# 6.045

## Lecture 14:
## Time Complexity
## and P vs NP

# Recognizability via Decidability

**Def.** A **decidable predicate R(x,y)** is a proposition about the input strings x and y, such that some TM M implements R. That is,

for all x, y, R(x,y) is TRUE $\Rightarrow$ M(x,y) accepts
R(x,y) is FALSE $\Rightarrow$ M(x,y) rejects

Can think of R as a function
$R : \Sigma^* \times \Sigma^* \to \{ \text{True}, \text{False} \}$

EXAMPLES: R(x,y) = "xy has at most 100 zeroes"
R(N,y) = "TM N halts on y in at most 99 steps"

**Theorem:** A language A is *recognizable* if and only if there is a decidable predicate R(x, y) such that:

$$A = \{ x \mid (\exists y \in \Sigma^*)[R(x, y) \text{ is true}]\}$$

**Proof:** (1) If $A = \{ x \mid \exists y\, R(x,y) \}$ then A is recognizable

Define the TM M(x): For all strings $y \in \Sigma^*$,
If R(x,y) is true, accept.

Then, M accepts exactly those x s.t. $\exists y\, R(x,y)$ is true

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

Suppose TM M recognizes A.
Let R(x,y) be TRUE iff M accepts x in |y| steps
Then, M accepts x $\Leftrightarrow$ $\exists y\, R(x,y)$ is true

**Example:** L = { ⟨M⟩ | TM M accepts at least one string} is recognizable.

Want: decidable predicate R such that

$L = \{ \langle M \rangle \mid \exists y \in \Sigma^* \; R(\langle M \rangle, y) \text{ is true } \}$

**Define** $R(\langle M \rangle, \langle x,y \rangle)$ = "TM M accepts string x in |y| steps"

**Note that R is decidable!**

Just run a universal TM on ⟨M, x⟩ for |y| steps

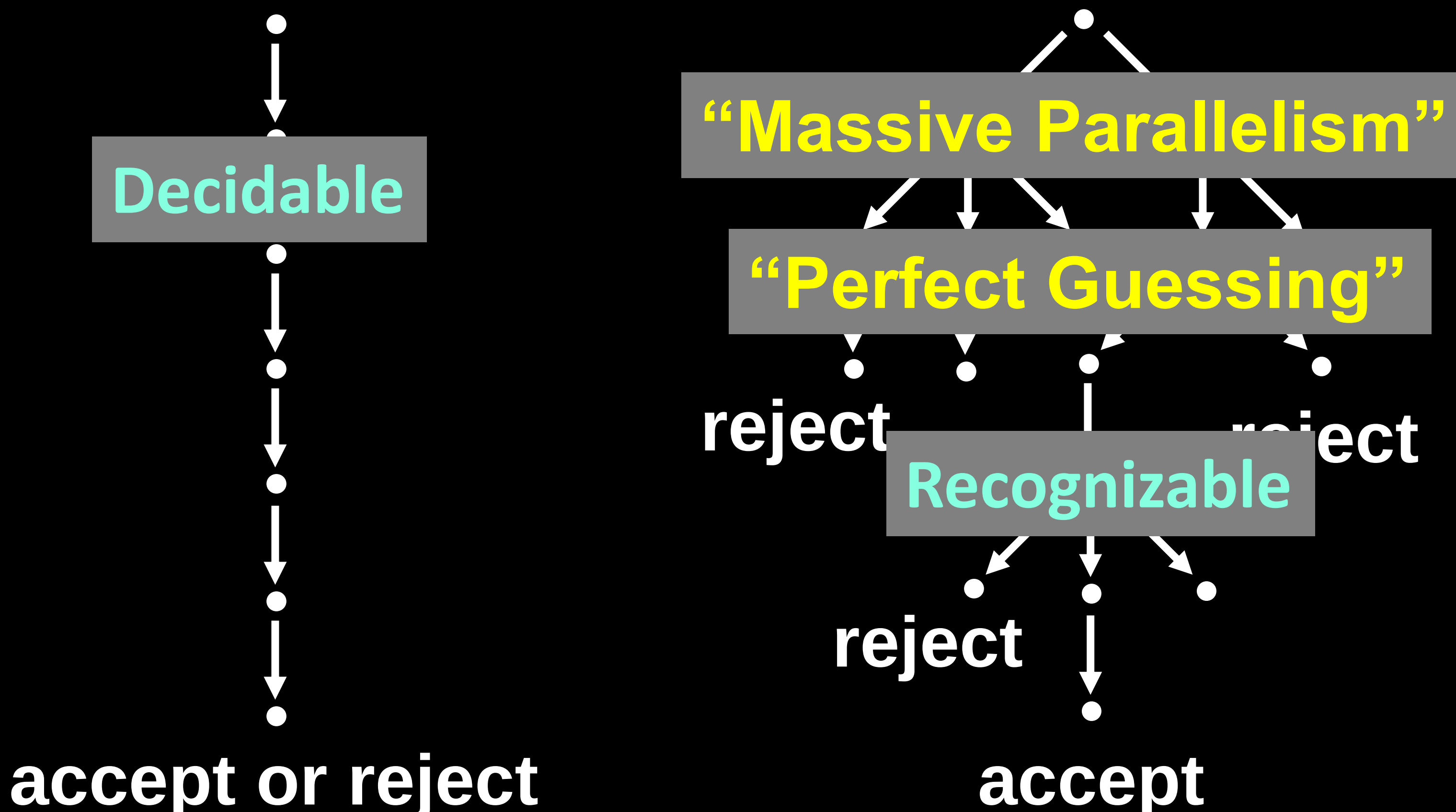
Then: $L = \{ \langle M \rangle \mid \exists \langle x,y \rangle \in \Sigma^* \; R(\langle M \rangle, \langle x,y \rangle) \text{ is true}\}$

