## Recursively Enumerable Languages: Examples

- The set of C program-input pairs that do not run into an infinite loop is recursively enumerable.
  - Just run its binary code in a simulator environment.
- The set of C programs that contain an infinite loop is *not* recursively enumerable (see p. 135).

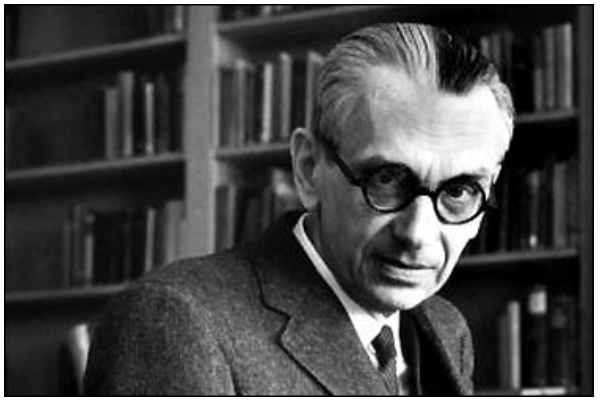
## Turing-Computable Functions

- Let $f : (\Sigma - \{\bigsqcup\})^* \to \Sigma^*$.
  - Optimization problems, root finding problems, etc.
- Let $M$ be a TM with alphabet $\Sigma$.
- $M$ **computes** $f$ if for any string $x \in (\Sigma - \{\bigsqcup\})^*$, $M(x) = f(x)$.
- We call $f$ a **recursive function**[a] if such an $M$ exists.

[a] Kurt Gödel (1931, 1934).

## Kurt Gödel[a] (1906–1978)

Quine (1978), "this theorem [$\cdots$] sealed his immortality."



[a]This photo was taken by Alfred Eisenstaedt (1898–1995).

## Church's Thesis or the Church-Turing Thesis

- What is computable is Turing-computable; TMs are algorithms.[a]
- No "intuitively computable" problems have been shown not to be Turing-computable, yet.

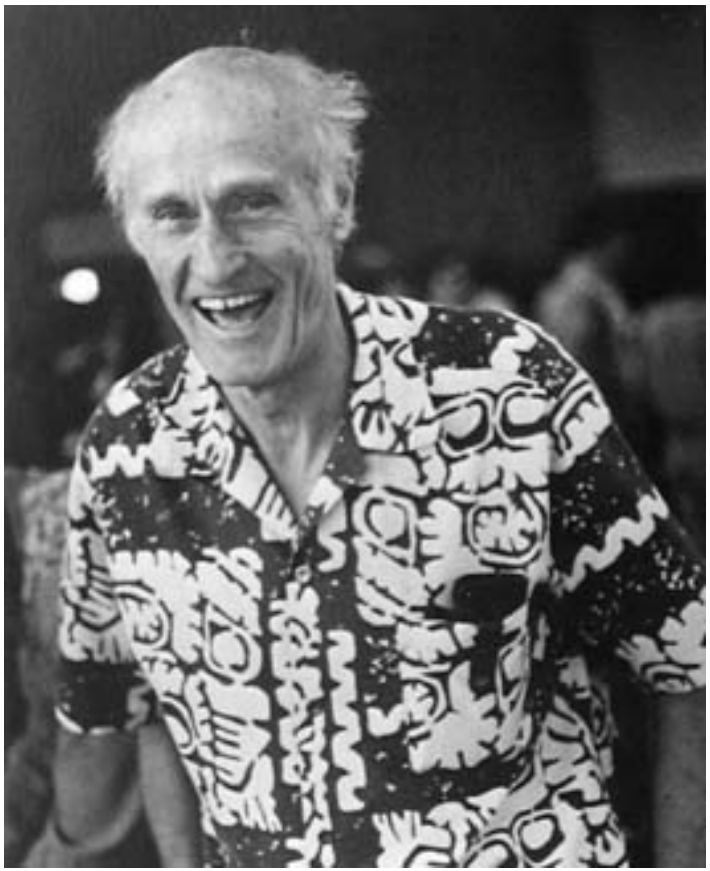
[a] Church (1936); Kleene (1953).

## Church's Thesis or the Church-Turing Thesis (concluded)

- Many other computation models have been proposed.
  - Recursive function (Gödel), $\lambda$ calculus (Church), formal language (Post), assembly language-like RAM (Shepherdson & Sturgis), boolean circuits (Shannon), extensions of the Turing machine (more strings, two-dimensional strings, and so on), etc.
- All have been proved to be equivalent.

# Alonso Church (1903–1995)

# Stephen Kleene (1909–1994)



## Extended Church's Thesis[a]

- All "reasonably succinct encodings" of problems are *polynomially related* (e.g., $n^2$ vs. $n^6$).
  - Representations of a graph as an adjacency matrix and as a linked list are both succinct.
  - The *unary* representation of numbers is not succinct.
  - The *binary* representation of numbers is succinct.
    * $1001_2$ vs. $111111111_1$.
- All numbers for TMs will be binary from now on.

[a]Some call it "polynomial Church's thesis," which Lószló Lovász attributed to Leonid Levin.

## Extended Church's Thesis (concluded)

- Representations that are not succinct may give misleadingly low complexities.
  - Consider an algorithm with binary inputs that runs in $2^n$ steps.
  - Suppose the input uses unary representation instead.
  - Then the same algorithm runs in linear time because the input length is now $2^n$!
- So a succinct representation is for honest accounting.

