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.
- $\delta : K \times \Sigma^k \rightarrow (K \cup \{h, "yes", "no"\}) \times (\Sigma \times \{←, →, –\})^k$.
- 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

\[ \delta \]

\[ \Rightarrow 1000110000111001110001110 \]

\[ \Rightarrow 111110000 \]

\[ \Rightarrow 111110000 \]
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.

©2022 Prof. Yuh-Dauh Lyuu, National Taiwan University
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 the extended Church’s thesis.\(^a\)
  - “Reasonable” models are related polynomially in running times.

\(^a\)Recall p. 68.
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

\[
(s, \triangleright, x, \triangleright, \epsilon, \triangleright, \epsilon, \ldots, \triangleright, \epsilon).
\]
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 $\infty$. 
Time Complexity (concluded)

- 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\textsuperscript{a}

- Suppose language $L \subseteq (\Sigma - \{\Box\})^*$ 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.
  - \textsc{Palindrome} is in $\text{TIME}(f(n))$, where $f(n) = O(n)$.
- Trivially, $\text{TIME}(f(n)) \subseteq \text{TIME}(g(n))$ if $f(n) \leq g(n)$ for all $n$.

\textsuperscript{a}Rabin (1963); Hartmanis & Stearns (1965); Hartmanis, Lewis, & Stearns (1965).
Michael O. Rabin\textsuperscript{a} (1931–)

\textsuperscript{a}Turing Award (1976).
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 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 $M$. 
The Proof

- Represent configuration \((q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k)\) of \(M\) by this configuration 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.\(^b\)

\(^a\)The first symbol is of course \(\triangleright\).
\(^b\)An alternative is to use \((q, \triangleright w'_1| u_1 \triangleleft w'_2| u_2 \triangleleft \cdots \triangleleft w'_k| u_k \triangleleft \triangleleft)\) by priming only \(\triangleright\) in \(w_i\), where “|” is a new symbol.
The Proof (continued)

- The first symbol of $w'_i$ is the primed version of $\triangleright$: $\triangleright'$.
  - Cursors are not allowed to move to the left of $\triangleright$.\textsuperscript{a}
  - So the cursor of $M'$ can move \textit{between} the simulated strings of $M$.\textsuperscript{b}

- The “priming” of the last symbol of each $w_i$ ensures that $M'$ knows which symbol is under each cursor of $M$.\textsuperscript{c}

\textsuperscript{a} Recall p. 24.
\textsuperscript{b} Thanks to a lively discussion on September 22, 2009.
\textsuperscript{c} Added because of comments made by Mr. Che-Wei Chang (R95922093) on September 27, 2006.
The Proof (continued)

• The initial configuration of $M'$ is

\[(s, \triangleright \triangleright '' x \triangleleft \triangleright '' \triangleleft \cdots \triangleright '' \triangleleft \downarrow)\).

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

- Again, think of it as a new symbol.

\(^a\) 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.\(^a\)
     - 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 $M$.

\(^a\)Recall the TM program on p. 36.
The Proof (continued)

- It is possible that some strings of $M$ need to be lengthened (see next page).
  - The linear-time algorithm on p. 39 can be used for each such string.

- The simulation continues until $M$ halts.

- $M'$ then erases all strings of $M$ except the last one.\(^a\)

\(^a\)Whatever remains on the tape of $M'$ before the first $\square$ is considered output by our convention. So $\triangleright$'s and $\triangleright''$'s must be removed.
If we interleave the strings, the simulation may be easier. Contributed by Mr. Kai-Yuan Hou (B99201038, R03922014) on September 22, 2015. This is similar to constructing a single-string multi-track TM in, e.g., Hopcroft & Ullman (1969). Or one may do the insertion starting from the last string by memorizing what needs to be inserted for each string. Contributed by Mr. Hsi-Kang Hsu (R10922128) on September 30, 2021.
The Proof (continued)

- Since $M$ halts within time $f(|x|)$, none of its strings ever becomes longer than $f(|x|)$.

- 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)

• There are \( k \) strings.

• So \( M' \) takes \( O(k^2 f(|x|)) \) steps to simulate each step of \( M \).

