CS 154

Finish up K-Complexity, Time Complexity, P and NP

CS 154

Midterms back today!

Thank you for your feedback!

There Exist 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

but (Number of descriptions of length < n)

$\leq$ (Number of binary strings of length < n)

$= 1 + 2 + 4 + \cdots + 2^{n-1} = 2^n - 1$

Therefore, there is at least one n-bit string x that does *not* have a description of length < n

Random Strings Are Incompressible!

Theorem: For all n and c ≥ 1,

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

"Most strings are highly incompressible"

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

but (Number of descriptions of length < n-c)

≤ (Number of binary strings of length < n-c)

= $2^{n-c} - 1$

Hence the probability that a *random* string x satisfies

$$K(x) < n-c$$

is at most $(2^{n-c} - 1)/2^n < 1/2^c$.

Kolmogorov Complexity: Try it!

Give short algorithms for generating the strings:

1. **0100011011000001010011100101110111000000001**
2. **1235813213455891442333776109871597**
3. **126241207205040403203628803628800399168800**

Kolmogorov Complexity: Try it!

Give short algorithms for generating the strings:

1. **0100011011000001010011100101110111000000001**
2. **1235813213455891442333776109871597**
3. **126241207205040403203628803628800399168800**

Kolmogorov Complexity: Try it!

Give short algorithms for generating the strings:

1. **0100011011000001010011100101111011100000001**
2. **1235813213455891442333776109871597**
3. **126241207205040403203628803628800399168800**

Kolmogorov Complexity: Try it!

Give short algorithms for generating the strings:

1. **01000110110000010100111001011011100000001**
2. **1235813213455891442333776109871597**
3. **126241207205040403203628803628800399168800**

This seems hard to determine in general. Why?

Determining Compressibility?

Can an algorithm perform optimal compression?
Can algorithms tell us if a given string is compressible?

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

Theorem: COMPRESS is undecidable!

Idea: If decidable, we could design an algorithm that prints the **shortest incompressible string of length n**

But such a string could then be succinctly described, by providing the algorithm code and n in binary!

Berry Paradox: "The smallest integer that cannot be defined in less than thirteen words."

Determining Compressibility?

COMPRESS = {(x,c) | K(x) ≤ c}

Theorem: COMPRESS is undecidable!

Proof: Suppose it's decidable. Consider the TM:

M = "On input x ϵ {0,1}*, let N = $2^{|x|}$.
For all y ϵ {0,1}* in lexicographical order,
If (y,N) ∉ COMPRESS then print y and halt."

M(x) prints the shortest string y' with K(y') > $2^{|x|}$.

<M,x> is a description of y', and |<M,x>| ≤ d + |x|

So $2^{|x|}$ < K(y') ≤ d + |x|. CONTRADICTION for large x!

Computational Complexity Theory

Computational Complexity Theory

**What can and can't be computed with limited resources on computation,
such as time, space, and so on**

Captures many of the significant issues in practical problem solving

The field is rich with important open questions that no one has any idea how to begin answering!

We'll start with: Time complexity

Very Quick Review of Big-O

Let f and g be functions f, g : $\mathbb{N} \to \mathbb{N}$.
We say that f(n) = O(g(n)) if there are positive integers c and n_0 so that for every integer $n \geq n_0$

$$f(n) \leq c\, g(n)$$

We say g(n) is an upper bound on f(n) if
f(n) = O(g(n))

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

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

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

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

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

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

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

Big-O can help isolate the "dominant" term of a function

Measuring Time Complexity of a TM

We measure time complexity by counting the steps taken for a Turing machine to halt

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

Here's a TM for A. On input of length n:

O(n) 1. Scan across the tape and reject if the string is not of the form 0^i1^j

$O(n^2)$ 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

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

Let M be a TM that halts on all inputs.
(We will only consider decidable languages now!)

Definition:
The running time or time complexity of M is the function T : $\mathbb{N} \to \mathbb{N}$ such that

T(n) = maximum number of steps taken by M over all inputs of length n

Time-Bounded Complexity Classes

Definition:

TIME(t(n)) = { L' | there is a Turing machine M with time complexity O(t(n)) so that L' = L(M) }

= { L' | L' is a language decided by a Turing machine with O(t(n)) running time }

We just showed: $A = \{ 0^k 1^k \mid k \geq 0 \} \in \text{TIME}(n^2)$

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

M(w) := If w is not of the form 0*1*, reject.