## Physical Church-Turing Thesis

- "[Church's thesis] is a profound claim about the physical laws of our universe, i.e.: any physical system that purports to be a 'computer' is not capable of any computational task that a Turing machine is incapable of."[a]
- "Anything computable in physics can also be computed on a Turing machine."[b]
- The universe is a Turing machine.[c]

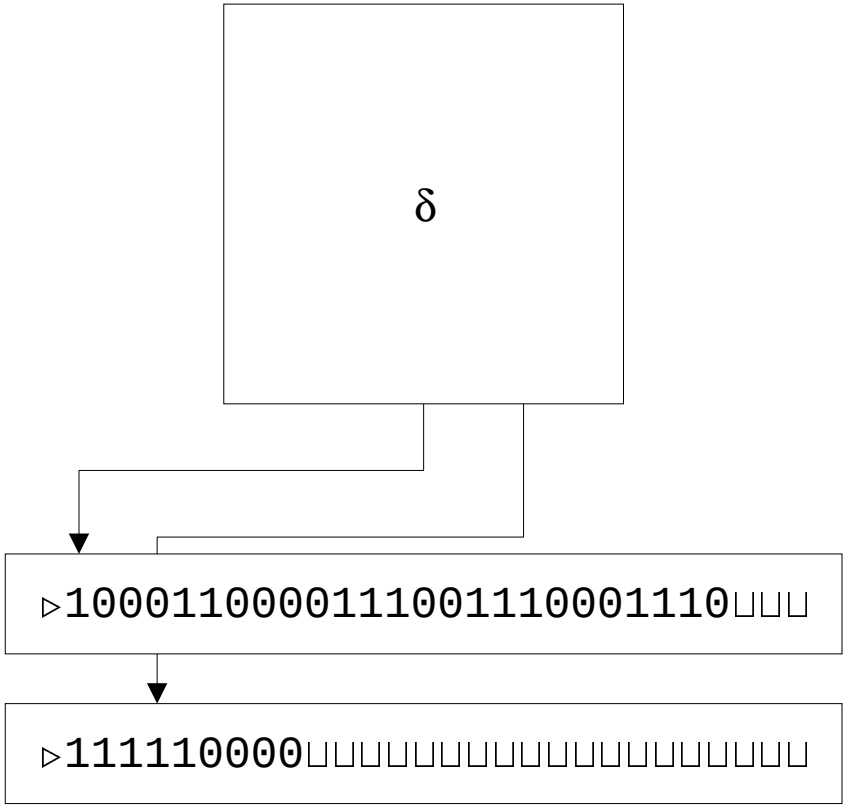
---

[a] Warren Smith (1998).
[b] Cooper (2012).
[c] Edward Fredkin's (1992) digital physics.

## Turing Machines with Multiple Strings

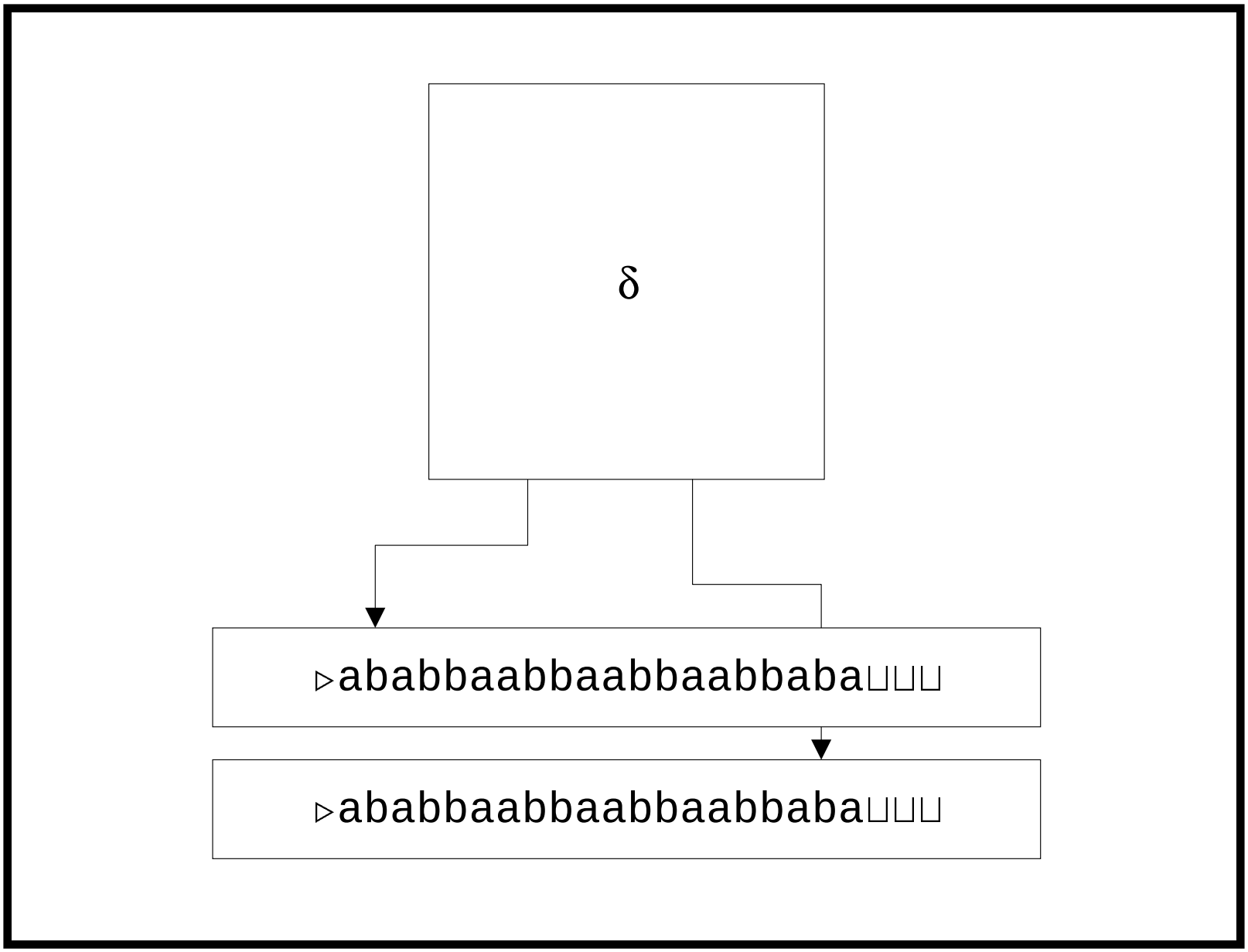
- A $k$-string Turing machine (TM) is a quadruple $M = (K, \Sigma, \delta, s)$.
- $K, \Sigma, s$ are as before.
- $\delta : K \times \Sigma^k \to (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k$.
- All strings start with a $\triangleright$.
- The first string contains the input.
- Decidability and acceptability are the same as before.
- When TMs compute functions, the output is the last ($k$th) string.

