CS 154

Lecture 13: Time Complexity, P and NP

CS 154

Midterms back during my office hours tomorrow!

Thanks for your feedback

Your Feedback

Feedback	Number of students
HW is too hard	10
HW is too easy	0
Pace is too fast	4
Pace/hw is just right	9
Likes Ryan/slides/lecs	51
Hates streaming/comm	4
Likes TAs	10
Want More feedback from TAs on hw	6

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

Quick Review of Big-O

Let f and g be two functions f, g : $N \rightarrow R^+$.
Recall 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** for f(n) if
$f(n) = O(g(n))$

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

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

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

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

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

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

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

Measuring Time Complexity

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

Consider the language $A = \{ 0^k 1^k \mid k \geq 0 \}$

On input of length n:

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

$O(n^2)$ 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' | there is a TM M and $c > 0$ such that the time complexity of M is $\leq c \cdot t(n)$ and 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^k1^k | $k \geq 0$ } $\in$ 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 the parity of 0's $\neq$ parity of 1's, reject.

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

Once all bits are crossed out, accept.

```
00000000000001111111111111
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: Let $f(n) = o(n \log n)$.
Prove: TIME(f(n)) contains only regular languages(!)

Recall: $f(n) = o(g(n)) \Leftrightarrow \lim_{n\to\infty} f(n)/g(n) = 0$

So for example, TIME(n log log n) contains only regular languages.

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.

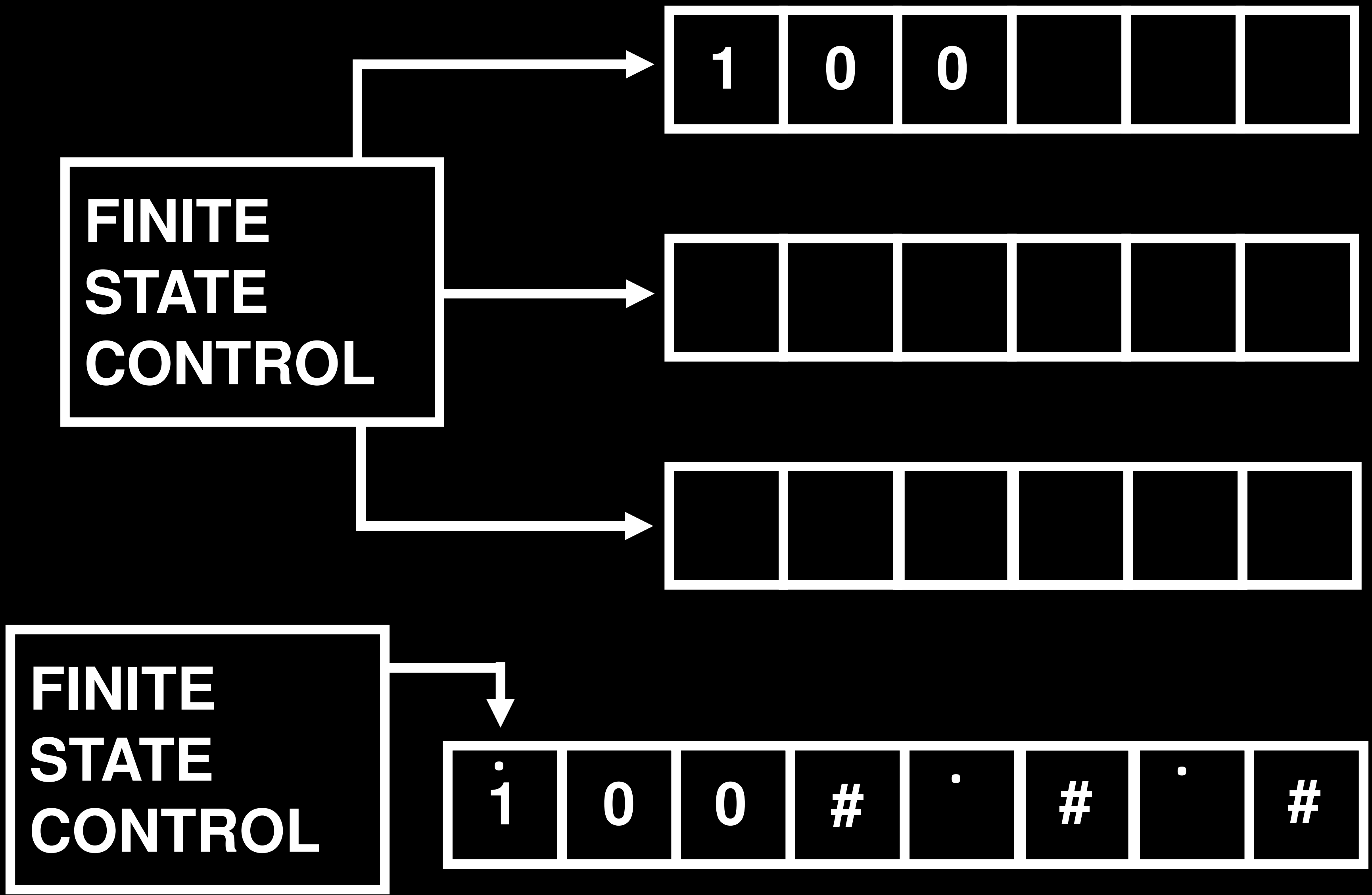
Different models of computation can yield different running times for the same language!

