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. 134).
Turing-Computable Functions

• Let $f : (\Sigma - \{\bot\})^* \rightarrow \Sigma^*$.
  
  – Optimization problems, root finding problems, etc.

• Let $M$ be a TM with alphabet $\Sigma$.

• $M$ computes $f$ if for any string $x \in (\Sigma - \{\bot\})^*$,
  
  $M(x) = f(x)$.

• We call $f$ a recursive function\(^a\) if such an $M$ exists.

\(^a\)Kurt Gödel (1931, 1934).
Kurt Gödel\textsuperscript{a} (1906–1978)

Quine (1978), “this theorem [...] sealed his immortality.”

\textsuperscript{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\textsuperscript{a}

- All “reasonably succinct encodings” of problems are \textit{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 \textit{unary} representation of numbers is not succinct.
  - The \textit{binary} representation of numbers is succinct.
    * 1001 vs. 111111111.

- All numbers for TMs will be binary from now on.

\textsuperscript{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.
  – If the input uses unary representation, the same algorithm runs in linear time!

• 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.
- $\delta : K \times \Sigma^k \rightarrow (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times (\Sigma \times \{←, →, −\})^k$.
- 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

δ

1001100011100111001110

111110000

111110000

©2012 Prof. Yuh-Dauh Lyuu, National Taiwan University
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.
PALINDROME Revisited (concluded)

- The running times of a 2-string TM and a single-string TM are quadratically related.
- 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

\[
\left( s, \triangleright, x, \triangleright, \epsilon, \triangleright, \epsilon, \ldots, \triangleright, \epsilon \right).
\]
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 $M(x) = \uparrow$, then the time required by $M$ on $x$ is $\infty$.

• Machine $M$ operates within time $f(n)$ for $f : \mathbb{N} \rightarrow \mathbb{N}$ 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

- Suppose language $L \subseteq (\Sigma - \{\boxslash\})^*$ 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.
  - $\text{PALINDROME}$ is in $\text{TIME}(f(n))$, where $f(n) = O(n)$.

---

*Hartmanis and Stearns (1965); Hartmanis, Lewis, and Stearns (1965).*
Juris Hartmanis\textsuperscript{a} (1928–)

\textsuperscript{a}Turing Award (1993).
Richard Edwin Stearns\textsuperscript{a} (1936–)

\textsuperscript{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 $x$. 

©2012 Prof. Yuh-Dauh Lyuu, National Taiwan University
The Proof

- 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 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 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 $M$.\textsuperscript{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 \textit{between} the simulated strings of $M$.\textsuperscript{b}

\textsuperscript{a}Added because of comments made by Mr. Che-Wei Chang (R95922093) on September 27, 2006.
\textsuperscript{b}Thanks to a lively discussion on September 22, 2009.
The Proof (continued)

- The initial configuration of $M'$ is
  
  \[
  (s, \triangleright\triangleright'' x \triangleleft\triangleleft'' \triangleleft \cdots \triangleleft'' \triangleleft \triangleleft).
  \]

  - $\triangleright$ is double-primed because it is the beginning and the ending symbol as the cursor is reading it.\(^{\text{a}}\)

\(^{\text{a}}\)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.
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\textsuperscript{a}

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

\textsuperscript{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 for 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.
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 $P$:
  \[ P = \bigcup_{k>0} \text{TIME}(n^k). \]

- $P$ 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-$\bot$ is never written over by $\bot$.\(^a\)
  - 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 $f : \mathbb{N} \rightarrow \mathbb{N}$ 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.
  
  – \( \text{PALINDROME} \in \text{SPACE}(\log n) \).\(^a\)

• As in the linear speedup theorem (p. 75), 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.

- \(\Delta \subseteq K \times \Sigma \times (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{←, →, −\}\) is a relation, not a function.\(^b\)
  - For each state-symbol combination, there may be multiple valid next steps—or none at all.
  - 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:

\[(q_1, \sigma_1, p_1, \rho_1, D_1) \in \Delta,\]
\[(q_2, \sigma_2, p_2, \rho_2, D_2) \in \Delta,\]
\[\vdots\]
\[(q_n, \sigma_n, p_n, \rho_n, D_n) \in \Delta.\]

— In the deterministic case (p. 22), we wrote

\[\delta(q_i, \sigma_i) = (p_i, \rho_i, D_i).\]

• A configuration yields another configuration in one step if there exists a rule in \(\Delta\) that makes this happen.
Michael O. Rabin\textsuperscript{a} (1931–)

\textsuperscript{a}Turing Award (1976).
Dana Stewart Scott\textsuperscript{a} (1932–)

\textsuperscript{a}Turing Award (1976).
Computation Tree and Computation Path

\[ s \]

\[ h \]

\[ \text{“no”} \]

\[ h \]

\[ \text{“yes”} \]

\[ \text{“yes”} \]
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.
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 \( \phi \) be a predicate whose validity we would like to prove.

• Consider the nondeterministic algorithm:
  1: \( b := \text{true}; \)
  2: while the input predicate \( \phi \neq b \) do
  3: Generate a logical conclusion of \( b \) by applying one of the axioms; \{Nondeterministic choice.\}
  4: Assign this conclusion to \( b \);
  5: end while
  6: “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 $\overline{L}$.

• But if $M$ is an NTM, then $M'$ may not decide $\overline{L}$.
  – 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.
Time Complexity under Nondeterminism

• Nondeterministic machine $N$ decides $L$ in time $f(n)$, where $f : \mathbb{N} \rightarrow \mathbb{N}$, 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

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

• Define

\[ NP = \bigcup_{k>0} \text{NTIME}(n^k). \]

• Clearly \( P \subseteq NP \).

• Think of \( NP \) as efficiently \textit{verifiable} problems (see p. 293).
  
  - Boolean satisfiability (p. 100 and p. 170).

• The most important open problem in computer science is whether \( P = NP \).