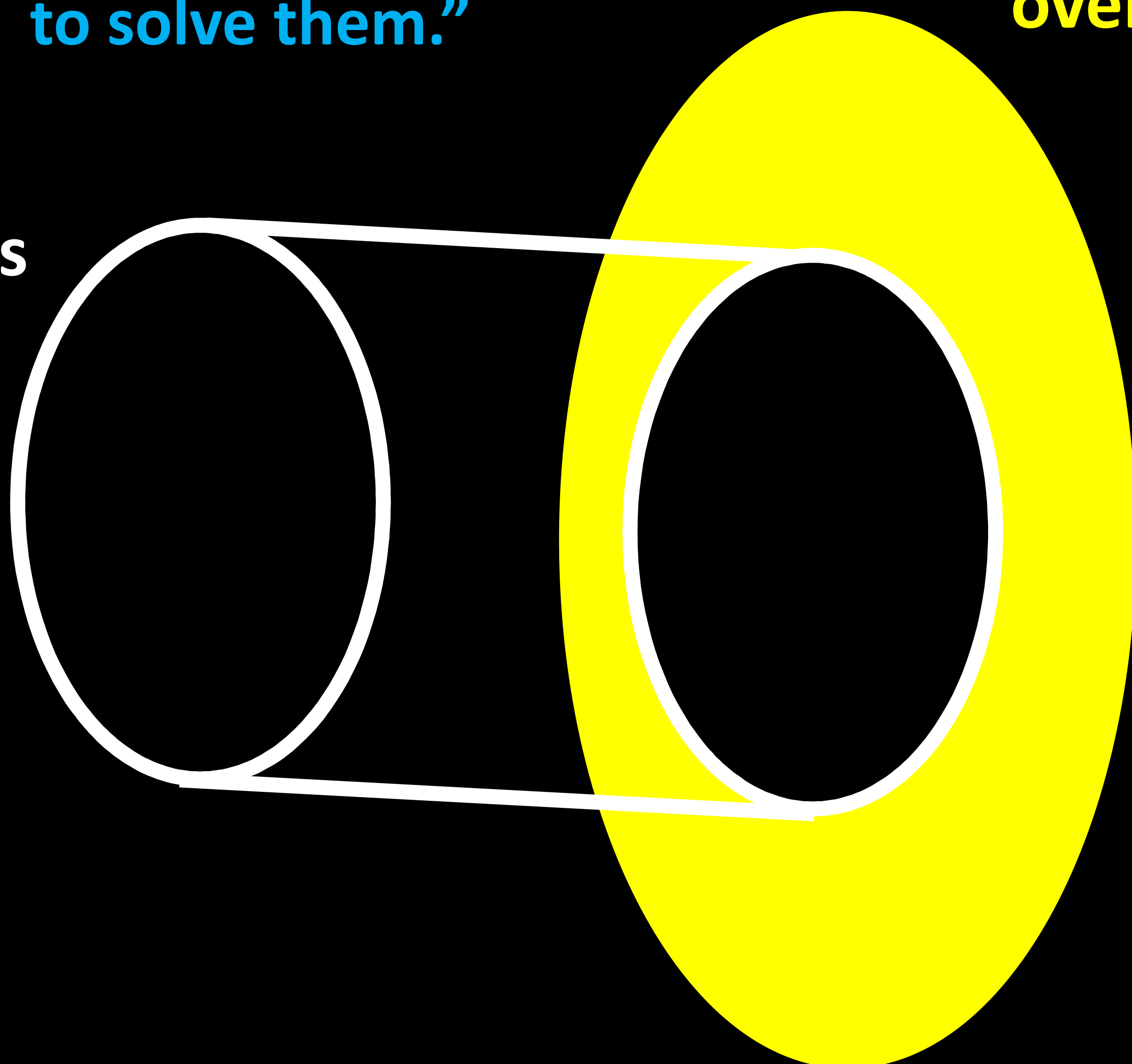
Thm: There are *unrecognizable* languages

Assuming the Church-Turing Thesis,
this means there are problems that
***NO* computing device can solve!**

We will prove that there is no onto function from the set of all Turing Machines to the set of all languages over {0,1}.
(But the proof will work for any *finite* Σ)

That is, every mapping from Turing machines to languages *fails to cover* all possible languages

"There are more problems to solve than there are programs to solve them."



