Recursive and Recursively Enumerable Languages

Proposition 2 *If L is recursive, then it is recursively enumerable.*

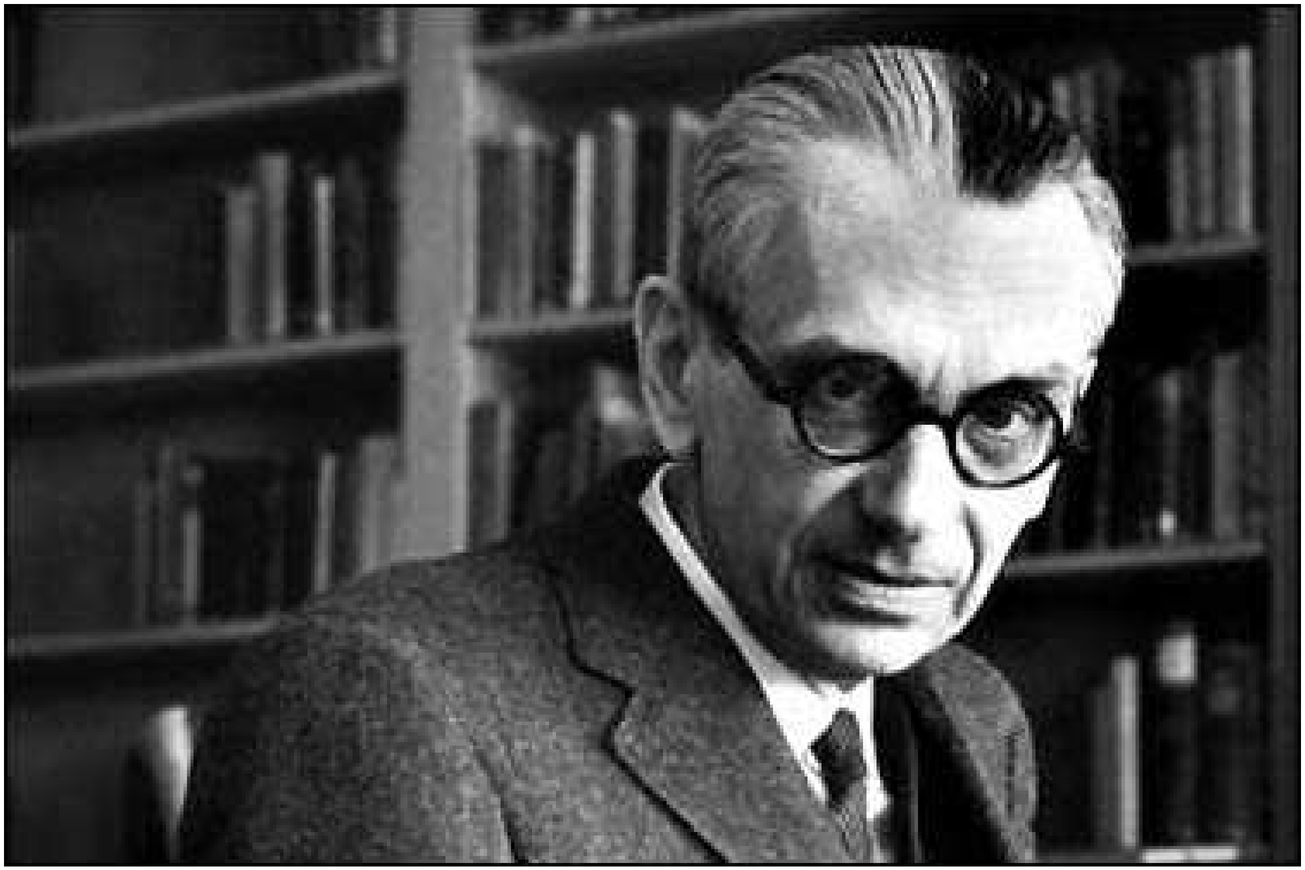
- We need to design a TM that accepts L.
- Let TM M decide L.
- We next modify M's program to obtain M' that accepts L.
- M' is identical to M except that when M is about to halt with a "no" state, M' goes into an infinite loop.
- M' accepts L.

