The Traveling Salesman Problem

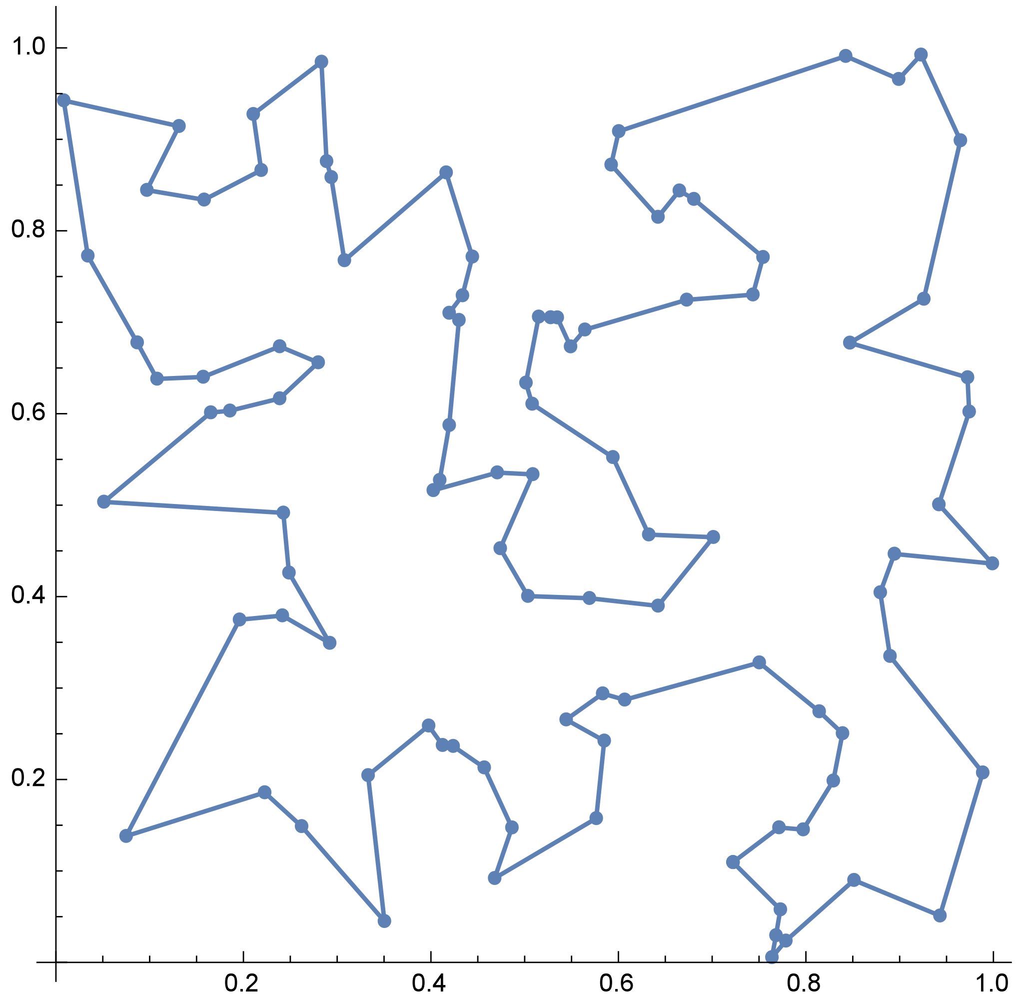
- We are given n cities $1, 2, \ldots, n$ and integer distance d_{ij} between any two cities i and j.
- Assume $d_{ij} = d_{ji}$ for convenience.
- The **traveling salesman problem** (TSP) asks for the total distance of the shortest tour of the cities.[a]
- The decision version TSP (D) asks if there 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. 399 and p. 501).

A Shortest Path



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

{Verification:}

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**

    "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.

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 just means such a computation path (i.e., a sequence of nondeterministic choices) 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 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 logicians 63 years to settle the Continuum Hypothesis; how long will it take for this one?

---

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

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]Theorem 5 (p. 92).