$f : A \rightarrow B$ **is onto** $\Leftrightarrow$ $(\forall\, b \in B)(\exists\, a \in A)[f(a) = b]$

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

Theorem: There is *no* onto function from L to 2^L

Proof: Assume, for a contradiction,

there is an onto function $f : L \rightarrow 2^L$

Define $S = \{\, x \in L \mid x \notin f(x) \,\} \in 2$

If f is onto, then there is a $y \in L$ with $f(y) = S$

Suppose $y \in S$. By definition of S, $y \notin f(y) = S$.

Suppose $y \notin S$. By definition of S, $y \in f(y) = S$.

Contradiction!

$f : A \rightarrow B$ is *not* onto $\Leftrightarrow$ $(\exists b \in B)(\forall a \in A)[f(a) \neq b]$

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

Theorem: There is *no* onto function from L to 2^L

Proof: Let $f : L \rightarrow 2^L$ be an arbitrary function

Define $S = \{ x \in L \mid x \notin f(x) \} \in 2^L$

For all $x \in L$,

If $x \in S$ then $x \notin f(x)$ [by definition of S]

If $x \notin S$ then $x \in f(x)$

In either case, we have $f(x) \neq S$. (Why?)

Therefore f is not onto!

What does this mean?

No function from L to 2^L
can "cover" all the elements in 2^L

No matter what the set L is,
the power set 2^L *always* has
strictly larger cardinality than L

Thm: There are *unrecognizable* languages

Proof: If all languages were recognizable, then for all L, there'd be a Turing machine M for recognizing L.

Hence there is an onto R: {Turing Machines} $\rightarrow$ {Languages}

{Turing Machines}

$\subseteq$

{0,1}*

=

Set **M**

{Languages over {0,1}}

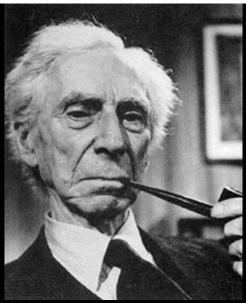
$\updownarrow$

{Sets of strings of 0s and 1s}

=

Set of all subsets of M: 2^M

Therefore, there is *no* onto function from {Turing Machines} $\subseteq$ M to {Languages}. **Contradiction!**



Russell's Paradox in Set Theory

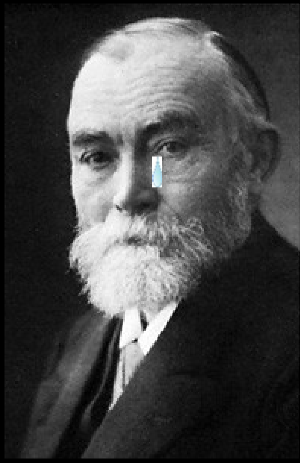
In the early 1900's, logicians were trying to define consistent foundations for mathematics.

Suppose X = "Universe of all possible sets"

Frege's Axiom: Let $f : X \rightarrow \{0,1\}$

Then $\{S \in X \mid f(S) = 1\}$ is a set.

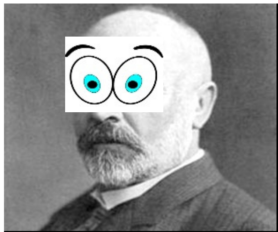
Define $F = \{ S \in X \mid S \notin S \}$



Suppose $F \in F$. Then by definition, $F \notin F$.

So $F \notin F$ and by definition $F \in F$.

This logical system is inconsistent!



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

{0,1}* Power set of {0,1}*

Proof: Suppose **f** is such a function:

$$
\begin{array}{lcl}
1 & \longrightarrow & 0.\mathbf{2}8347279\ldots \\
2 & \longrightarrow & 0.8\mathbf{8}388384\ldots \\
3 & \longrightarrow & 0.77\mathbf{6}35284\ldots \\
4 & \longrightarrow & 0.111\mathbf{1}1111\ldots \\
5 & \longrightarrow & 0.1234\mathbf{5}678\ldots \\
\vdots & & \quad\vdots
\end{array}
$$