• As there are \( f(|x|) \) steps of \( M \) to simulate, \( M' \) operates within time \( O(k^2 f(|x|)^2) \).

\[ \text{a} \]

\( ^{a} \)Is the time reduced to \( O(kf(|x|)^2) \) if the interleaving data structure is adopted?
Simulation with Two-String TMs

We can do better with two-string simulating TMs.

**Theorem 4** Given any $k$-string $M$ operating within time $f(n)$, $k > 2$, there exists a two-string $M'$ operating within time $O(f(n) \log f(n))$ such that $M(x) = M'(x)$ for any input $x$. 
Linear Speedup\textsuperscript{a}

Theorem 5  Let \( L \in \text{TIME}(f(n)) \). Then for any \( \epsilon > 0 \),
\( L \in \text{TIME}(f'(n)) \), where \( f'(n) \overset{\Delta}{=} \epsilon f(n) + n + 2 \).

See Theorem 2.2 of the textbook for a proof.

\textsuperscript{a}Hartmanis & Stearns (1965).
Proof Ideas

• Take the TM program on p. 36.

• It accepts if and only if the input contains two consecutive 1’s.

• Assume $M = (K, \Sigma, \delta, s)$, where
  
  $K = \{s', s_{00}, s_{01}, s_{10}, s_{11}, \ldots, "yes", "no" \}$,
  
  $\Sigma = \{0, 1, (00), (01), (10), (11), (0\sqcup), (1\sqcup), \sqcup, \triangleright\}$. 
Proof Ideas (continued)

- First convert the input into 2-tuples onto the second string.

\[
\begin{array}{c}
11 \\
\hline
\end{array}
\quad
\begin{array}{c}
6 \\
\hline
\end{array}
\]

- So 10011001110 becomes (10)(01)(10)(01)(11)(0\text{□}).

- The length is therefore about halved.

- The transition table below covers only the second string for brevity.

- It presents only the key lines of code.
<table>
<thead>
<tr>
<th>$p \in K$</th>
<th>$\sigma \in \Sigma$</th>
<th>$\delta(p, \sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$s'$</td>
<td>(00)</td>
<td>$(s', (00), \rightarrow)$</td>
</tr>
<tr>
<td>$s'$</td>
<td>(01)</td>
<td>$(s_{01}, (01), \rightarrow)$</td>
</tr>
<tr>
<td>$s'$</td>
<td>(10)</td>
<td>$(s', (10), \rightarrow)$</td>
</tr>
<tr>
<td>$s'$</td>
<td>(11)</td>
<td>(“yes”, (11), $-$)</td>
</tr>
<tr>
<td>$s'$</td>
<td>(0$\sqcup$)</td>
<td>(“no”, (0$\sqcup$), $-$)</td>
</tr>
<tr>
<td>$s'$</td>
<td>(1$\sqcup$)</td>
<td>(“no”, (1$\sqcup$), $-$)</td>
</tr>
<tr>
<td>$s'$</td>
<td>$\sqcup$</td>
<td>(“no”, $\sqcup$, $-$)</td>
</tr>
</tbody>
</table>
Proof Ideas (concluded)

<table>
<thead>
<tr>
<th>( s_{01} )</th>
<th>(10)</th>
<th>(&quot;yes&quot;, (10), −)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_{01} )</td>
<td>(11)</td>
<td>(&quot;yes&quot;, (11), −)</td>
</tr>
<tr>
<td>( s_{01} )</td>
<td>(01)</td>
<td>(( s_{01}, (01), \rightarrow ))</td>
</tr>
<tr>
<td>( s_{01} )</td>
<td>(00)</td>
<td>(( s', (00), \rightarrow ))</td>
</tr>
<tr>
<td>( s_{01} )</td>
<td>(0₁)</td>
<td>(&quot;no&quot;, (1₁), −)</td>
</tr>
<tr>
<td>( s_{01} )</td>
<td>(1₁)</td>
<td>(&quot;yes&quot;, (1₁), −)</td>
</tr>
<tr>
<td>( s_{01} )</td>
<td>⊒</td>
<td>(&quot;no&quot;, ⊒, −)</td>
</tr>
</tbody>
</table>

\(^a\)Corrected by Mr. Yu-Ming Lu (R06723032, D08922008) on September 30, 2021.
Implications of the Speedup Theorem

• State size can be traded for speed.\textsuperscript{a}

• If the running time is $cn$ with $c > 1$, then $c$ can be made arbitrarily close to 1.