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

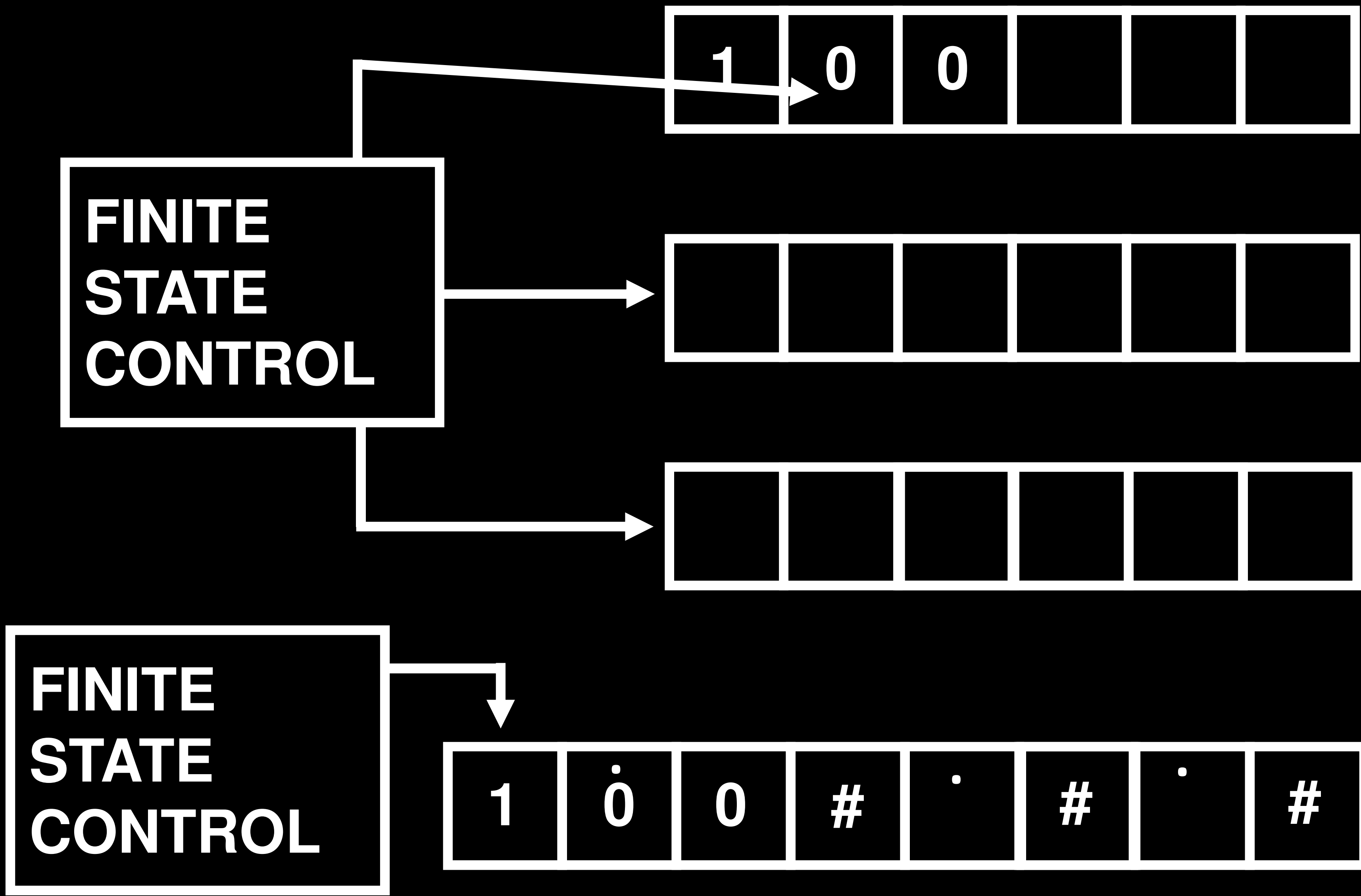
The simulation of multitape TMs with one-tape TMs achieves this!

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).

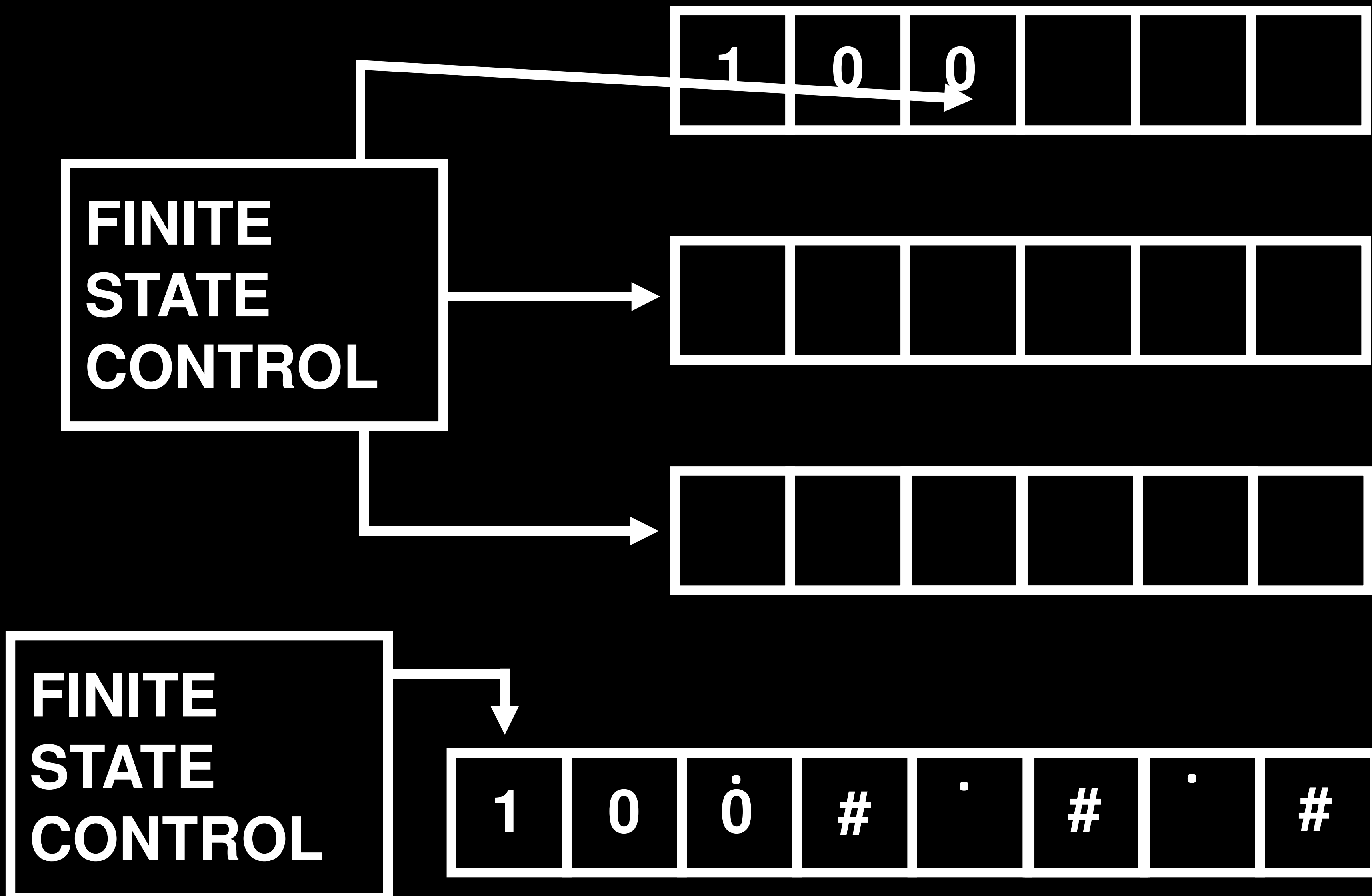
Theorem: For every t(n) time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM



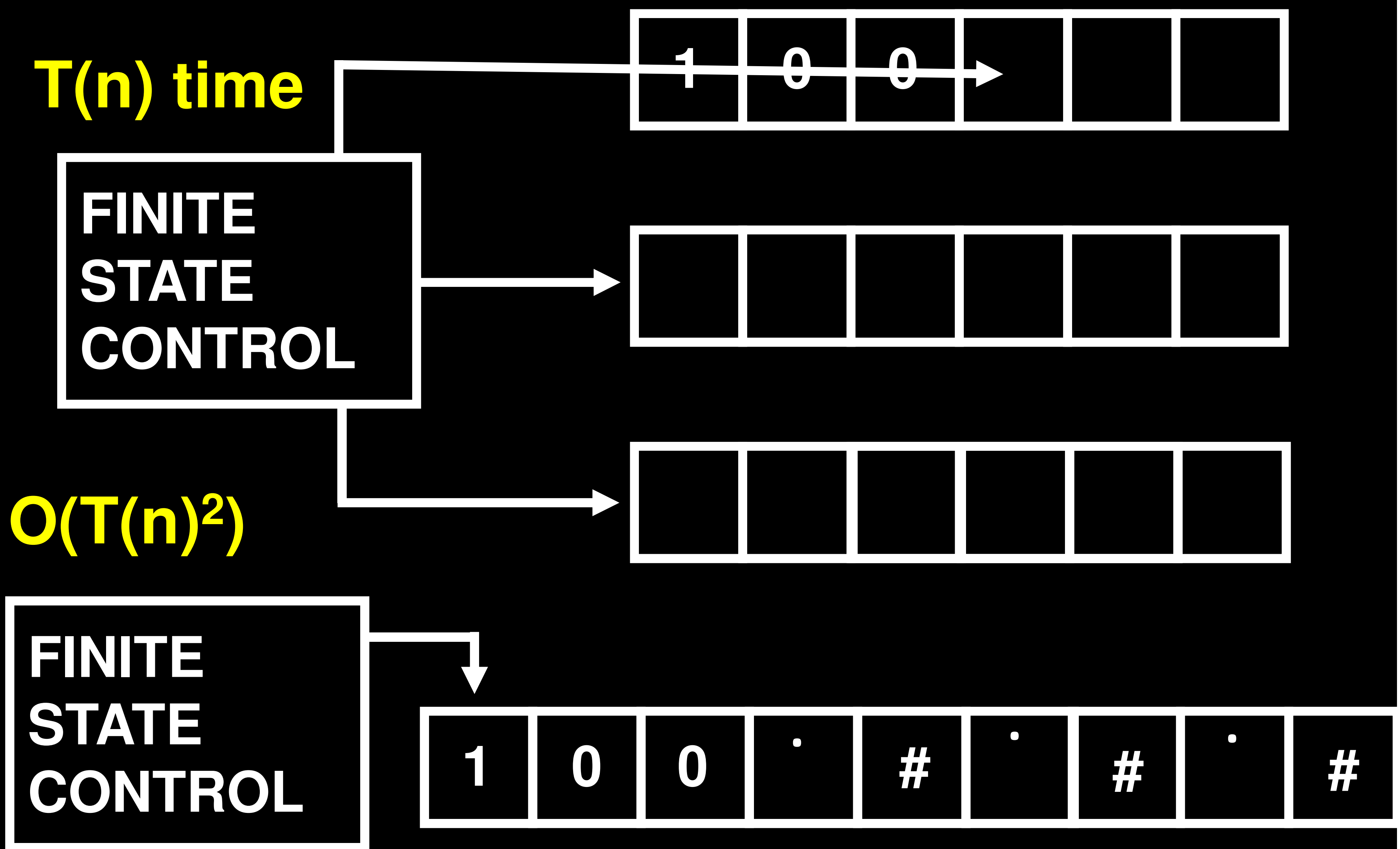
Theorem: For every $t(n)$ time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM



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 an integer $t > |w|$

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

and U accepts (M, w, t) $\Leftrightarrow$ M accepts w in t steps

The Universal TM with a Clock

There is a Turing Machine that can (efficiently) run arbitrary Turing Machine code!

The Time Hierarchy Theorem

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

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

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,
and flips M's answer.