# A 2-String TM



## PALINDROME Revisited

- A 2-string TM can decide PALINDROME in $O(n)$ steps.
  - It copies the input to the second string.
  - The cursor of the first string is positioned at the first symbol of the input.
  - The cursor of the second string is positioned at the last symbol of the input.
  - The symbols under the cursors are then compared.
  - The two cursors are then moved in opposite directions until the ends are reached.
  - The machine accepts if and only if the symbols under the two cursors are identical at all steps.


δ
▷ababbaabbaabbaabbaba␣␣␣
▷ababbaabbaabbaabbaba␣␣␣

## PALINDROME Revisited (concluded)

- The running times of a 2-string TM and a single-string TM are quadratically related: $n^2$ vs. $n$.
- This is consistent with extended Church's thesis.

## Configurations and Yielding

- The concept of configuration and yielding is the same as before except that a configuration is a $(2k+1)$-tuple

$$(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k).$$

  - $w_i u_i$ is the $i$th string.
  - The $i$th cursor is reading the last symbol of $w_i$.
  - Recall that $\triangleright$ is each $w_i$'s first symbol.

- The $k$-string TM's initial configuration is

$$(s, \overbrace{\underbrace{\triangleright, x}_{1}, \underbrace{\triangleright, \epsilon}_{2}, \underbrace{\triangleright, \epsilon}_{3}, \ldots, \underbrace{\triangleright, \epsilon}_{k}}^{2k}).$$

Time seemed to be
the most obvious measure
of complexity.
— Stephen Arthur Cook (1939–)

## Time Complexity

- The multistring TM is the basis of our notion of the time expended by TMs.
- If a $k$-string TM $M$ halts after $t$ steps on input $x$, then the **time required by $M$ on input** $x$ is $t$.
- If $M(x) = \nearrow$, then the time required by $M$ on $x$ is $\infty$.

## Time Complexity (concluded)

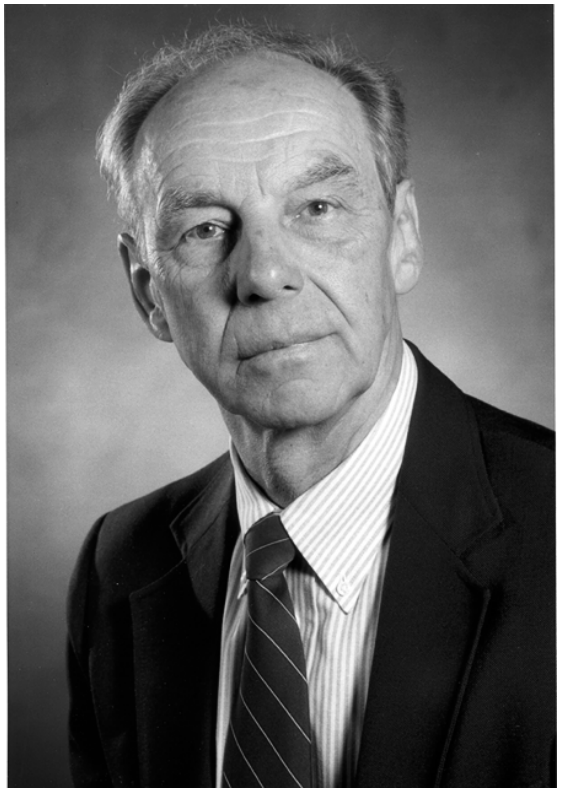
- Machine $M$ **operates within time** $f(n)$ for $f : \mathbb{N} \to \mathbb{N}$ if for any input string $x$, the time required by $M$ on $x$ is at most $f(|\, x \,|)$.
  - $|\, x \,|$ is the length of string $x$.
- Function $f(n)$ is a **time bound** for $M$.

## Time Complexity Classes[a]

- Suppose language $L \subseteq (\Sigma - \{\bigsqcup\})^*$ is decided by a multistring TM operating in time $f(n)$.
- We say $L \in \text{TIME}(f(n))$.
- $\text{TIME}(f(n))$ is the set of languages decided by TMs with multiple strings operating within time bound $f(n)$.
- $\text{TIME}(f(n))$ is a **complexity class**.
  - PALINDROME is in $\text{TIME}(f(n))$, where $f(n) = O(n)$.

[a] Hartmanis and Stearns (1965); Hartmanis, Lewis, and Stearns (1965).

# Juris Hartmanis[a] (1928–)



[a]Turing Award (1993).

# Richard Edwin Stearns[a] (1936–)

[a]Turing Award (1993).

## The Simulation Technique

**Theorem 2** *Given any $k$-string $M$ operating within time $f(n)$, there exists a (single-string) $M'$ operating within time $O(f(n)^2)$ such that $M(x) = M'(x)$ for any input $x$.*

- The single string of $M'$ implements the $k$ strings of $M$.

## The Proof

- Represent configuration $(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k)$ of $M$ by this string of $M'$:

$$(q, \triangleright w_1' u_1 \triangleleft w_2' u_2 \triangleleft \cdots \triangleleft w_k' u_k \triangleleft \triangleleft).$$

  - $\triangleleft$ is a special delimiter.
  - $w_i'$ is $w_i$ with the first[a] and last symbols "primed."
  - It serves the purpose of "," in a configuration.