Repeat until all bits of w are crossed out:

If (parity of 0's) ≠ (parity of 1's), reject.

Cross out every other 0. Cross out every other 1.

Once all bits are crossed out, accept.

```
000000000000111111111111
x0x0x0x0x0x0xx1x1x1x1x1x1x
xxx0xxx0xxx0xxxx1xxx1xxx1x
xxxxxxx0xxxxxxxxxxxx1xxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxx
```

It can be proved that a (one-tape) Turing Machine *cannot* decide A in *less* than O(n log n) time!

Extra Credit Problem:

Let f(n) = $\mathbf{O}\left(\frac{n \log n}{\alpha(n)}\right)$ where $\alpha(n)$ is unbounded.

Prove: TIME(f(n)) contains only regular languages(!)

For example, TIME(n log log n) contains only regular languages!

Two Tapes Can Be More Efficient

Theorem: $A = \{ 0^k 1^k \mid k \geq 0 \}$ can be decided in O(n) time with a *two-tape* TM.

Proof Idea:

Scan all 0s, copy them to the second tape.

Scan all 1s. For each 1 scanned, cross off a 0 from the second tape.

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

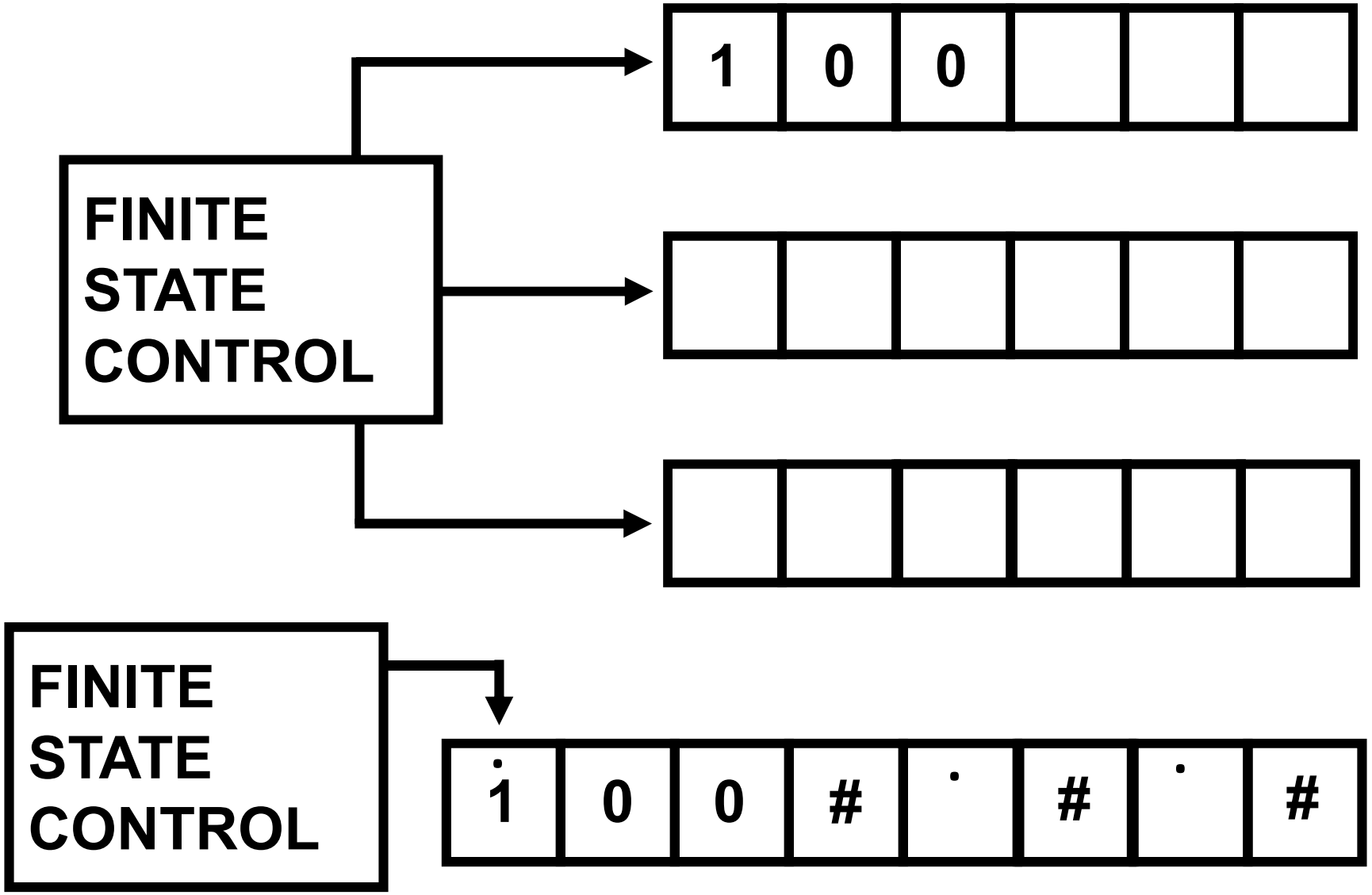
Let's revisit some of the key concepts from computability theory…