Turing-Computable Functions

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

[a] Kurt Gödel (1931).

Kurt Gödel (1906–1978)



Church's Thesis or the Church-Turing Thesis

- What is computable is Turing-computable; TMs are algorithms (Kleene 1953).
- Many other computation models have been proposed.
  - Recursive function (Gödel), λ 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.
- No "intuitively computable" problems have been shown not to be Turing-computable (yet).

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

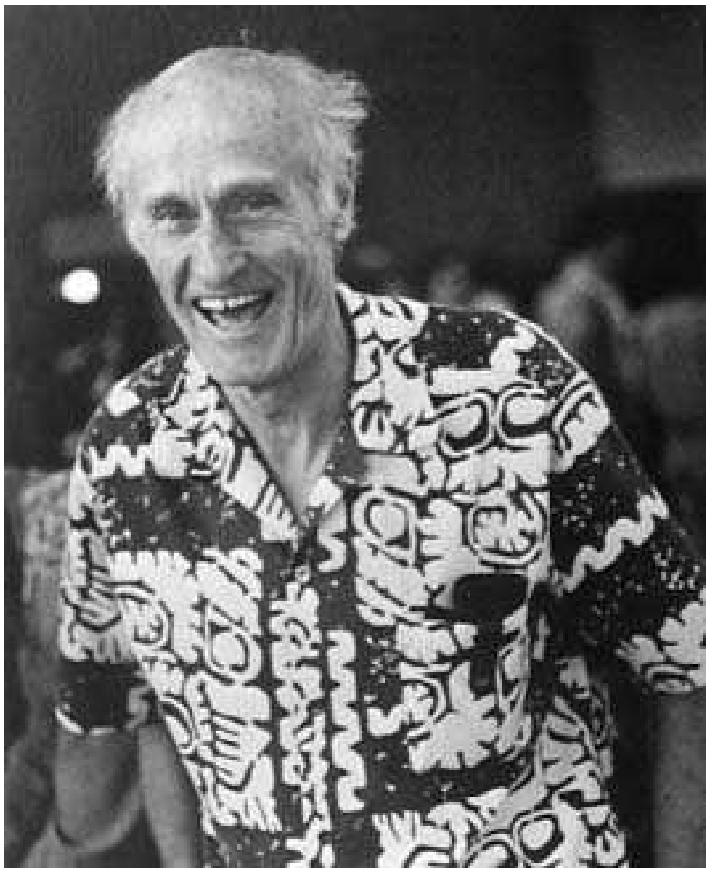
- The thesis may sound merely definitional at first.
- It can also be interpreted as[a]

  > 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] Smith (1998).

Alonso Church (1903–1995)

Stephen Kleene (1909–1994)