[a] The first symbol is always $\triangleright$.

## The Proof (continued)

- The "priming" of the last symbol of $w_i$ ensures that $M'$ knows which symbol is under each cursor of $M$.[a]
- The first symbol of $w_i$ is the primed version of $\triangleright$: $\triangleright'$.
  - Recall TM cursors are not allowed to move to the left of $\triangleright$ (p. 21).
  - Now the cursor of $M'$ can move *between* the simulated strings of $M$.[b]

[a] Added because of comments made by Mr. Che-Wei Chang (R95922093) on September 27, 2006.

[b] Thanks to a lively discussion on September 22, 2009.

## The Proof (continued)

- The initial configuration of $M'$ is

$$(s, \rhd \rhd'' \, x \lhd \overbrace{\rhd'' \lhd \cdots \rhd'' \lhd}^{k-1 \text{ pairs}} \lhd).$$

  - $\rhd$ is double-primed because it is the beginning and the ending symbol as the cursor is reading it.[a]

[a] Added after the class discussion on September 20, 2011.

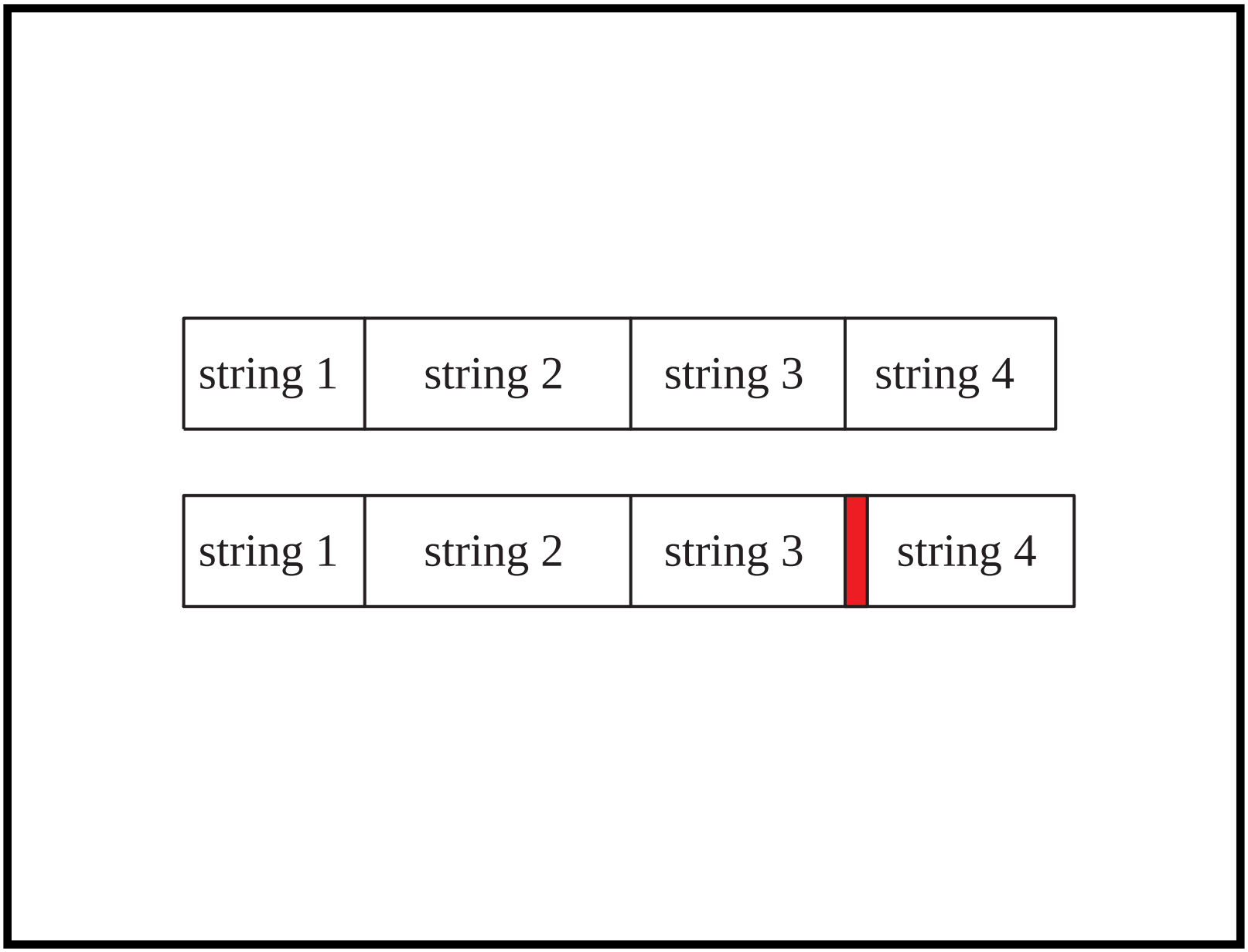
## The Proof (continued)

- We simulate each move of $M$ thus:
  1. $M'$ scans the string to pick up the $k$ symbols under the cursors.
     - The states of $M'$ must be enlarged to include $K \times \Sigma^k$ to remember them.
     - The transition functions of $M'$ must also reflect it.
  2. $M'$ then changes the string to reflect the overwriting of symbols and cursor movements of $M$.

## The Proof (continued)

- It is possible that some strings of $M$ need to be lengthened (see next page).
  - The linear-time algorithm on p. 34 can be used for each such string.
- The simulation continues until $M$ halts.
- $M'$ then erases all strings of $M$ except the last one.[a]

[a] Because whatever appears on the string of $M'$ will be the output. So those $\triangleright'$s and $\triangleright''$s need to be removed.


string 1
string 2
string 3
string 4
string 1
string 2
string 3
string 4

## The Proof (continued)

- Since $M$ halts within time $f(|\,x\,|)$, none of its strings ever becomes longer than $f(|\,x\,|)$.[a]
- The length of the string of $M'$ at any time is $O(kf(|\,x\,|))$.
- Simulating each step of $M$ takes, *per string of* $M$, $O(kf(|\,x\,|))$ steps.
  - $O(f(|\,x\,|))$ steps to collect information from this string.
  - $O(kf(|\,x\,|))$ steps to write and, if needed, to lengthen the string.