Therefore, L is recognizable!

Can always recognize L by

"guessing ⟨x,y⟩ and verifying in finite time"

## Deterministic Computation | Non-Deterministic Computation




**Are these equally powerful???**

YES for finite automata, NO for Turing machines!

# 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 running time $\leq$ c $t(n)$ + c, for some $c \geq 1$ }

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

Puzzle: Show $A \notin \text{TIME}((n \log n)/\log\log n)$

# An Efficient 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 $U$ on $\langle M, w, 1^t \rangle$ halts in $O(|M|^2\, t^2)$ steps and $U$ accepts $\langle M, w, 1^t \rangle \Leftrightarrow$ $M$ 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.

Each step of M on w is $O(|M|)$ steps of U'

# 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, $g(n) > n^2 f(n)^2$, $\mathrm{TIME}(f(n)) \subsetneq \mathrm{TIME}(g(n))$

**Proof Idea:** Diagonalization with a clock

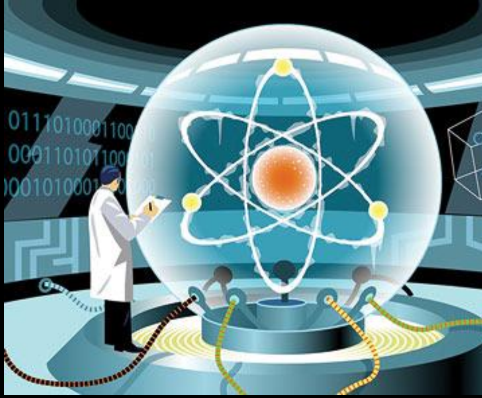
Make TM N that on input $\langle M \rangle$,
simulates the TM M on input $\langle M \rangle$ for $f(|M|)$ steps,
*then* flips the answer.

We showed L(N) cannot have time complexity $f(n)$
And there is a TM running in $O(g(n))$ time for L(N)

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

**Polynomial Time**

**The analogue of "decidability" in the world of complexity theory**



# The EXTENDED Church-Turing Thesis

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

**A controversial (dead?) thesis!**

*Counterexamples include $n^{100}$ time algorithms, randomized algorithms, quantum algorithms, …*