Theorem: Let $t : \mathbb{N} \to \mathbb{N}$ satisfy $t(n) \geq n$, for all n.
Then every t(n) time multi-tape TM has an equivalent $O(t(n)^2)$ time one-tape TM

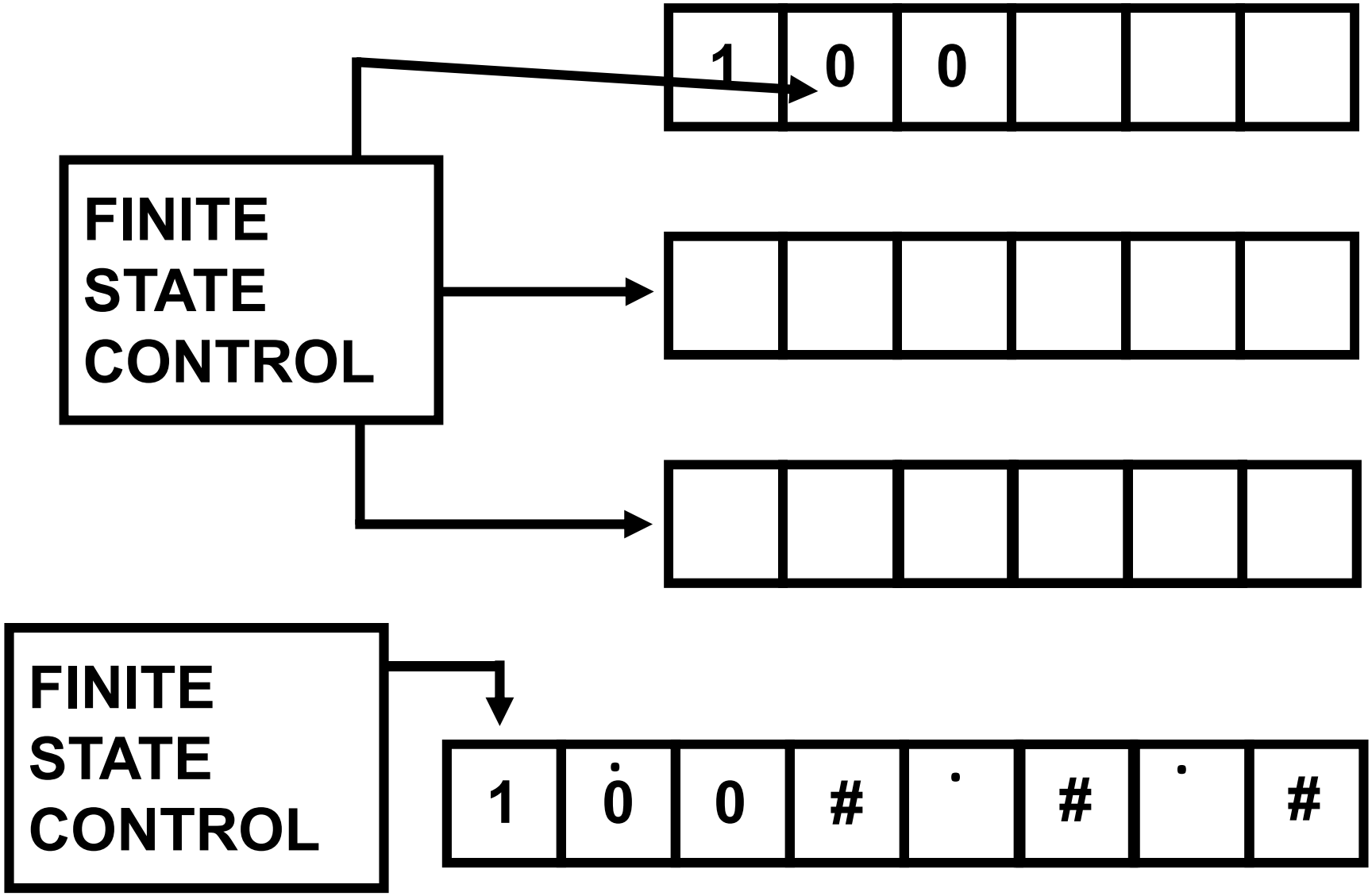
Our simulation of multitape TMs by one-tape TMs achieves this!