• If the running time is superlinear, say $14n^2 + 31n$, then the constant in the leading term (14 in this example) can be made arbitrarily small.
  \begin{itemize}
    \item Arbitrary linear speedup can be achieved.\textsuperscript{b}
    \item This justifies the big-O notation in the analysis of algorithms.
  \end{itemize}

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

\textsuperscript{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 $n^k$.

• If $L \in \text{TIME}(n^k)$ for some $k \in \mathbb{N}$, it is a polynomially decidable language.
  - Clearly, $\text{TIME}(n^k) \subseteq \text{TIME}(n^{k+1})$.

• The union of all polynomially decidable languages is denoted by $P$:

\[
P \overset{\Delta}{=} \bigcup_{k>0} \text{TIME}(n^k).
\]

• $P$ contains problems that can be efficiently solved.

\[^{a}\text{Cobham (1964).}\]
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
  \[\sum_{i=1}^{k} |w_iu_i|\]  

\(^a\)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.$^a$
  – The cursor on the last string never moves to the left.
    
    * The output string is essentially write-only.
  – The cursor of the input string does not go beyond the first $\sqcup$.

\footnote{$^a$Called an \textbf{off-line TM} in Hartmanis, Lewis, & Stearns (1965).}
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 linear speedup theorem similar to the one on p. 97 exists, so constant coefficients do not matter.

\(^{a}\)Keep 3 counters.
If she can hesitate as to “Yes,” she ought to say “No” directly.
— Jane Austen (1775–1817), Emma (1815)
Nondeterminism\(^a\)

- A nondeterministic Turing machine (\(\text{NTM}\)) 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.
  - But only one will be taken.

\(^a\)Rabin & Scott (1959).
\(^b\)Corrected by Mr. Jung-Ying Chen (D95723006) on September 23, 2008.
Nondeterminism (continued)

• As before, a program contains lines of code:

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

• But we cannot write

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

as in the deterministic case (p. 25) anymore.
Nondeterminism (concluded)

- A configuration yields another configuration in one step if there \textit{exists} a rule in \(\Delta\) that makes this happen.
- There remains only one thread of computation.\(^a\)
  - Nondeterminism is \textit{not} parallelism, multiprocessing, multithreading, or quantum computation.

\(^a\)Thanks to a lively discussion on September 22, 2015.
Dana Stewart Scott\textsuperscript{a} (1932–)

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

\[ s \]

\[ h \]

“no”

\[ h \]

“yes”

\[ “yes” \]
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 $N(x) = \text{“yes”}$ for some computation path.
  - If $x \not\in L$, then $N(x) \neq \text{“yes”}$ for all computation paths.
Decidability under Nondeterminism (continued)

- It is not required that the deciding 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.

- Note that determinism is a special case of nondeterminism.

---

aUnlike the deterministic case (p. 53). So “accepts” may be a more proper term. Some books use “decides” only when the NTM always halts.
Decidability under Nondeterminism (concluded)

- For example, suppose $L$ is the set of primes.\(^a\)
- Then we have the primality testing problem.
- An NTM $N$ decides $L$ if:
  - If $x$ is a prime, then $N(x) =$ “yes” for some computation path.
  - If $x$ is not a prime, then $N(x) \neq “yes”$ for all computation paths.

\(^a\)Contributed by Mr. Yu-Ming Lu (R06723032, D08922008) on March 7, 2019.
Complementing a TM’s Halting States

- Let $M$ decide $L$, and $M'$ be $M$ after “yes” ↔ “no”.
- If $M$ is deterministic, then $M'$ decides $\overline{L}$.\(^a\)
  - So $M$ and $M'$ decide languages that complement each other.
- But if $M$ is an NTM, then $M'$ may not decide $\overline{L}$.
  - It is possible that $M$ and $M'$ accept the same input $x$ (see next page).
  - So $M$ and $M'$ may accept languages that are *not* even disjoint.