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

The Time Hierarchy Theorem

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

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

Proof Sketch: Define a TM N as follows:

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

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

Claim: L(N) does not have time complexity f(n).

Proof: Suppose N' runs in f(n) time, and L(N') = L(N).

Consider N'(N'). This runs in f(|N'|) time and outputs the *opposite* answer of U(N', N', f(|N'|))

But by definition of U, U(N', N', f(|N'|)) accepts

⇔ N'(N') accepts in f(|N'|) steps.

This is a contradiction!

The Time Hierarchy Theorem

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

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

Proof Sketch: Define a TM N as follows:

N(M) = Compute $t = f(|M|)$.
Run U(M, M, 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, t) in $O(g(|M|))$ time

Recall: U(M, w, 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!

The Time Hierarchy Theorem

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

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

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!

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

Polynomial Time

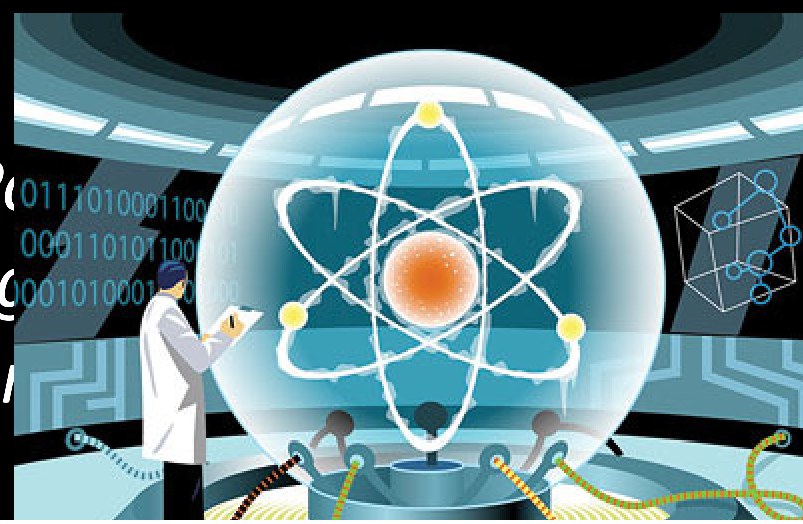
The EXTENDED Church-Turing Thesis

Everyone's Intuitive Notion of Efficient Algorithms = **Polynomial-Time Turing Machines**

A controversial thesis! *P*

include n^{100} time alg

algorithms, quantu



Nondeterminism and NP

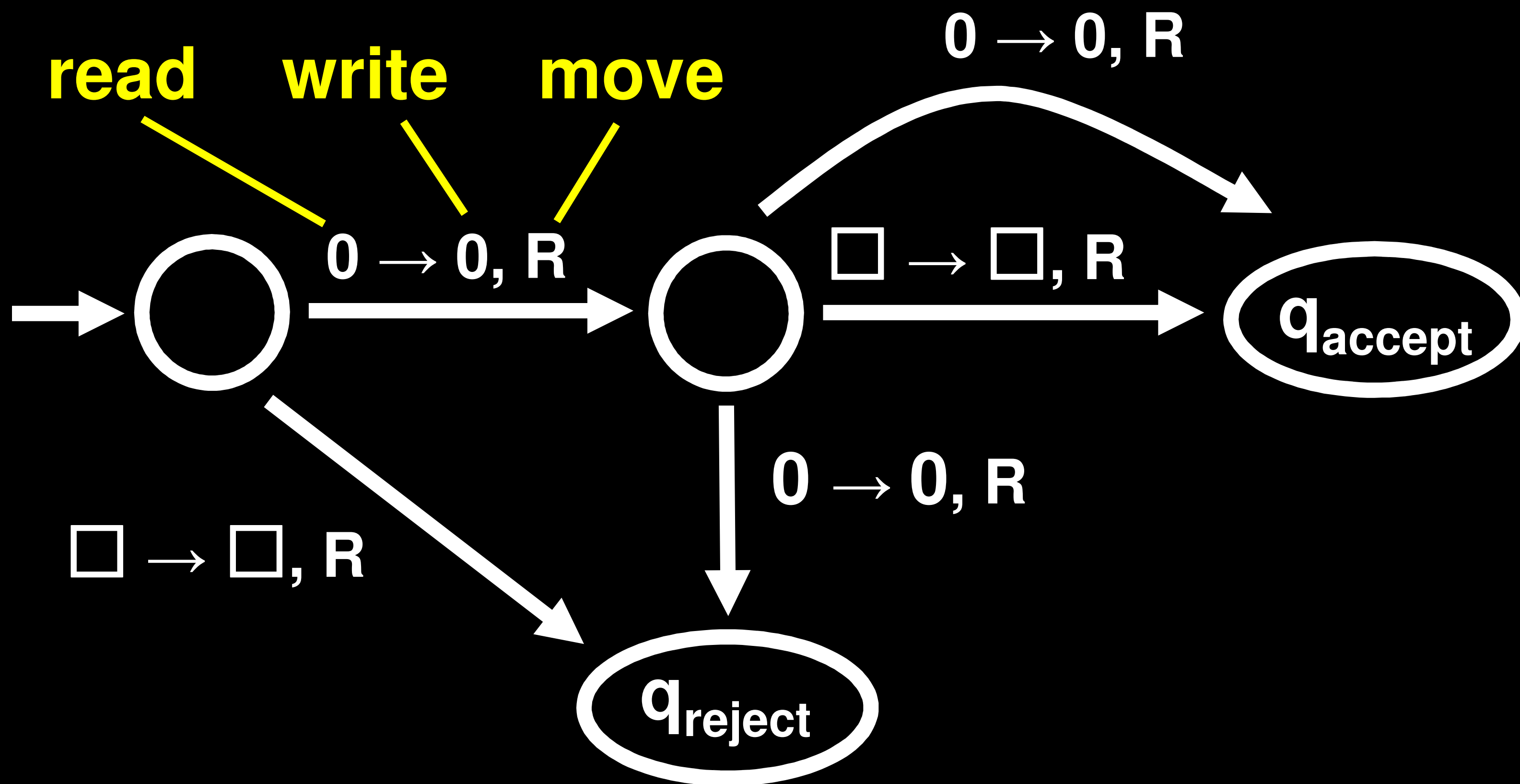
KEEP

CALM

AND

Keep

Guessing


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

Nondeterministic Turing Machines

…are just like standard TMs, except:

1. The machine may proceed according to several possible transitions (like an NFA)

2. The machine *accepts* an input string if there *exists* an accepting computation history for the machine on the string

Definition: A **nondeterministic** 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 $\square \notin \Sigma$