Corollary: Suppose language A can be decided by a multi-tape TM in p(n) steps, for some polynomial p. Then A can be decided by a one-tape TM in q(n) steps, for some polynomial q(n).

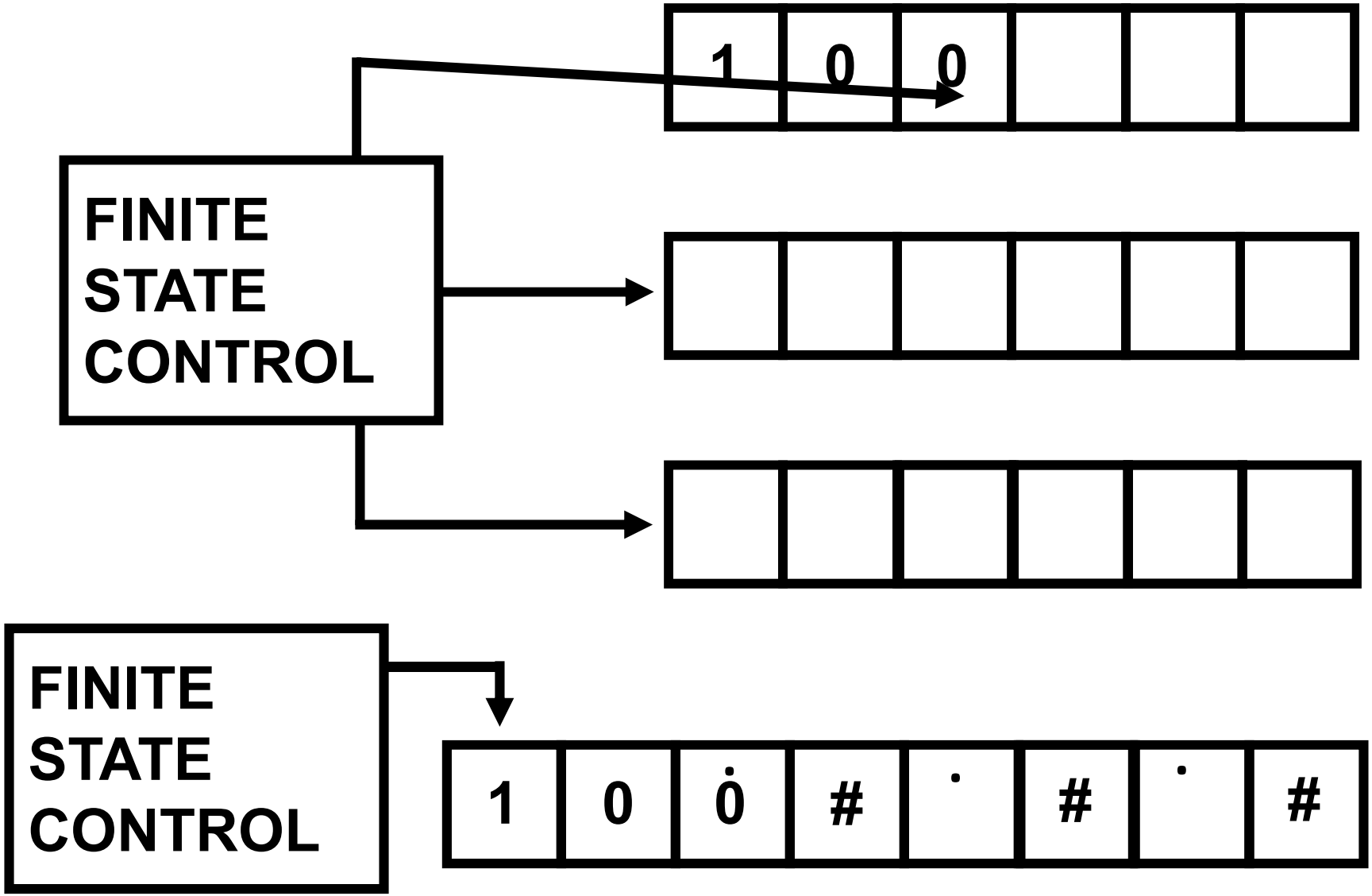
Theorem: For every t(n) time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM