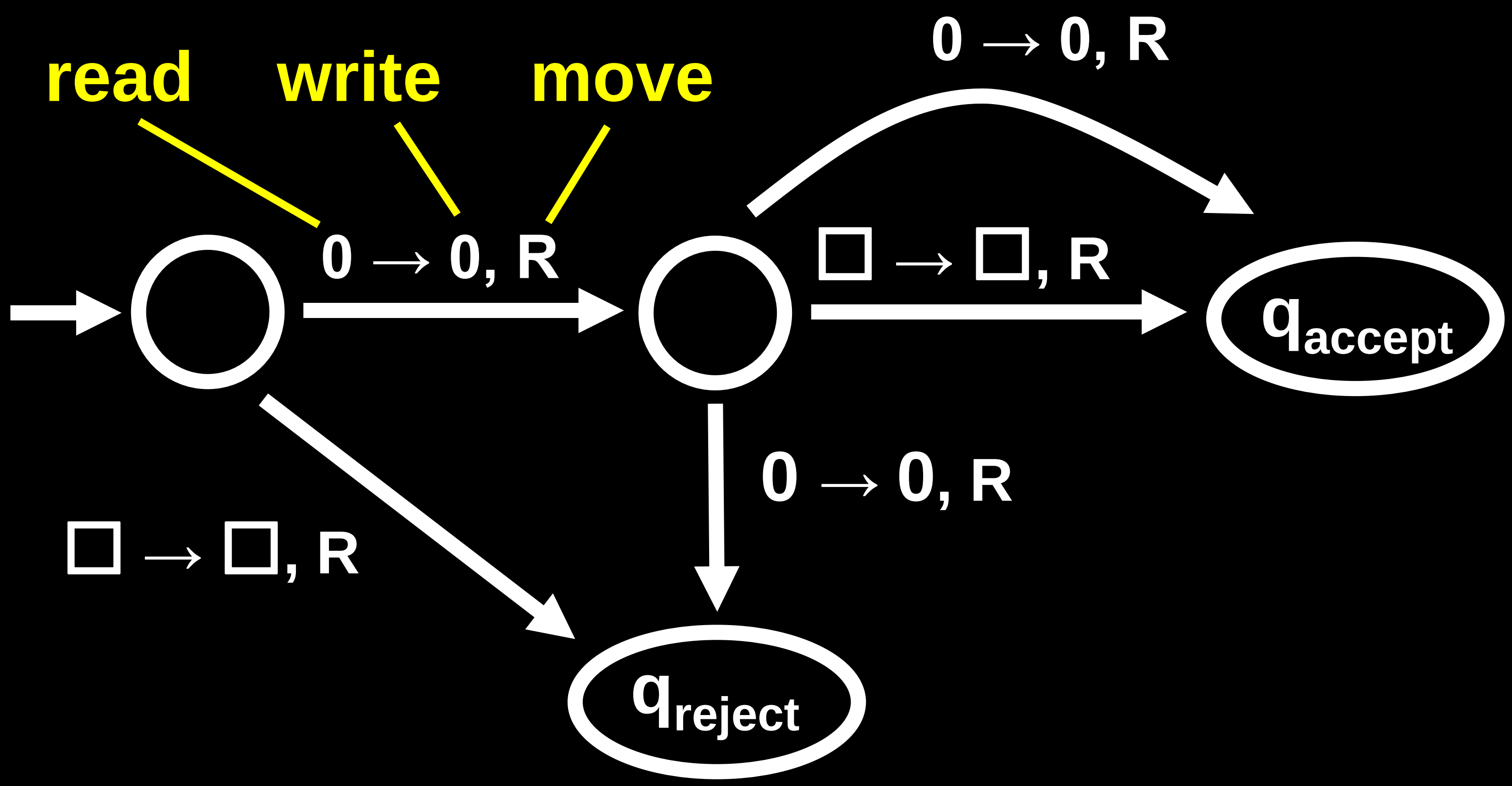
# Nondeterminism and NP

# 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


read
write
move
0 → 0, R
0 → 0, R
□ → □, R
0 → 0, R
□ → □, R
q accept
q reject

**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

$\Sigma$ is the input alphabet, where $\Box \notin \Sigma$

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

# 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 $q_0 w$,
2. $C_t$ is an accepting configuration,
3. Each configuration $C_i$ yields $C_{i+1}$

**Def.** $N(w)$ accepts in $t$ time $\Leftrightarrow$ 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 }

**Note: TIME(t(n)) $\subseteq$ NTIME(t(n))**

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

**THIS IS AN OPEN QUESTION!**

**What can be done in “short” NTIME that cannot be done in “short” TIME?**

# Boolean Formulas

logical operations

parentheses

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 $\phi$

Boolean variables (0 or 1)

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

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 $\quad a \land b \land c \land \neg d$

NO $\quad \neg(x \lor y) \land x$

SAT = { $\phi$ | $\phi$ is a satisfiable Boolean formula }

(Q: How are we encoding formulas? A: In a "reasonable" way!)