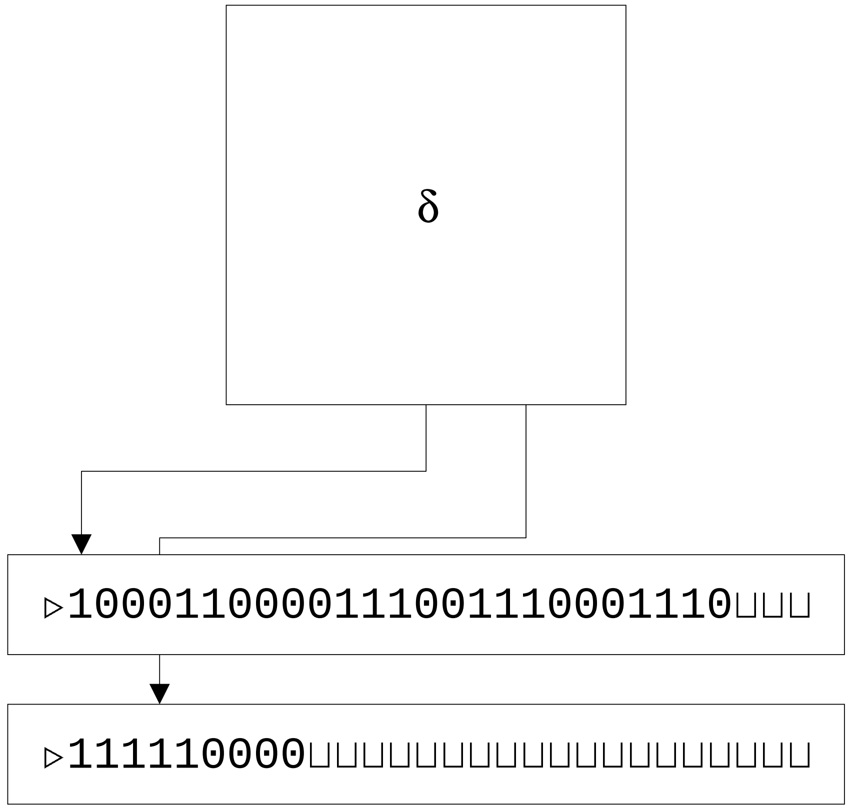
Extended Church's Thesis

- All "reasonably succinct encodings" of problems are *polynomially related.*
  - 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 vs. 111111111.
- All numbers for TMs will be binary from now on.

Turing Machines with Multiple Strings

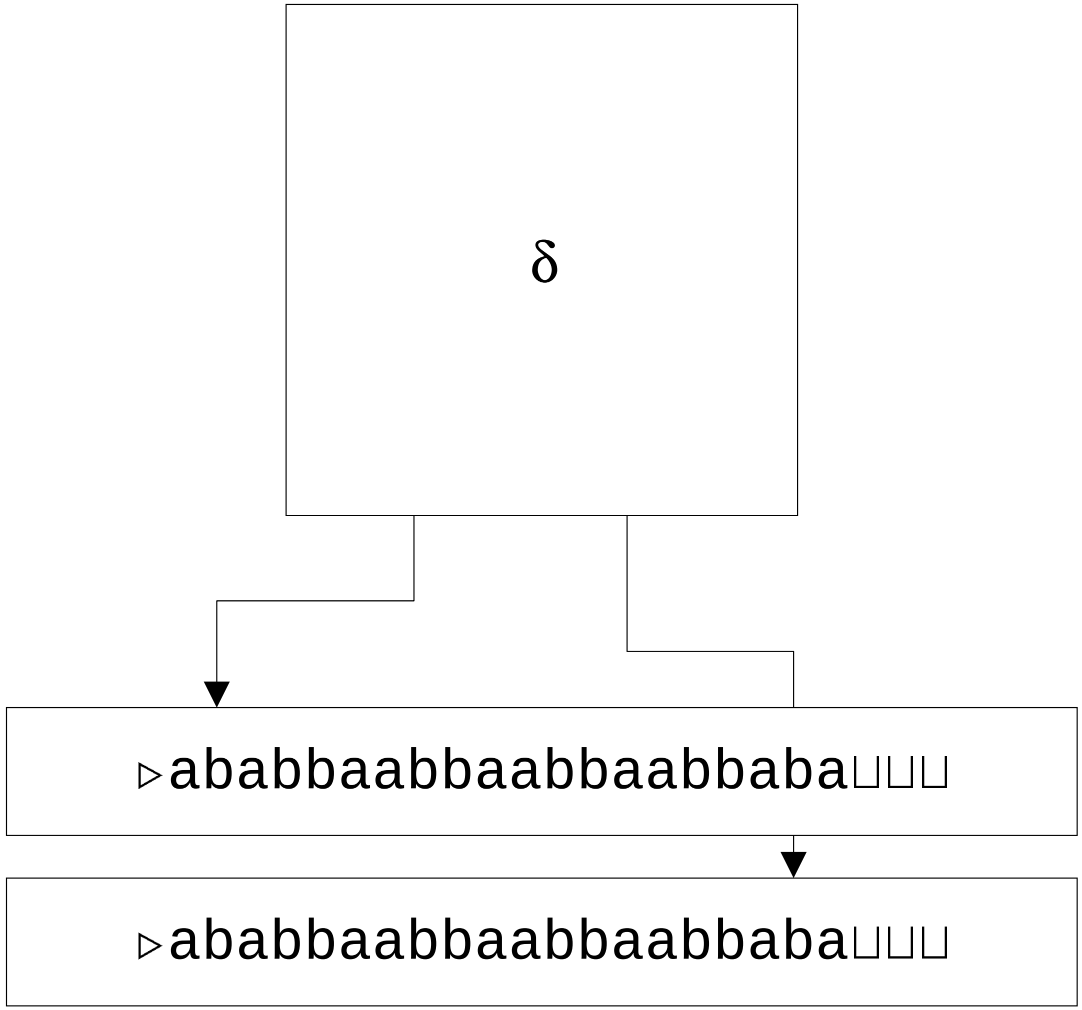
- A k-string Turing machine (TM) is a quadruple $M = (K, \Sigma, \delta, s)$.
- K, Σ, s are as before.
- $\delta : K \times \Sigma^k \rightarrow (K \cup \{h, \text{"yes"}, \text{"no"}\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k$.
- All strings start with a $\triangleright$.
- The first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