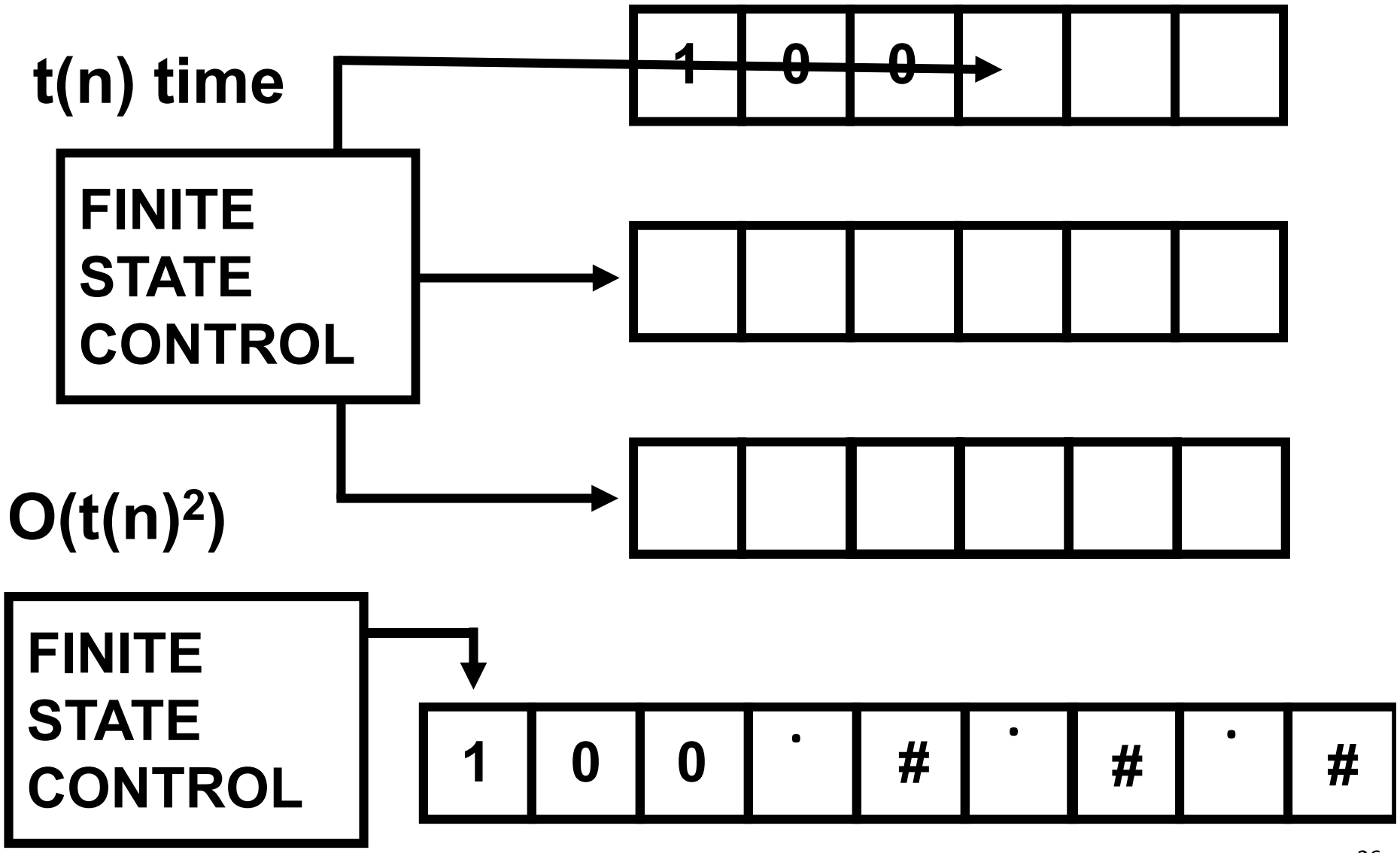
Theorem: For every t(n) time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM



Theorem: For every t(n) time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM



Theorem: For every t(n) time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM



Time Complexity of the Universal TM

Theorem: There is a (one-tape) Turing machine U which takes as input:

- **the code of an arbitrary TM M**
- **an input string w**
- **and a string of t 1s, t > |w|**

such that U(M, w, 1^t) halts in $O(|M|^2 t^2)$ steps

and U accepts (M, w, 1^t) ⇔ M accepts w in t steps

The Universal TM with a Clock

Idea: Make a multi-tape TM U' that does the above, and runs in O(|M| t) steps

The Time Hierarchy Theorem

Intuition: If you get more time to compute, then you can solve strictly more problems.

Theorem: For all "reasonable" f, g : $\mathbb{N} \to \mathbb{N}$ where for all n, $g(n) > n^2 f(n)^2$, $\mathbf{TIME(f(n)) \subsetneq TIME(g(n))}$

Proof Idea: Diagonalization with a clock.
Make a TM N that on input M,
simulates the TM M on input M for f(|M|) steps,
then flips the answer.

Then, L(N) cannot have time complexity f(n)

The Time Hierarchy Theorem

Theorem: For "reasonable" f, g where $g(n) > n^2 f(n)^2$,

$$\mathbf{TIME(f(n)) \subsetneq TIME(g(n))}$$

Proof Sketch: Define a TM N as follows:

N(M) = Compute t = f(|M|)
Run U(M, M, 1^t) and output the opposite answer.

Claim: L(N) does not have time complexity f(n).
Proof: Assume N' runs in f(n) time, and L(N') = L(N).
By assumption, N'(N') runs in f(|N'|) time and
outputs the *opposite* answer of U(N', N', $1^{f(|N'|)}$)
But by definition of U, U(N', N', $1^{f(|N'|)}$) accepts
$\Leftrightarrow$ N'(N') accepts in f(|N'|) steps.
This is a contradiction!

The Time Hierarchy Theorem

Theorem: For "reasonable" f, g where $g(n) > n^2 f(n)^2$,

$$\mathbf{TIME(f(n)) \subsetneq TIME(g(n))}$$

Proof Sketch: Define a TM N as follows:

N(M) = Compute t = f(|M|)

Run U(M, M, 1^t) and output the opposite answer.

So, L(N) does *not* have time complexity f(n).

What do we need in order for N to run in O(g(n)) time?

1. **Compute f(|M|) in O(g(|M|)) time ["reasonable"]**
2. **Simulate U(M, M, 1^t) in O(g(|M|)) time**

Recall: U(M, w, 1^t) halts in $O(|M|^2 t^2)$ steps

Set g(n) so that $g(|M|) > |M|^2 f(|M|)^2$ for all n. QED

Remark: Time hierarchy also holds for multitape TMs!

A Better Time Hierarchy Theorem

Theorem: For "reasonable" f, g where
$g(n) > f(n) \log^2 f(n)$, $\mathbf{TIME(f(n)) \subsetneq TIME(g(n))}$

Corollary: $TIME(n) \subsetneq TIME(n^2) \subsetneq TIME(n^3) \subsetneq \ldots$

There is an infinite hierarchy of increasingly more time-consuming problems

Question: Are there important everyday problems that are high up in this time hierarchy?

A *natural* problem that *needs exactly* n^{10} time?

THIS IS AN OPEN QUESTION!

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

Polynomial Time