Define: $r \in (0, 1)$

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

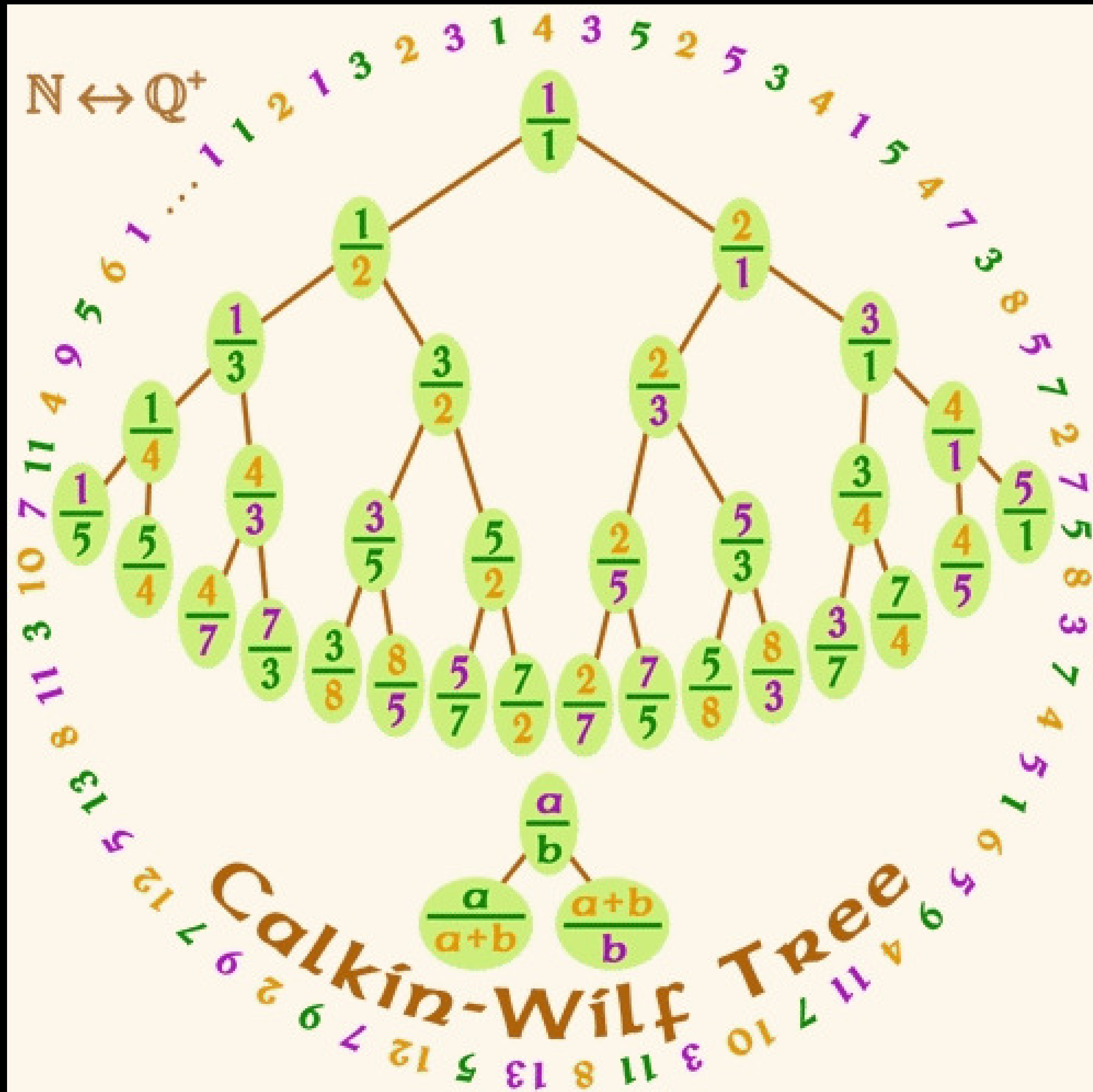
$f(n) \neq r$ for all n (Here, $r = 0.11121\ldots$)

r is never output by f

Let $Z^+ = \{1, 2, 3, 4, \ldots\}$

There *is* a bijection between Z^+ and $Z^+ \times Z^+$

(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)	…



$\mathbb{N} \leftrightarrow \mathbb{Q}^+$
Calkin-Wilf Tree
$\frac{a}{b}$
$\frac{a}{a+b}$
$\frac{a+b}{b}$

A Concrete Undecidable Problem: The Acceptance Problem for TMs

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

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

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

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

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

Suppose H is a machine that decides A_{TM}

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

Define a new TM D as follows:

D(M): Run H on (M,M) and output the opposite of H

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

The table of outputs of H(x,y)

	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		?

The outputs of D(x)

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

D(x) outputs the opposite of H(x,x)

D(D) outputs the opposite of H(D,D)=D(D)