[a] We tacitly assume $f(n) \geq n$.

## The Proof (concluded)

- $M'$ takes $O(k^2 f(|\, x \,|))$ steps to simulate each step of $M$ because there are $k$ strings.
- As there are $f(|\, x \,|)$ steps of $M$ to simulate, $M'$ operates within time $O(k^2 f(|\, x \,|)^2)$.

## Linear Speedup[a]

**Theorem 3** *Let $L \in TIME(f(n))$. Then for any $\epsilon > 0$, $L \in TIME(f'(n))$, where $f'(n) = \epsilon f(n) + n + 2$.*

[a] Hartmanis and Stearns (1965).

## Implications of the Speedup Theorem

- State size can be traded for speed.[a]
- If $f(n) = cn$ with $c > 1$, then $c$ can be made arbitrarily close to 1.
- If $f(n)$ is superlinear, say $f(n) = 14n^2 + 31n$, then the constant in the leading term (14 in this example) can be made arbitrarily small.
  - *Arbitrary* linear speedup can be achieved.[b]
  - This justifies the big-O notation in the analysis of algorithms.

---

[a] $m^k \cdot |\Sigma|^{3mk}$-fold increase to gain a speedup of $O(m)$. No free lunch.

[b] Can you apply the theorem multiple times to achieve superlinear speedup? Thanks to a question by a student on September 21, 2010.

# P

- By the linear speedup theorem, any polynomial time bound can be represented by its leading term $n^k$ for some $k \geq 1$.
- If $L$ is a **polynomially decidable language**, it is in $\text{TIME}(n^k)$ for some $k \in \mathbb{N}$.
  - Clearly, $\text{TIME}(n^k) \subseteq \text{TIME}(n^{k+1})$.
- The union of all polynomially decidable languages is denoted by P:

$$\text{P} = \bigcup_{k>0} \text{TIME}(n^k).$$

- P contains problems that can be efficiently solved.

Philosophers have explained space.
They have not explained time.
— Arnold Bennett (1867–1931),
*How To Live on 24 Hours a Day* (1910)

I keep bumping into that silly quotation
attributed to me that says
640K of memory is enough.
— Bill Gates (1996)

## Space Complexity

- Consider a $k$-string TM $M$ with input $x$.
- Assume non-$\sqcup$ is never written over by $\sqcup$.[a]
  - The purpose is not to artificially reduce the space needs (see below).
- If $M$ halts in configuration $(H, w_1, u_1, w_2, u_2, \ldots, w_k, u_k)$, then the **space required by $M$ on input $x$** is

$$\sum_{i=1}^{k} |w_i u_i|.$$

[a] Corrected by Ms. Chuan-Ju Wang (`R95922018`, `F95922018`) on September 27, 2006.

## Space Complexity (continued)

- Suppose we do not charge the space used only for input and output.
- Let $k > 2$ be an integer.
- A **$k$-string Turing machine with input and output** is a $k$-string TM that satisfies the following conditions.
  - The input string is *read-only*.
  - The last string, the output string, is *write-only*.
  - So the cursor never moves to the left.
  - The cursor of the input string does not wander off into the $\sqcup$s.

## Space Complexity (concluded)

- If $M$ is a TM with input and output, then the space required by $M$ on input $x$ is

$$\sum_{i=2}^{k-1} |w_i u_i|.$$

- Machine $M$ **operates within space bound** $f(n)$ for $f : \mathbb{N} \to \mathbb{N}$ if for any input $x$, the space required by $M$ on $x$ is at most $f(|\, x \,|)$.

## Space Complexity Classes

- Let $L$ be a language.
- Then

  $$L \in \text{SPACE}(f(n))$$

  if there is a TM with input and output that decides $L$ and operates within space bound $f(n)$.
- $\text{SPACE}(f(n))$ is a set of languages.
  - PALINDROME $\in \text{SPACE}(\log n)$.[a]
- As in the linear speedup theorem (p. 76), constant coefficients do not matter.

[a] Keep 3 counters.

## Nondeterminism[a]

- A **nondeterministic Turing machine** (**NTM**) is a quadruple $N = (K, \Sigma, \Delta, s)$.
- $K, \Sigma, s$ are as before.
- $\Delta \subseteq K \times \Sigma \times (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$ is a relation, not a function.[b]
  - For each state-symbol combination $(q, \sigma)$, there may be multiple valid next steps.
  - Multiple lines of code may be applicable.

---

[a] Rabin and Scott (1959).

[b] Corrected by Mr. Jung-Ying Chen (`D95723006`) on September 23, 2008.