Configurations and Yielding

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

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

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

- The k-string TM's initial configuration is

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

Time Complexity

- The multistring TM is the basis of our notion of the time expended by TM computations.
- 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 ∞.
- 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 $f(\mid x \mid)$.
  - $\mid x \mid$ is the length of string x.
- Function $f(n)$ is a **time bound** for M.

Time Complexity Classes[a]

- Suppose language $L \subseteq (\Sigma - \{\sqcup\})^*$ 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).

The Simulation Technique

Theorem 3 *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.
- Represent configuration $(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k)$ of M by configuration

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

  of M'.
  - $\triangleleft$ is a special delimiter.
  - w_i' is w_i with the first[a] and last symbols "primed."

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

The Proof (continued)

- The "priming" is to ensure that M' knows which symbol is under the cursor for each simulated string.[a]
- The initial configuration of M' is

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

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

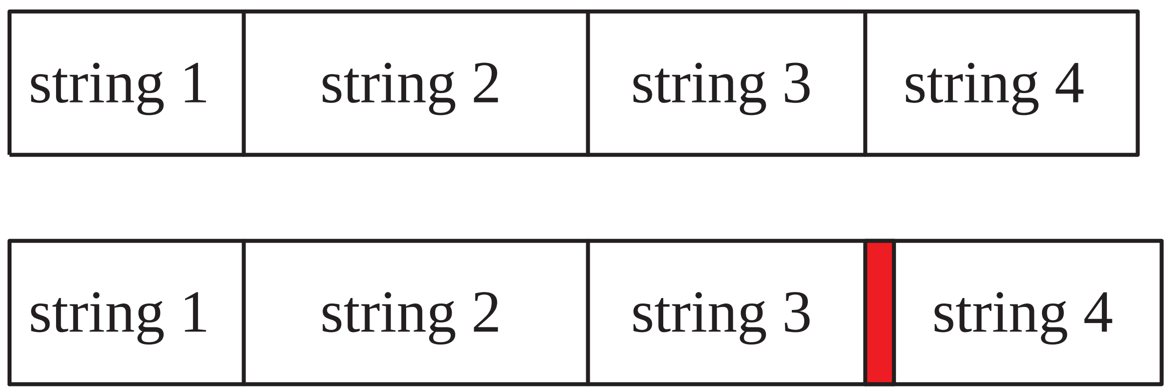
The Proof (continued)

- 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.
  - The linear-time algorithm on p. 31 can be used for each such string.
- The simulation continues until M halts.
- M' erases all strings of M except the last one.
- 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 $O(kf(|\,x\,|))$.

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


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

The Proof (concluded)

- Simulating each step of M takes, *per string of* M, $O(kf(|x|))$ steps.
  - $O(f(|x|))$ steps to collect information.
  - $O(kf(|x|))$ steps to write and, if needed, to lengthen the string.
- 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 4 *Let* $L \in \mathit{TIME}(f(n))$*. Then for any* $\epsilon > 0$*,* $L \in \mathit{TIME}(f'(n))$*, 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.
  - $m^k \cdot |\Sigma|^{3mk}$-fold increase to gain a speedup of $O(m)$.
- 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.
  - This justifies the asymptotic big-O notation.

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:

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

- Problems in P can be efficiently solved.

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 downplay the space requirement.
- 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`) on September 27, 2006.