**Encoding:** takes formula $\phi$ of $n$ symbols, and outputs O($n^c$) bits

**Decoding:** takes O($n^c$) bits and $i$, and outputs $i$-th symbol of $\phi$

A 3cnf-formula has the form:

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

literals

clauses

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

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

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

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

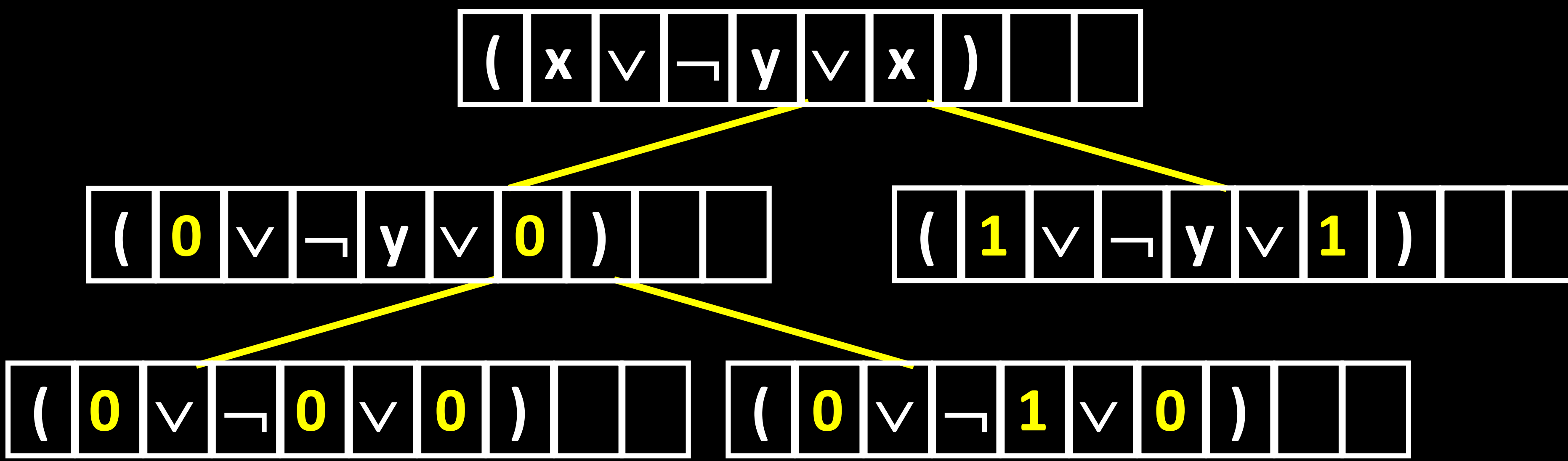
3SAT = { $\phi$ | $\phi$ is a satisfiable 3cnf-formula }

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

**Theorem: 3SAT ∈ NTIME($n^c$) for some constant c > 1**

**Proof Idea:** 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 with 0-1s all plugged in, *accept* iff ϕ is true

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

**Nondeterministic Polynomial Time**

**The analogue of "recognizability" in complexity theory**

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

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

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

Given the poly-time TM V, our NP machine for L is:

N(x): Nondeterministically guess y.
Run V(x,y) and output its answer.

(2) If $L \in NP$ then

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

Let N be a nondet. poly-time TM that decides L.