## Nondeterminism (concluded)

- As before, a program contains lines of code:

$$\begin{aligned}(q_1, \sigma_1, p_1, \rho_1, D_1) &\in \Delta,\\ (q_2, \sigma_2, p_2, \rho_2, D_2) &\in \Delta,\\ &\vdots\\ (q_n, \sigma_n, p_n, \rho_n, D_n) &\in \Delta.\end{aligned}$$

  - We cannot write

$$\delta(q_i, \sigma_i) = (p_i, \rho_i, D_i)$$

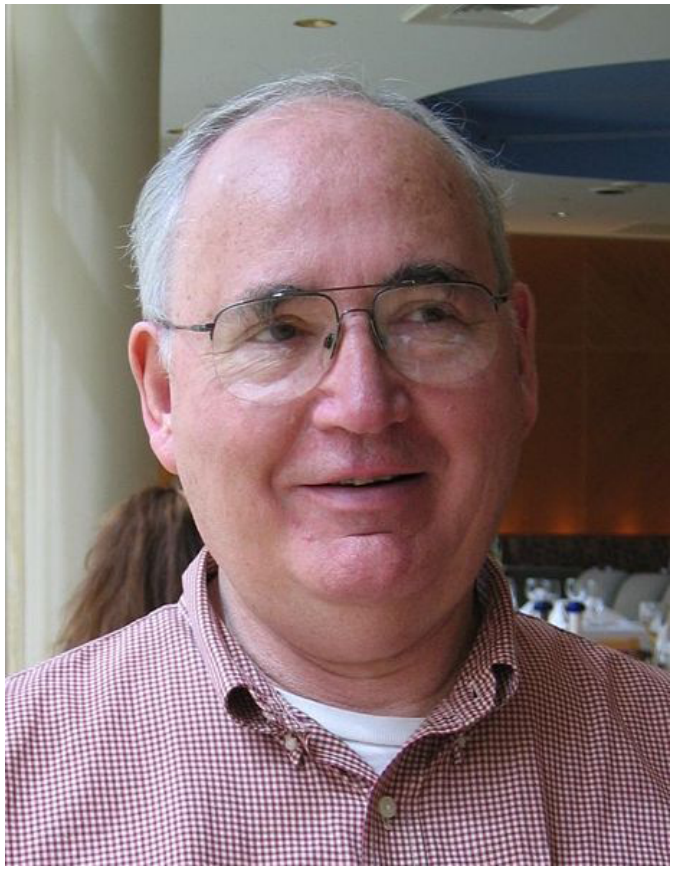
    as in the deterministic case (p. 22) anymore.

- A configuration yields another configuration in one step if there *exists* a rule in $\Delta$ that makes this happen.

# Michael O. Rabin[a] (1931–)

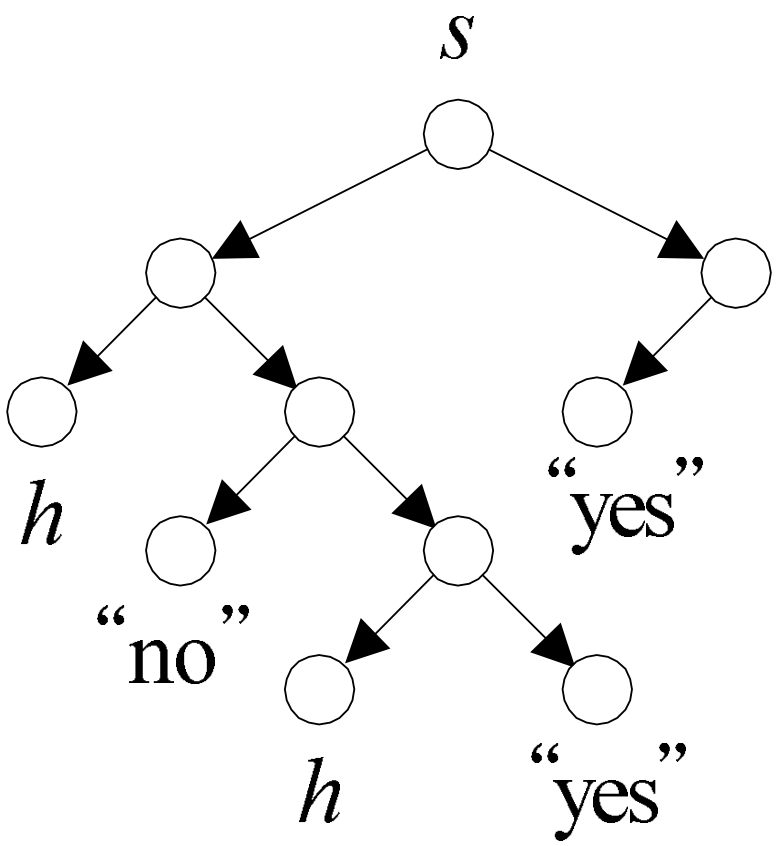
[a]Turing Award (1976).

# Dana Stewart Scott[a] (1932–)



[a]Turing Award (1976).

# Computation Tree and Computation Path



## Decidability under Nondeterminism

- Let $L$ be a language and $N$ be an NTM.
- $N$ **decides** $L$ if for any $x \in \Sigma^*$, $x \in L$ if and only if there is a sequence of valid configurations that ends in "yes."
- In other words,
  - If $x \in L$, then $M(x) =$ "yes" for some computation path.
  - If $x \notin L$, then $M(x) \neq$ "yes" for all computation paths.