Space Complexity (continued)

- 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 $f(n)$.
- SPACE$(f(n))$ is a set of languages.
  - PALINDROME $\in$ SPACE$(\log n)$: Keep 3 counters.
- As in the linear speedup theorem (Theorem 4), constant coefficients do not matter.

Nondeterminism[a]

- A **nondeterministic Turing machine** (**NTM**) is a quadruple $N = (K, \Sigma, \Delta, s)$.
- K, Σ, 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, there may be more than one next steps—or none at all.

[a] Rabin and Scott (1959).

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

Nondeterminism (concluded)

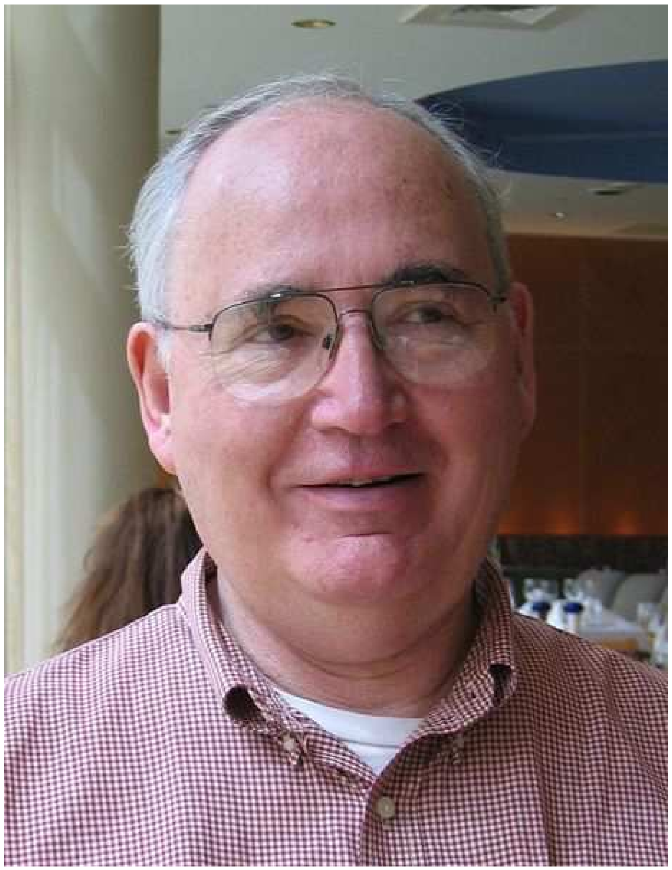
- Think of the program as lines of codes:

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

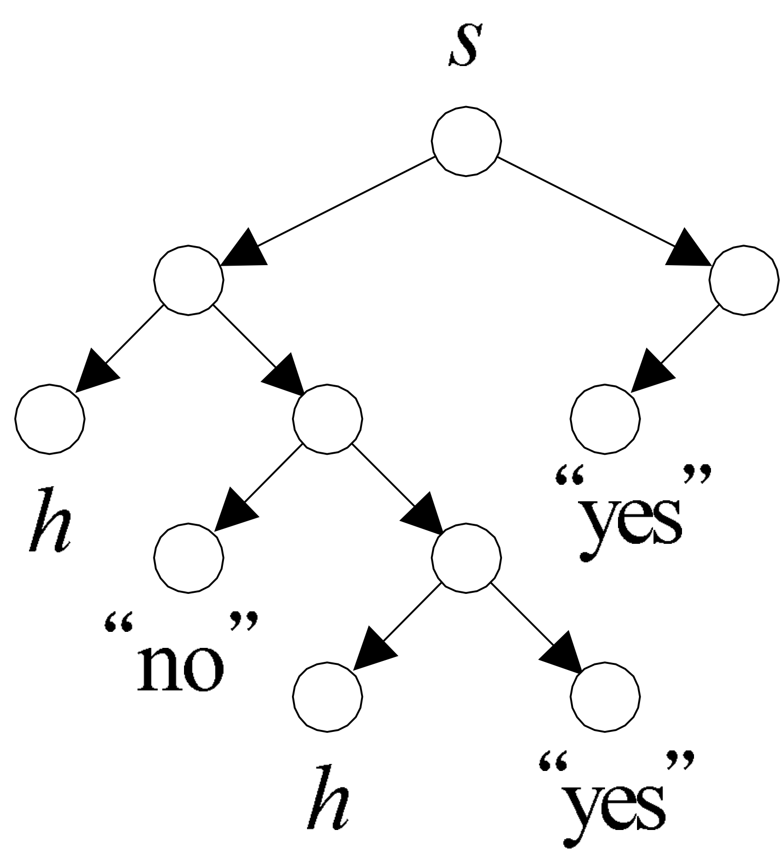
- A configuration yields another configuration in one step if there *exists* a rule in Δ that makes this happen.