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

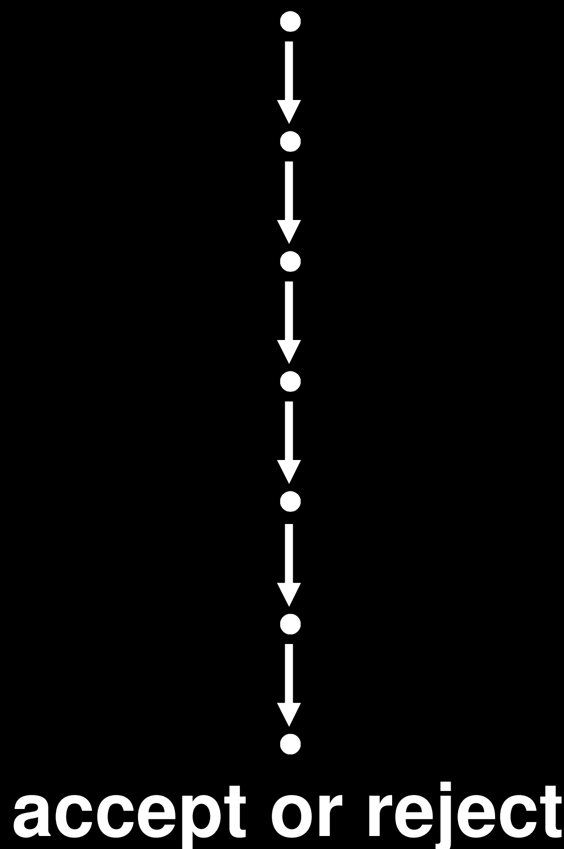
$\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L,R\})}$

$q_0 \in Q$ is the start state

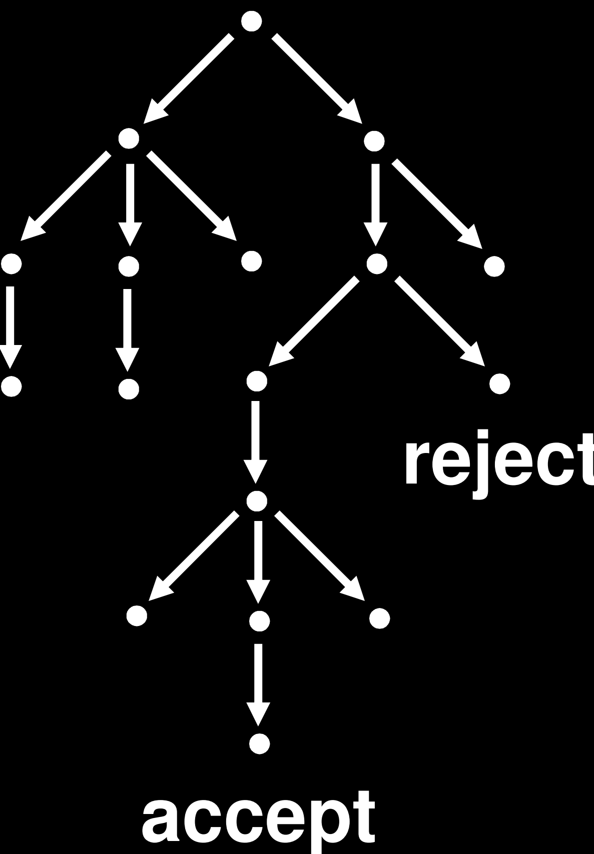
$q_{accept} \in Q$ is the accept state

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

Deterministic Computation



Nondeterministic Computation



Defining Acceptance for NTMs

Let N be a nondeterministic Turing machine

An **accepting computation history for N on w** is a sequence of configurations $C_0, C_1, \ldots, C_t$ where

1. C_0 is the start configuration q_0w,
2. C_t is an accepting configuration,
3. Each configuration C_i yields C_{i+1}

N(w) accepts in t time ⇔ such a history exists.

N has time complexity T(n) if for all n, for all inputs of length n and for all histories, N halts in T(n) time

Definition: NTIME(t(n)) =

{ L | L is decided by a O(t(n)) time nondeterministic Turing machine }

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