## Decidability under Nondeterminism (concluded)

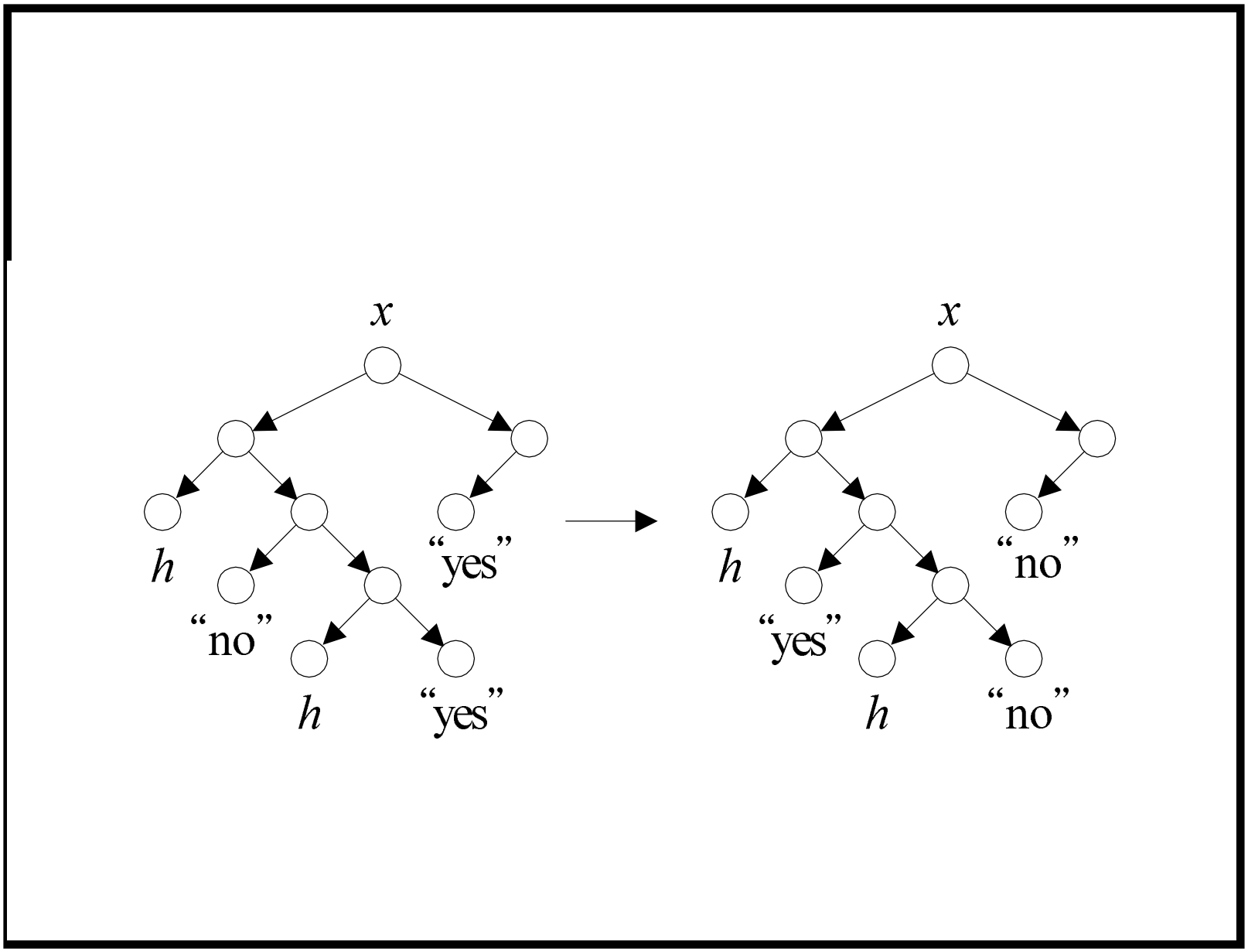
- It is not required that the NTM halts in all computation paths.[a]
- If $x \notin L$, no nondeterministic choices should lead to a "yes" state.
- The key is the algorithm's *overall* behavior not whether it gives a correct answer for each particular run.
- Determinism is a special case of nondeterminism.

---

[a] So "accepts" is a more proper term, and other books use "decides" only when the NTM always halts.

## Complementing a TM's Halting States

- Let $M$ decide $L$, and $M'$ be $M$ after "yes" $\leftrightarrow$ "no".
- If $M$ is a deterministic TM, then $M'$ decides $\bar{L}$.
  - So $M$ and $M'$ decide languages that are complements of each other.
- But if $M$ is an NTM, then $M'$ may not decide $\bar{L}$.
  - It is possible that both $M$ and $M'$ accept $x$ (see next page).
  - So $M$ and $M'$ accept languages that are not complements of each other.


x
h
“yes”
“no”
h
“yes”
x
h
“no”
“yes”
h
“no”

## Time Complexity under Nondeterminism

- Nondeterministic machine $N$ decides $L$ **in time** $f(n)$, where $f : \mathbb{N} \to \mathbb{N}$, if
  - $N$ decides $L$, and
  - for any $x \in \Sigma^*$, $N$ does not have a computation path longer than $f(|\, x \,|)$.
- We charge only the "depth" of the computation tree.

## Time Complexity Classes under Nondeterminism

- NTIME($f(n)$) is the set of languages decided by NTMs within time $f(n)$.
- NTIME($f(n)$) is a complexity class.

## NP

- Define
$$\mathrm{NP} = \bigcup_{k>0} \mathrm{NTIME}(n^k).$$
- Clearly $\mathrm{P} \subseteq \mathrm{NP}$.
- Think of NP as efficiently *verifiable* problems (see p. 300).
  - Boolean satisfiability (p. 99 and p. 171).
- The most important open problem in computer science is whether $\mathrm{P} = \mathrm{NP}$.

## Simulating Nondeterministic TMs

Nondeterminism does not add power to TMs.

**Theorem 4** *Suppose language $L$ is decided by an NTM $N$ in time $f(n)$. Then it is decided by a 3-string deterministic TM $M$ in time $O(c^{f(n)})$, where $c > 1$ is some constant depending on $N$.*

- On input $x$, $M$ goes down every computation path of $N$ using depth-first search.
  - $M$ does *not* need to know $f(n)$.
  - As $N$ is time-bounded, the depth-first search will not run indefinitely.

## The Proof (concluded)

- If any path leads to "yes," then $M$ immediately enters the "yes" state.
- If none of the paths leads to "yes," then $M$ enters the "no" state.
- The simulation takes time $O(c^{f(n)})$ for some $c > 1$ because the computation tree has that many nodes.