Define a TM V(x,y) which accepts

$\Leftrightarrow$ y encodes an accepting computation history of N on x

**Moral:** A language L is in NP
if and only if
there are polynomial-length proofs
for membership in L

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

SAT = { $\phi$ | $\exists$y such that $\phi$ is a Boolean formula and
y is a satisfying assignment to $\phi$ }

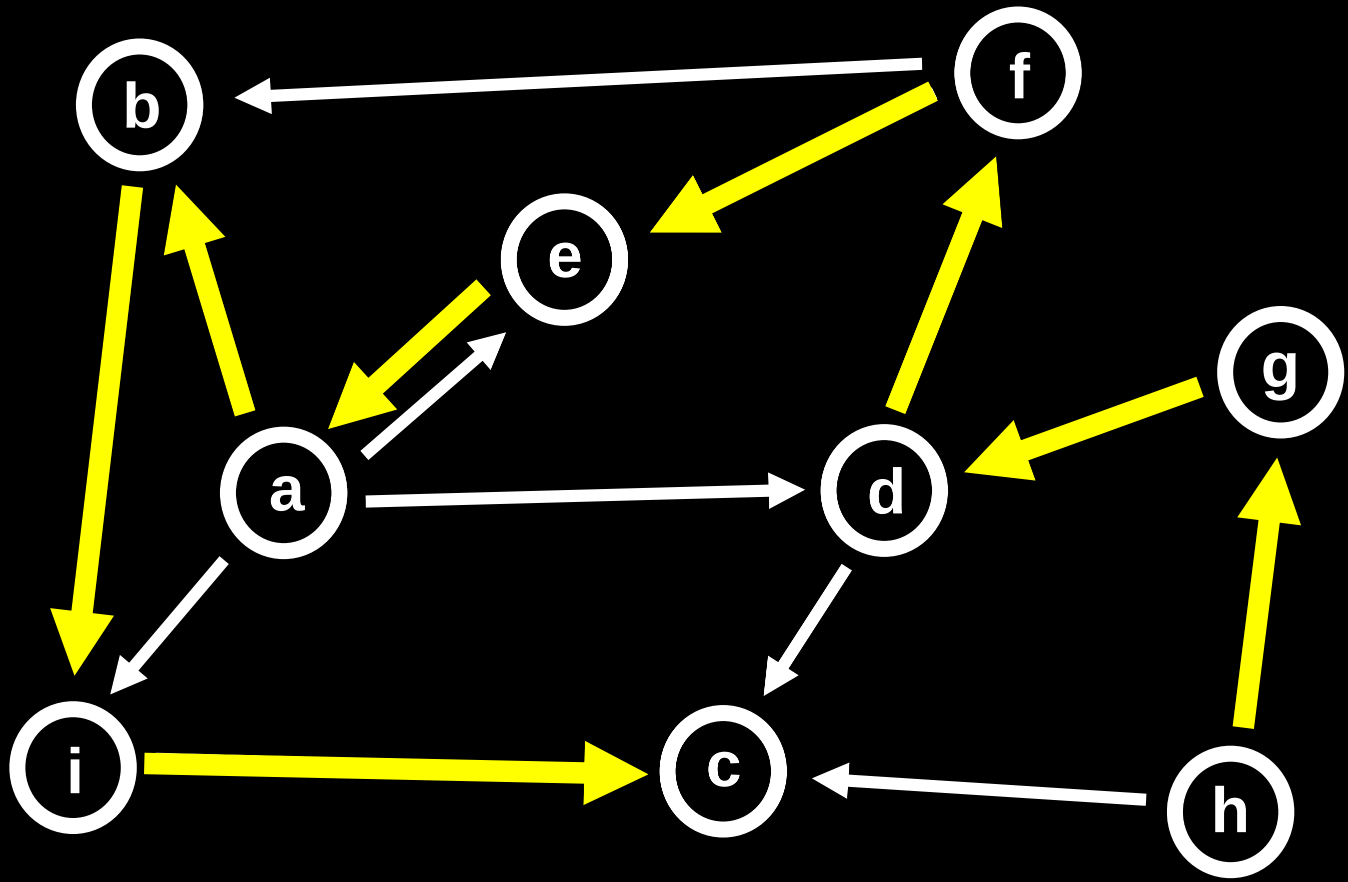
**NP = "Nifty Proofs"**

**NP** $\approx$ **Problems with the property that, once you *have* a *solution*, it is "easy" to verify the solution**

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

**When $\phi \in$ SAT, I can prove that fact to you with a short proof you can quickly verify**

# The Hamiltonian Path Problem




A Hamiltonian path traverses through each node exactly once

**Assume a reasonable encoding of graphs (example: the adjacency matrix is reasonable)**

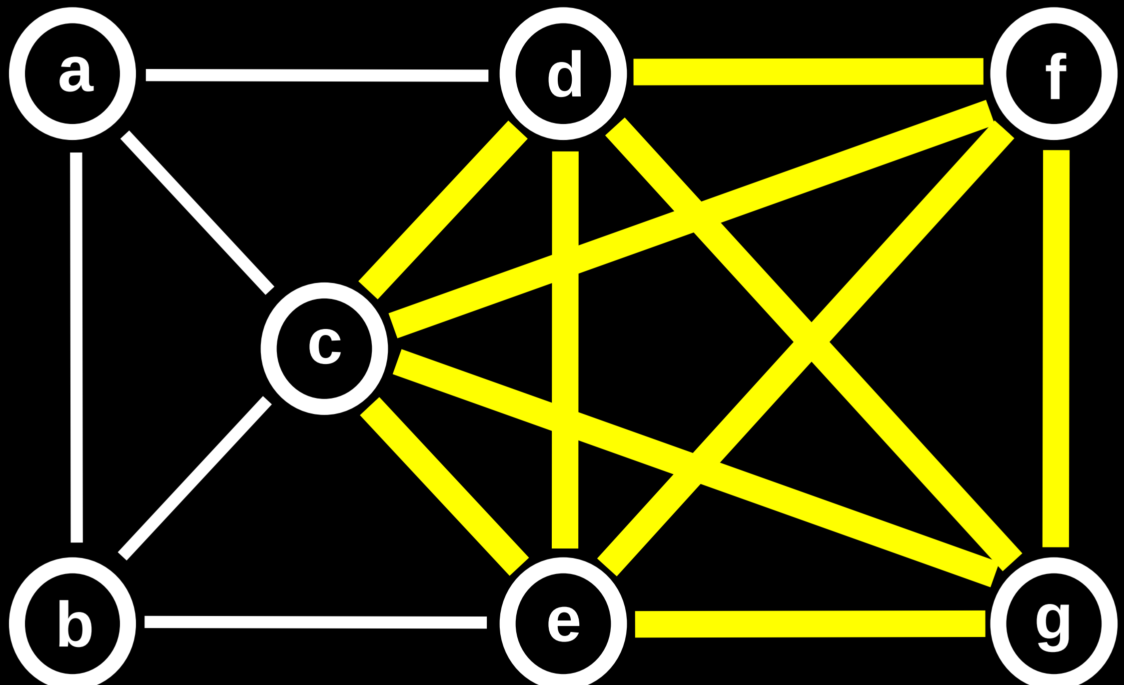
**HAMPATH = { (G,s,t) | G is a directed graph with a Hamiltonian path from s to t }**

**Theorem: HAMPATH $\in$ NP**

**A Hamiltonian path P in G from s to t is a proof that (G,s,t) is in HAMPATH**

**Given P (as a permutation on the nodes) can easily check that it is a path through all nodes exactly once**

# The k-Clique Problem




k-clique = complete subgraph on k nodes

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

Theorem: CLIQUE $\in$ NP

A k-clique in G is a proof that (G, k) is in CLIQUE

Given a subset S of k nodes from G, we can efficiently check that all possible edges are present between the nodes in S

**A language is in NP if and only if there are "polynomial-length proofs" for membership in the language**

**P** $\approx$ **the problems that can be *efficiently solved***

**NP** $\approx$ **the problems where *proposed solutions can be efficiently verified***

# Is P = NP?

***Can problem solving be automated?***

# $$$

# P = NP?

# $$$

# If P = NP…

Mathematicians/creators may be out of a job

This problem is in NP:

Short-Provability$_{\mathcal{F}}$

= { (T, $1^k$) | T has a proof in $\mathcal{F}$ of length $\leq k$}

Cryptography as we know it may be impossible – there are no "one-way" functions!