\(^a\)By the definition on p. 53, $M$ must halt on all inputs.
A Nondeterministic Algorithm for Satisfiability

\( \phi \) is a boolean formula with \( n \) variables.

1: \textbf{for} \( i = 1, 2, \ldots, n \) \textbf{do}
2: \hspace{1em} Guess \( x_i \in \{0, 1\} \); \{Nondeterministic choices.\}
3: \hspace{4em} \textbf{end for}
4: \hspace{1em} \{Verification:\}
5: \hspace{4em} \textbf{if} \( \phi(x_1, x_2, \ldots, x_n) = 1 \) \textbf{then}
6: \hspace{7em} “yes”;
7: \hspace{4em} \textbf{else}
8: \hspace{7em} “no”;
9: \hspace{4em} \textbf{end if}
Computation Tree for Satisfiability

\[ x_1 = 0 \]
\[ x_2 = 1 \]
\[ x_3 = 1 \]
\[ x_4 = 0 \]
\[ x_5 = 0 \]
\[ x_6 = 1 \]
\[ x_7 = 1 \]
\[ x_8 = 0 \]
Analysis

- Recall that $\phi$ is satisfiable if and only if there is a truth assignment that satisfies $\phi$.

- The computation tree is a complete binary tree of depth $n$.

- Every computation path corresponds to a particular truth assignment\(^a\) out of $2^n$.

\(^a\)Equivalently, a sequence of nondeterministic choices.
Analysis (concluded)

• The algorithm decides language

\( \{ \phi : \phi \text{ is satisfiable} \} \).

  – Suppose \( \phi \) is satisfiable.
     * 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” 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.$^a$

• The decision version TSP (D) asks if there is a tour with a total distance at most $B$, where $B$ is an input.$^b$

---

$^a$Each city is visited exactly once.

$^b$Both problems are extremely important. They are equally hard (p. 421 and p. 524).
A Shortest Tour
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. {Verification:}
5. if $x_1, x_2, \ldots, x_n$ are distinct and $\sum_{i=1}^{n-1} d_{x_i, x_{i+1}} \leq B$ then
6. "yes";
7. else
8. "no";
9. 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 for that tour.\(^a\)
  – And it leads to “yes.”

• 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 merely requires that such a computation path (i.e., a sequence of nondeterministic choices) exists.
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

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

- $\text{NTIME}(f(n))$ is a complexity class.
NP ("Nondeterministic Polynomial")

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

- Clearly \( P \subseteq NP \).

- Think of NP as efficiently *verifiable* problems (see p. 349).
  - Boolean satisfiability (p. 120 and p. 205), e.g.

- The most important open problem in computer science is whether \( P = NP \).
Remarks on the $P \equiv NP$ Open Problem

- Many practical applications depend on answers to the $P \equiv NP$ question.
- Verification of password should be 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 $P$).
- It took 63 years to settle the Continuum Hypothesis; how long will it take for this one?

---

Contributed by Mr. Kuan-Lin Huang (B96902079, R00922018) on September 27, 2011.
Simulating Nondeterministic TMs

Nondeterminism does not add power to TMs.\(^a\)

**Theorem 6** 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 \(x\), \(M\) explores the computation tree of \(N(x)\) using depth-first search.
  - \(M\) does not need to know \(f(n)\).
  - As \(N\) is time-bounded, the depth-first search will halt.\(^b\)

\(^a\)Like finite-state automata, but unlike pushdown automata.
\(^b\)If there is no time bound, breadth-first search is safer.
The Proof (concluded)

- If any path leads to “yes,” then $M$ immediately enters the “yes” state.

- If none of the paths lead 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 7  \(\text{NTIME}(f(n)) \subseteq \bigcup_{c>1} \text{TIME}(c^{f(n)})\).\(^a\)

\(^{a}\text{Mr. Kai-Yuan Hou (B99201038, R03922014) on October 6, 2015:}\)

\(\bigcup_{c>1} \text{TIME}(c^{f(n)}) \subseteq \text{NTIME}(f(n))\)?
NTIME vs. TIME

• Does converting an NTM into a TM require exploring all computation paths of the NTM in the worst case as done in Theorem 6 (p. 132)?