Michael O. Rabin (1931–)

Dana Stewart Scott (1932–)



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."
  - 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.
- What is 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" may be a more proper term here.

An Example

- Let L be the set of logical conclusions of a set of axioms.
  - Predicates not in L may be false under the axioms.
  - They may also be independent of the axioms.
    * That is, they can be assumed true or false without contradicting the axioms.

An Example (concluded)

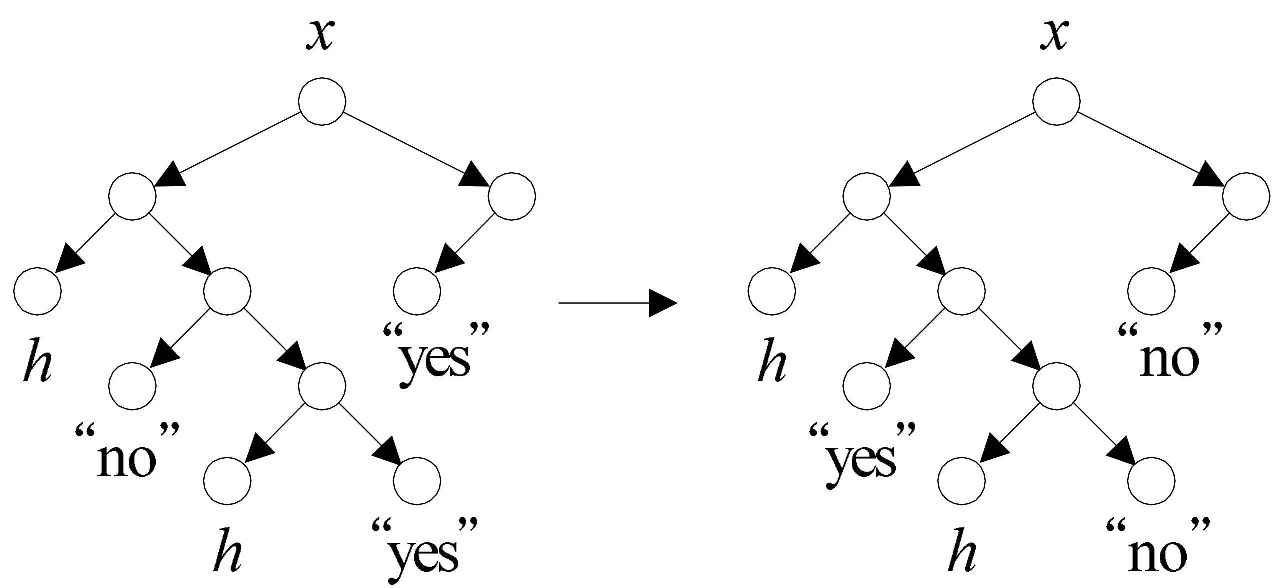
- Let ϕ be a predicate whose validity we would like to prove.
- Consider the nondeterministic algorithm:

  $b :=$ `true`;
  while the input predicate $\phi \neq b$ **do**
      Generate a logical conclusion of b by applying some of the axioms; {Nondeterministic choice.}
      Assign this conclusion to b;
  end while
  "yes";

- This algorithm decides L.

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 $\bar{L}$.
- 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).
  - When this happens, M and M' accept languages that are not complements of each other.


x
h
"no"
h
"yes"
"yes"
x
h
"yes"
h
"no"
"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.
  - Boolean satisfiability (SAT).
  - TSP (D).
- The most important open problem in computer science is whether $\mathrm{P} = \mathrm{NP}$.