**Corollary 5** $\mathrm{NTIME}(f(n))) \subseteq \bigcup_{c>1} \mathrm{TIME}(c^{f(n)})$.

## NTIME vs. TIME

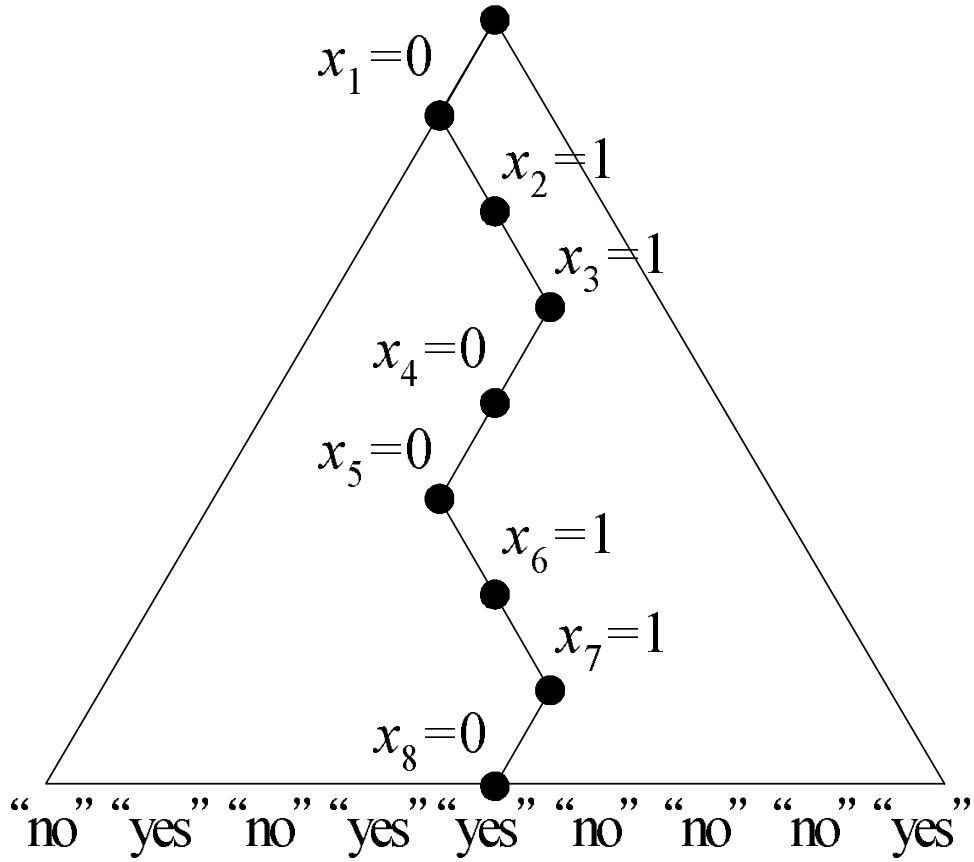
- Does converting an NTM into a TM require exploring all computation paths of the NTM as done in Theorem 4 (p. 96)?
- This is the most important question in theory with important practical implications.

## A Nondeterministic Algorithm for Satisfiability

$\phi$ is a boolean formula with $n$ variables.

1: **for** $i = 1, 2, \ldots, n$ **do**
2: Guess $x_i \in \{0, 1\}$; {Nondeterministic choice.}
3: **end for**
4: {Verification:}
5: **if** $\phi(x_1, x_2, \ldots, x_n) = 1$ **then**
6: "yes";
7: **else**
8: "no";
9: **end if**

# Computation Tree for Satisfiability



## Analysis

- The computation tree is a complete binary tree of depth $n$.
- Every computation path corresponds to a particular truth assignment out of $2^n$.
- $\phi$ is satisfiable iff there is a truth assignment that satisfies $\phi$.

## Analysis (concluded)

- The algorithm decides language $\{\phi : \phi \text{ is satisfiable}\}$.
  - Suppose $\phi$ is satisfiable.
    - That means there is a truth assignment that satisfies $\phi$.
    - So there is a computation path that results in "yes."
  - Suppose $\phi$ is not satisfiable.
    - That means every truth assignment makes $\phi$ false.
    - So every computation path results in "no."
- General paradigm: Guess a "proof" and then verify it.

## The Traveling Salesman Problem

- We are given $n$ cities $1, 2, \ldots, n$ and integer distance $d_{ij}$ between any two cities $i$ and $j$.
- Assume $d_{ij} = d_{ji}$ for convenience.
- The **traveling salesman problem** (TSP) asks for the total distance of the shortest tour of the cities.
- The decision version TSP (D) asks if there is a tour with a total distance at most $B$, where $B$ is an input.[a]

[a] Both problems are extremely important and are equally hard (p. 360 and p. 461).

## A Nondeterministic Algorithm for TSP (D)

1: **for** $i = 1, 2, \ldots, n$ **do**
2: Guess $x_i \in \{1, 2, \ldots, n\}$; {The $i$th city.}[a]
3: **end for**
4: $x_{n+1} := x_1$;
5: {Verification:}
6: **if** $x_1, x_2, \ldots, x_n$ are distinct and $\sum_{i=1}^{n} d_{x_i, x_{i+1}} \leq B$ **then**
7: "yes";
8: **else**
9: "no";
10: **end if**

[a]Can be made into a series of $\log_2 n$ *binary* choices for each $x_i$ so that the next-state count (2) is a constant, independent of input size. Contributed by Mr. Chih-Duo Hong (R95922079) on September 27, 2006.

## Analysis

- Suppose the input graph contains at least one tour of the cities with a total distance at most $B$.
- Then there is a computation path that leads to "yes."[a]
- Suppose the input graph contains no tour of the cities with a total distance at most $B$.
- Then every computation path leads to "no."

[a] It does not mean the algorithm will follow that path. It just means such a computation path exists.

## Remarks on the P $\stackrel{?}{=}$ NP Open Problem[a]

- Many practical applications depend on answers to the P $\stackrel{?}{=}$ NP question.
- Verification of password is easy (so it is in NP).
  - A computer should not take a long time to let a user log in.
- A password system should be hard to crack (loosely speaking, cracking it should not be in P).

---

[a]Contributed by Mr. Kuan-Lin Huang (B96902079, R00922018) on September 27, 2011.