• This is a key question in theory with important practical implications.
Nondeterministic Space Complexity Classes

• Let $L$ be a language.

• Then

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

if there is an NTM with input and output that decides $L$ and operates within space bound $f(n)$.

• $\text{NSPACE}(f(n))$ is a set of languages.

• As in the linear speedup theorem, a constant coefficients do not matter.

\(^{a}\text{Theorem 5 (p. 97).}\)
Graph Reachability

• Let $G(V, E)$ be a directed graph (digraph).

• REACHABILITY asks, given nodes $a$ and $b$, does $G$ contain a path from $a$ to $b$?

• Can be easily solved in polynomial time by breadth-first search.

• How about its nondeterministic space complexity?
The First Try: \textsc{NSPACE}(n \log n)

1: Determine the number of nodes \( m \); \{Note \( m \leq n \).\}
2: \( x_1 := a \); \{Assume \( a \neq b \).\}
3: \textbf{for} \( i = 2, 3, \ldots, m \) \textbf{do}
4: \hspace{1em} Guess \( x_i \in \{ v_1, v_2, \ldots, v_m \} \); \{The \( i \)th node.\}
5: \textbf{end for}
6: \textbf{for} \( i = 2, 3, \ldots, m \) \textbf{do}
7: \hspace{1em} \textbf{if} \ (x_{i-1}, x_i) \notin E \ \textbf{then}
8: \hspace{2em} “no”;
9: \hspace{1em} \textbf{end if}
10: \hspace{1em} \textbf{if} \ \( x_i = b \) \ \textbf{then}
11: \hspace{2em} “yes”;
12: \hspace{1em} \textbf{end if}
13: \textbf{end for}
14: “no”;

\( \copyright \) 2022 Prof. Yuh-Dauh Lyuu, National Taiwan University
In Fact, \textsc{reachability} $\in \text{NSPACE}(\log n)$

1: Determine the number of nodes $m$; \{Note $m \leq n$.\}

2: $x := a$;

3: for $i = 2, 3, \ldots, m$ do

4: Guess $y \in \{v_1, v_2, \ldots, v_m\}$; \{The next node.\}

5: if $(x, y) \notin E$ then

6: \hspace{1em} “no”;

7: end if

8: if $y = b$ then

9: \hspace{1em} “yes”;

10: end if

11: $x := y$; \{Recycle the space.\}

12: end for

13: “no”;
Space Analysis

• Variables $m$, $i$, $x$, and $y$ each require $O(\log n)$ bits.

• Testing $(x, y) \in E$ is accomplished by consulting the input string with counters of $O(\log n)$ bits long.

• Hence

$$\text{REACHABILITY} \in \text{NSPACE}(\log n).$$

  – REACHABILITY with more than one terminal node also has the same complexity.

  – In fact, REACHABILITY for undirected graphs is in $\text{SPACE}(\log n)$.\(^a\)

• It is well-known that REACHABILITY $\in \text{P}$.\(^b\)

\(^a\)Reingold (2004).
\(^b\)See, e.g., p. 250.
Undecidability
He [Turing] invented
the idea of software, essentially[.] It’s software that’s really
the important invention.
— Freeman Dyson (2015)
Universal Turing Machine\textsuperscript{a}

- A universal Turing machine $U$ interprets the input as the description of a TM $M$ concatenated with the description of an input to that machine, $x$.\textsuperscript{b}
  - Both $M$ and $x$ are over the alphabet of $U$.
- $U$ simulates $M$ on $x$ so that

$$U(M; x) = M(x).$$

- $U$ is like a modern computer, which executes any valid machine code, or a Java virtual machine, which executes any valid bytecode.

\textsuperscript{a}Turing (1936) calls it “universal computing machine.”
\textsuperscript{b}See pp. 57–58 of the textbook.
The Halting Problem

• Undecidable problems are problems that have no algorithms.
  – Equivalently, they are languages that are not recursive.

• We now define a concrete undecidable problem, the halting problem:

\[ H \triangleq \{ M; x : M(x) \neq \uparrow \}. \]

  – Does \( M \) halt on input \( x \)?