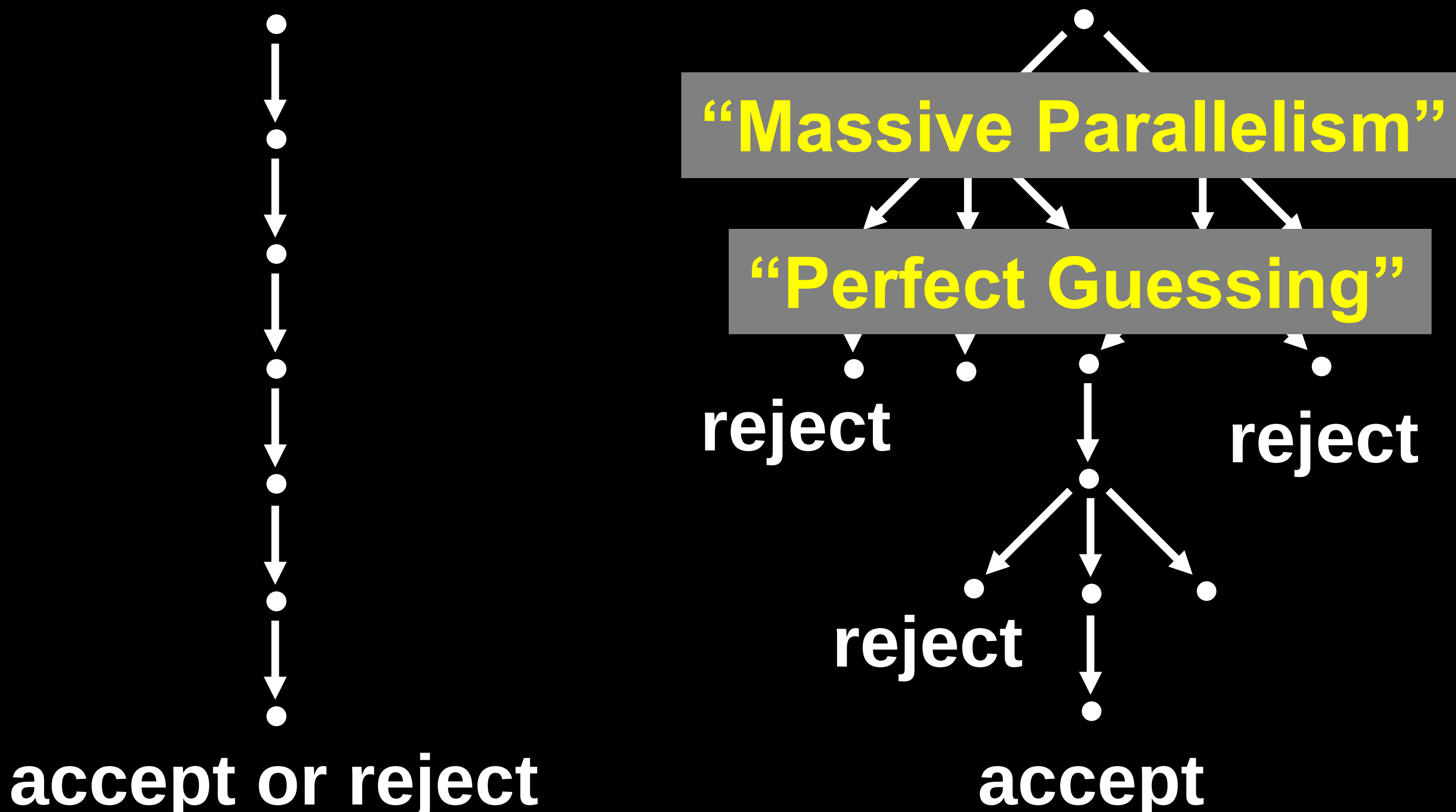
Machines could effectively learn *any concept with a short description*

In principle, every aspect of daily life could be efficiently and globally optimized…

… life as we know it would be different

## Conjecture: P $\neq$ NP

## Deterministic Computation | Non-Deterministic Computation




**Are these equally powerful???**

YES for FAs, NO for TMs, OPEN for Polynomial Time