Is TIME(t(n)) = NTIME(t(n)) for all t(n)?

THIS IS AN OPEN QUESTION!

Boolean Formulas

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

$$\phi = (\neg x \wedge y) \vee z$$

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

$$\neg(x \vee y) \wedge (z \wedge \neg x)$$

0 0 1 0

A Boolean formula is **satisfiable** if there exists a true/false setting to the variables that makes the formula true

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

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

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

A Boolean formula is satisfiable if there exists a true/false setting to the variables that makes the formula true

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 has the form:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee x_2 \vee x_5) \wedge (x_3 \vee \neg x_2 \vee \neg x_1)$$

literals

clauses

Ex: $(x_1 \vee \neg x_2 \vee x_1)$

$(x_3 \vee x_1) \wedge (x_3 \vee \neg x_2 \vee \neg x_1)$

$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_4 \vee x_2 \vee x_1) \vee (x_3 \vee x_1 \vee \neg x_1)$

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_3 \wedge \neg x_2 \wedge \neg x_1)$

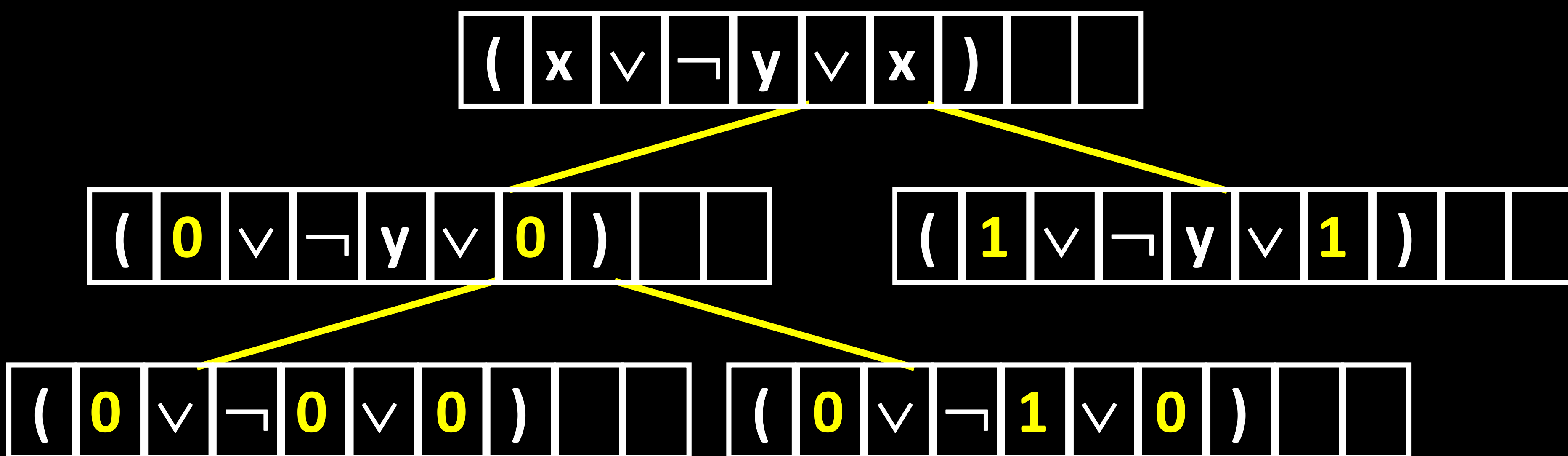
3SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

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 v in ϕ, nondeterministically substitute either 0 or 1 in place of v



3. Evaluate the formula and *accept* iff ϕ is true

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

Nondeterministic Polynomial Time

Theorem: $L \in NP \Leftrightarrow$ There is a constant k and polynomial-time TM V such that

$$L = \{ x \mid \exists\, y \in \Sigma^* \; [\, |y| \leq |x|^k \text{ and } V(x,y) \text{ accepts }] \}$$

Proof:

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

Nondeterministically guess y and then run V(x,y)

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

Let N be a nondeterministic poly-time TM that decides L. Define V(x,y) to accept iff y encodes an accepting computation history of N on x