• \( H \) is called the halting set.
$H$ is recursively enumerable

- Use the universal TM $U$ to simulate $M$ on $x$.
- When $M$ is about to halt, $U$ enters a “yes” state.
- If $M(x)$ diverges, so does $U$.
- This TM accepts $H$. 
$H$ Is Not Recursive$^a$

- Suppose $H$ is recursive.
- Then there is a TM $M_H$ that decides $H$.
- Consider the program $D(M)$ that calls $M_H$:
  1. `if $M_H(M;M) = \text{“yes” then}
  2. ↗; \{\text{Inserting an infinite loop here.}\}
  3. else
  4. “yes”;
  5. end if

$^a$Turing (1936).
$H$ Is Not Recursive (concluded)

- Consider $D(D)$:
  - $D(D) \uparrow \Rightarrow M_H(D; D) = "yes" \Rightarrow D; D \in H \Rightarrow D(D) \neq \uparrow$, a contradiction.
  - $D(D) = "yes" \Rightarrow M_H(D; D) = "no" \Rightarrow D; D \notin H \Rightarrow D(D) = \uparrow$, another contradiction.
Comments

• Two levels of interpretations of $M$:\(^a\)
  - A sequence of 0s and 1s (data).
  - An encoding of instructions (programs).

• There are no paradoxes with $D(D)$.
  - Concepts should be familiar to computer scientists.
  - Feed a C compiler to a C compiler, a Lisp interpreter to a Lisp interpreter, a sorting program to a sorting program, etc.

\(^a\)Eckert & Mauchly (1943); von Neumann (1945); Turing (1946).
It seemed unworthy of a grown man to spend his time on such trivialities, but what was I to do? […]
The whole of the rest of my life might be consumed in looking at that blank sheet of paper.
Self-Loop Paradoxes

Russell’s Paradox (1901): Consider \( R = \{ A : A \notin A \} \).
- If \( R \in R \), then \( R \notin R \) by definition.
- If \( R \notin R \), then \( R \in R \) also by definition.
- In either case, we have a “contradiction.”

Liar’s Paradox: “This sentence is false.”

Plato (375 B.C.), The Republic: “master of himself.”

---

\( ^a \)E.g., Quine (1966), The Ways of Paradox and Other Essays and Hofstadter (1979), Gödel, Escher, Bach: An Eternal Golden Braid.

\( ^b \)Gottlob Frege (1848–1925) to Bertrand Russell in 1902, “Your discovery of the contradiction […] has shaken the basis on which I intended to build arithmetic.”
Self-Loop Paradoxes (continued)

Epimenides and Eubulides: The Cretan says, “All Cretans are liars.”

Psalms 116:11: “Everyone is a liar.”

Hypochondriac: a patient with imaginary symptoms and ailments.

Sharon Stone in The Specialist (1994): “I’m not a woman you can trust.”

Numbers 12:3: “Moses was the most humble person in all the world […]” (attributed to Moses).

---

\[a\] Also quoted in Titus 1:12.

\[b\] Like Gödel and the pianist Glenn Gould (1932–1982).
Self-Loop Paradoxes (continued)

A restaurant in Boston: No Name Restaurant (1917–2020).


The Egyptian Book of the Dead: “ye live in me and I would live in you.”\(^{a}\)

\(^{a}\)See also John 14:10 and 17:21.
Self-Loop Paradoxes (concluded)

Jerome K. Jerome (1887), *Three Men in a Boat*: “How could I wake you, when you didn’t wake me?”

Winston Churchill (January 23, 1948): “For my part, I consider that it will be found much better by all parties to leave the past to history, especially as I propose to write that history myself.”

Bertrand Russell\textsuperscript{a} (1872–1970)

Norbert Wiener (1953), “It is impossible to describe Bertrand Russell except by saying that he looks like the Mad Hatter.”

Karl Popper (1974), “perhaps the greatest philosopher since Kant.”