Simulating Nondeterministic TMs

Surprisingly, nondeterminism does not add power to TMs.

Theorem 5 *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* know $f(n)$.
  - As N is time-bounded, the depth-first search will not run indefinitely.

The Proof (concluded)

- If some path leads to "yes," then M enters the "yes" state.
- If none of the paths leads to "yes," then M enters the "no" state.

Corollary 6 $\mathrm{NTIME}(f(n))) \subseteq \bigcup_{c>1} \mathrm{TIME}(c^{f(n)})$.

NTIME vs. TIME

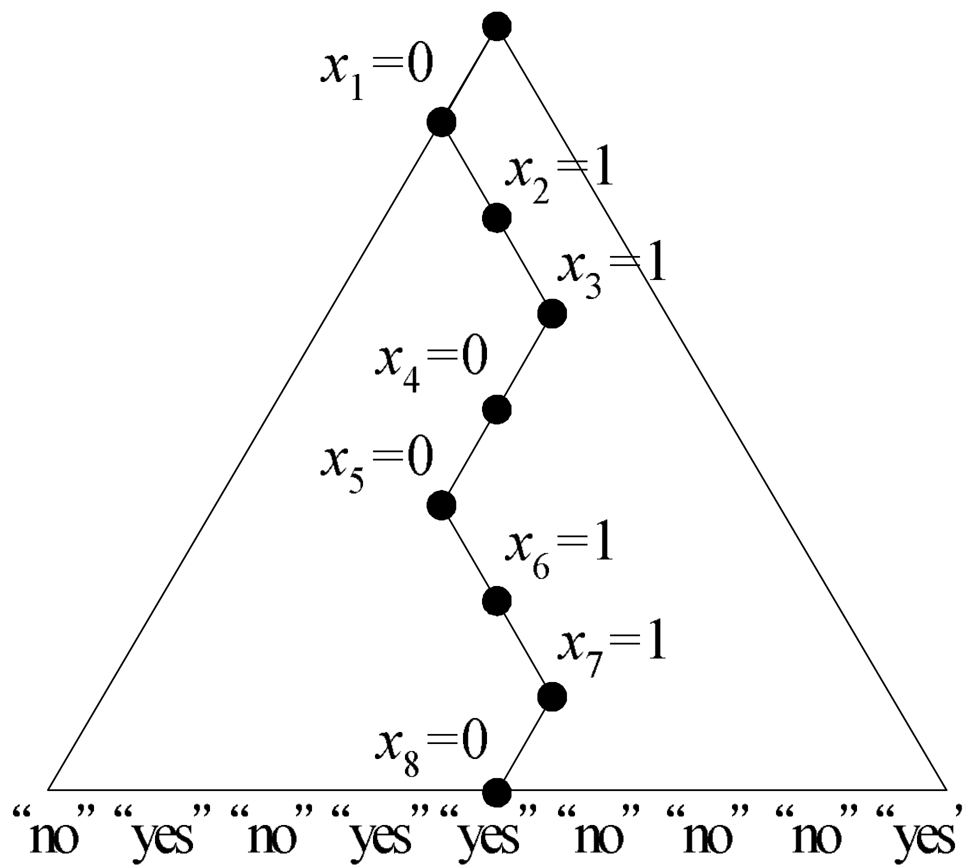
- Does converting an NTM into a TM require exploring all of the computation paths of the NTM as done in Theorem 5 (p. 92)?
- This is the most important question in theory with practical implications.

A Nondeterministic Algorithm for Satisfiability

ϕ is a boolean formula with n variables.

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

The Computation Tree for Satisfiability



Analysis

- The algorithm decides language $\{\phi : \phi \text{ is satisfiable}\}$.
  - The computation tree is a complete binary tree of depth n.
  - Every computation path corresponds to a particular truth assignment out of 2^n.
  - ϕ is satisfiable if and only if there is a computation path (truth assignment) that results in "yes."
- General paradigm: Guess a "proof" and then verify it.

The Traveling Salesman Problem

- We are given n cities $1, 2, \ldots, n$ and integer distances 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.
- Both problems are extremely important but equally hard (p. 336 and p. 418).

A Nondeterministic Algorithm for TSP (D)

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