\textsuperscript{a}Nobel Prize in Literature (1950).
Reductions in Proving Undecidability

• Suppose we are asked to prove that $L$ is undecidable.
• Suppose $L'$ (such as $H$) is known to be undecidable.
• Find a computable transformation $R$ (called reduction\textsuperscript{a}) from $L'$ to $L$ such that\textsuperscript{b}

$$\forall x \{ x \in L' \text{ if and only if } R(x) \in L \}.$$ 

• Now we can answer “$x \in L'$?” for any $x$ by answering “$R(x) \in L$?” because it has the same answer.

\textsuperscript{a}Post (1944).
\textsuperscript{b}Contributed by Mr. Tai-Dai Chou (J93922005) on May 19, 2005.
Algorithm for $L'$

$x$ \rightarrow R \rightarrow R(x) \rightarrow \text{algorithm for } L \rightarrow \text{yes/no}
Reductions in Proving Undecidability (concluded)

• $L'$ is said to be reduced to $L$.\(^a\)
  
  – It is written as $L' \leq L$ or even $L' \leq_m L$ to emphasize that the transformation is many-one.

• If $L$ were decidable, “$R(x) \in L$?” becomes computable and we have an algorithm to decide $L'$, a contradiction!

• So $L$ must be undecidable.

Theorem 8 Suppose language $L_1$ can be reduced to language $L_2$. If $L_1$ is undecidable, then $L_2$ is undecidable.

\(^a\)Intuitively, $L$ can be used to solve $L'$.
Special Cases and Reduction

• Suppose $L_1$ can be reduced to $L_2$.

• As the reduction $R$ maps members of $L_1$ to a *subset* of $L_2$, we may say $L_1$ is a “special case” of $L_2$.

• That is one way to understand the use of the somewhat confusing term “reduction.”

---

*Because $R$ may not be onto.*

*Contributed by Ms. Mei-Chih Chang (D03922022) and Mr. Kai-Yuan Hou (B99201038, R03922014) on October 13, 2015.*
Subsets and Decidability

• Suppose $L_1$ is undecidable and $L_1 \subseteq L_2$.
• Is $L_2$ undecidable?\(^a\)
• It depends.
• When $L_2 = \Sigma^*$, $L_2$ is decidable: Just answer “yes.”
• If $L_2 - L_1$ is decidable, then $L_2$ is undecidable.
  – Clearly,

  \[ x \in L_1 \text{ if and only if } x \in L_2 \text{ and } x \notin L_2 - L_1. \]

  – Therefore, if $L_2$ were decidable, then $L_1$ would be.

\(^a\)Contributed by Ms. Mei-Chih Chang (D03922022) on October 13, 2015.
Subsets and Decidability (concluded)

- Suppose $L_2$ is decidable and $L_1 \subseteq L_2$.
- Is $L_1$ decidable?
- It depends again.
- When $L_1 = \emptyset$, $L_1$ is decidable: Just answer “no.”
- But if $L_2 = \Sigma^*$ and $L_1 = H$, then $L_1$ is undecidable.
The Universal Halting Problem

• The universal halting problem:

\[ H^* \triangleq \{ M : M \text{ halts on all inputs} \}. \]

• It is also called the totality problem.
$H^*$ Is Not Recursive$^a$

- We will reduce $H$ to $H^*$.
- Given the question “$M; x \in H?$”, construct the following machine (this is the reduction):$^b$
  \[
  M_x(y) \{ M(x) \}
  \]
- $M$ halts on $x$ if and only if $M_x$ halts on all inputs.
- In other words, $M; x \in H$ if and only if $M_x \in H^*$.
- So if $H^*$ were recursive (recall the box for $L$ on p. 155), $H$ would be recursive, a contradiction.

$^a$Kleene (1936).

$^b$Simplified by Mr. Chih-Hung Hsieh (D95922003) on October 5, 2006. $M_x$ ignores its input $y$; $x$ is part of $M_x$’s code but not $M_x$’s input.
More Undecidability

• \{ M; x : \text{there is a } y \text{ such that } M(x) = y \}.

• \{ M; x : 
  \text{the computation } M \text{ on input } x \text{ uses all states of } M \}.

• \{ M; x; y : M(x) = y \}.
Complements of Recursive Languages

The complement of $L$, denoted by $\bar{L}$, is the language $\Sigma^* - L$.

**Lemma 9** *If $L$ is recursive, then so is $\bar{L}$.***

- Let $L$ be decided by a deterministic $M$.
- Swap the “yes” state and the “no” state of $M$.
- The new machine decides $\bar{L}$.\(^a\)

\(^a\)Recall p. 118.
Recursive and Recursively Enumerable Languages

Lemma 10 (Kleene’s theorem; Post, 1944) $L$ is recursive if and only if both $L$ and $\overline{L}$ are recursively enumerable.

- Suppose both $L$ and $\overline{L}$ are recursively enumerable, accepted by $M$ and $\overline{M}$, respectively.
- Simulate $M$ and $\overline{M}$ in an interleaved fashion.
- If $M$ accepts, then halt on state “yes” because $x \in L$.
- If $\overline{M}$ accepts, then halt on state “no” because $x \notin L$.
- The other direction is trivial.

\[ ^a\text{Either } M \text{ or } \overline{M} \text{ (but not both) must accept the input and halt.} \]
A Useful Corollary and Its Consequences

Corollary 11 \( L \) is recursively enumerable but not recursive, then \( \overline{L} \) is not recursively enumerable.

- Suppose \( \overline{L} \) is recursively enumerable.
- Then both \( L \) and \( \overline{L} \) are recursively enumerable.
- By Lemma 10 (p. 164), \( L \) is recursive, a contradiction.

Corollary 12 \( \overline{H} \) is not recursively enumerable.\(^a\)

\(^a\)Recall that \( \overline{H} \triangleq \{ M; x : M(x) = \uparrow \} \).
R, RE, and coRE

RE: The set of all recursively enumerable languages.

coRE: The set of all languages whose complements are recursively enumerable.

R: The set of all recursive languages.

- Note that coRE is not RE.
  - coRE $\triangleq \{ L : \overline{L} \in \text{RE} \} = \{ \overline{L} : L \in \text{RE} \}$.
  - $\overline{\text{RE}} \triangleq \{ L : L \not\in \text{RE} \}$.
R, RE, and coRE (concluded)

- $R = RE \cap \text{coRE}$ (p. 164).

- There exist languages in RE but not in R and not in coRE.
  - Such as $\mathcal{H}$ (p. 144, p. 145, and p. 165).

- There are languages in coRE but not in RE.
  - Such as $\overline{\mathcal{H}}$ (p. 165).

- There are languages in neither RE nor coRE.
$H$ Is Complete for RE\textsuperscript{a}

- Let $L$ be any recursively enumerable language.
- Assume $M$ accepts $L$.
- Clearly, one can decide whether $x \in L$ by asking if $M : x \in H$.
- Hence all recursively enumerable languages are reducible to $H$!
- $H$ is said to be RE-complete.

\textsuperscript{a}Post (1944).
Notations

- The language *accepted* by TM $M$ is written as $L(M)$.
- If $M(x) = \uparrow$ for all $x$, then $L(M) = \emptyset$.
- If $M(x)$ is never “yes” nor $\uparrow$ (as required by the definition of acceptance), we also let $L(M) = \emptyset$. 
Nontrivial Properties of Sets in RE

- A property of the recursively enumerable languages can be defined by the set $C$ of all the recursively enumerable languages that satisfy it.
  - The property of finite recursively enumerable languages is
    \[ \{ L : L = L(M) \text{ for a TM } M, L \text{ is finite} \} \].
  - The property of recursiveness is
    \[ \{ L : L = L(M) \text{ for a TM } M, L \text{ is recursive} \} \].
Nontrivial Properties of Sets in RE (continued)

• A property is trivial if $C = \text{RE}$ or $C = \emptyset$.
  – Answer to a trivial property (about the language a TM accepts) is either always “yes” or always “no.”
  – It is either possessed by all recursively enumerable languages or by none.

• Here is a trivial property (always yes): Does the TM accept a recursively enumerable language?

• Here is a trivial property (always no): Does the TM accept a language that is finite and infinite?

\[ a \]

\[ a \text{Or, } L(M) \in \text{RE? Formally, } \{ L : L = L(M) \text{ for a TM } M, L \in \text{RE} \}. \]