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: NSPACE$(n \log n)$

Determine the number of nodes m; {Note $m \leq n$.}
$x_1 := a$; {Assume $a \neq b$.}
for $i = 2, 3, \ldots, m$ **do**
  Guess $x_i \in \{ v_1, v_2, \ldots, v_m \}$; {The ith node.}
end for
for $i = 2, 3, \ldots, m$ **do**
  if $(x_{i-1}, x_i) \notin E$ **then**
    "no";
  end if
  if $x_i = b$ **then**
    "yes";
  end if
end for
"no";

In Fact, REACHABILITY $\in$ NSPACE$(\log n)$

```
Determine the number of nodes m; {Note m ≤ n.}
x := a;
for i = 2, 3, ..., m do
    Guess y ∈ { v_1, v_2, ..., v_m }; {The next node.}
    if (x, y) ∉ E then
        "no";
    end if
    if y = b then
        "yes";
    end if
    x := y;
end for
"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

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

  - REACHABILITY with more than one terminal node also has the same complexity.
  - In fact, REACHABILITY for *undirected* graphs is in SPACE$(\log n)$.[a]
- REACHABILITY $\in$ P (see, e.g., p. 235).

[a] Reingold (2005).

Undecidability

He [Turing] invented
the idea of software, essentially[.]
It's software that's really
the important invention.
— Freeman Dyson (2015)

Universal Turing Machine[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.[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.

---

[a]Turing (1936).

[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**:

$$H = \{ M; x : M(x) \neq \nearrow \}.$$

  - Does M halt on input x?

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:

```
if M_H(M; M) = "yes" then
    ↗; {Writing an infinite loop is easy.}
else
    "yes";
end if
```

[a]Turing (1936).

H Is Not Recursive (concluded)

- Consider $D(D)$:
  - $D(D) = \nearrow \Rightarrow M_H(D; D) =$ "yes" $\Rightarrow D; D \in H \Rightarrow D(D) \neq \nearrow$, a contradiction.
  - $D(D) =$ "yes" $\Rightarrow M_H(D; D) =$ "no" $\Rightarrow D; D \notin H \Rightarrow D(D) = \nearrow$, a 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? $[\cdots]$
The whole of the rest of my life might be
consumed in looking at
that blank sheet of paper.
— Bertrand Russell (1872–1970),
Autobiography, Vol. I (1967)

Self-Loop Paradoxes[a]

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."[b]

Eubulides: The Cretan says, "All Cretans are liars."

Liar's Paradox: "This sentence is false."

---

[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)

Hypochondriac: a patient with imaginary symptoms and ailments.[a]

Sharon Stone in *The Specialist* (1994): "I'm not a woman you can trust."

***Numbers* 12:3, Old Testament:** "Moses was the most humble person in all the world $[\cdots]$" (attributed to Moses).

---

[a]Like Gödel and Glenn Gould (1932–1982).

Self-Loop Paradoxes (continued)

***The Egyptian Book of the Dead*:** "ye live in me and I would live in you."

***John* 14:10, New Testament:** "Don't you believe that I am in the Father, and that the Father is in me?"

***John* 17:21, New Testament:** "just as you are in me and I am in you."

Self-Loop Paradoxes (concluded)

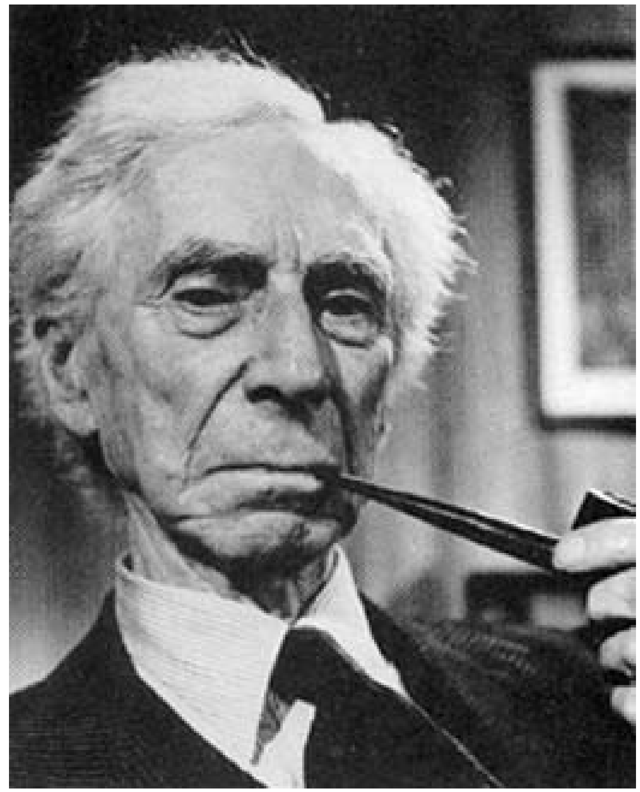
Jerome K. Jerome, *Three Men in a Boat* (1887):
"How could I wake you, when you didn't wake me?"

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."

Nicola Lacey, *A Life of H. L. A. Hart* (2004): "Top Secret [MI5] Documents: Burn before Reading!"

Bertrand Russell[a] (1872–1970)

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



[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**[a]) from L' to L such that[b]

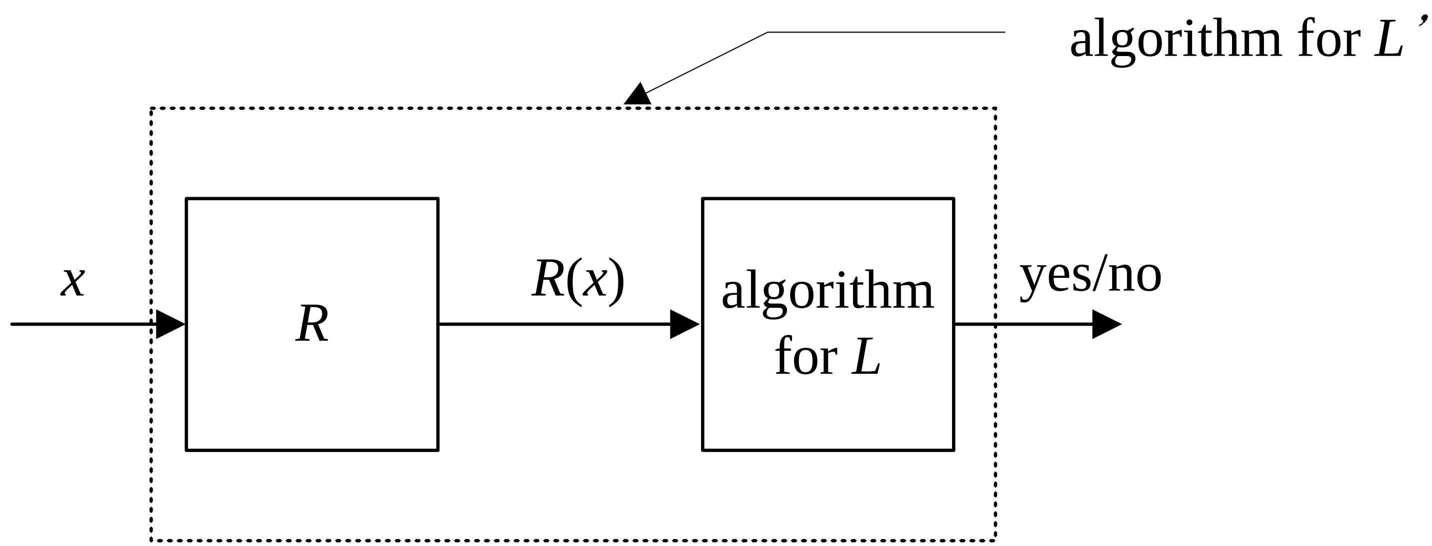
$$\forall x \; \{ \, x \in L' \text{ if and only if } R(x) \in L \, \}.$$

- Now we can answer "$x \in L'$?" for *any* x by answering "$R(x) \in L$?" because it has the same answer.
- L' is said to be **reduced** to L.

---

[a] Post (1944).

[b] Contributed by Mr. Tai-Dai Chou (J93922005) on May 19, 2005.


algorithm for *L'*
x
R
R(*x*)
algorithm for *L*
yes/no

Reductions in Proving Undecidability (concluded)

- 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.*

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,[a] we may say L_1 is a "special case" of L_2.[b]
- That is one way to understand the use of the term "reduction."

---

[a] Because R may not be onto.

[b] 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.

The Universal Halting Problem

- The **universal halting problem**:

$$H^* = \{ M : M \text{ halts on all inputs} \}.$$

- 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. 146), 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

- $\{ M; x : \text{there is a } y \text{ such that } M(x) = y \}$.
- $\{ M; x :$
  the computation M on input x uses all states of $M \}$.
- $L = \{ M; x; y : M(x) = y \}$.

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 M, which is deterministic.
- Swap the "yes" state and the "no" state of M.
- The new machine decides $\bar{L}$.[a]

[a] Recall p. 109.

Recursive and Recursively Enumerable Languages

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

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

[a]Either M or $\bar{M}$ (but not both) must accept the input and halt.

A Very Useful Corollary and Its Consequences

Corollary 11 *L is recursively enumerable but not recursive, then $\bar{L}$ is not recursively enumerable.*

- Suppose $\bar{L}$ is recursively enumerable.
- Then both L and $\bar{L}$ are recursively enumerable.
- By Lemma 10 (p. 154), L is recursive, a contradiction.

Corollary 12 *$\bar{H}$ is not recursively enumerable.*[a]

---

[a]Recall that $\bar{H} = \{ M; x : M(x) = \nearrow \}$.

R, RE, and coRE

RE: The set of all recursively enumerable languages.

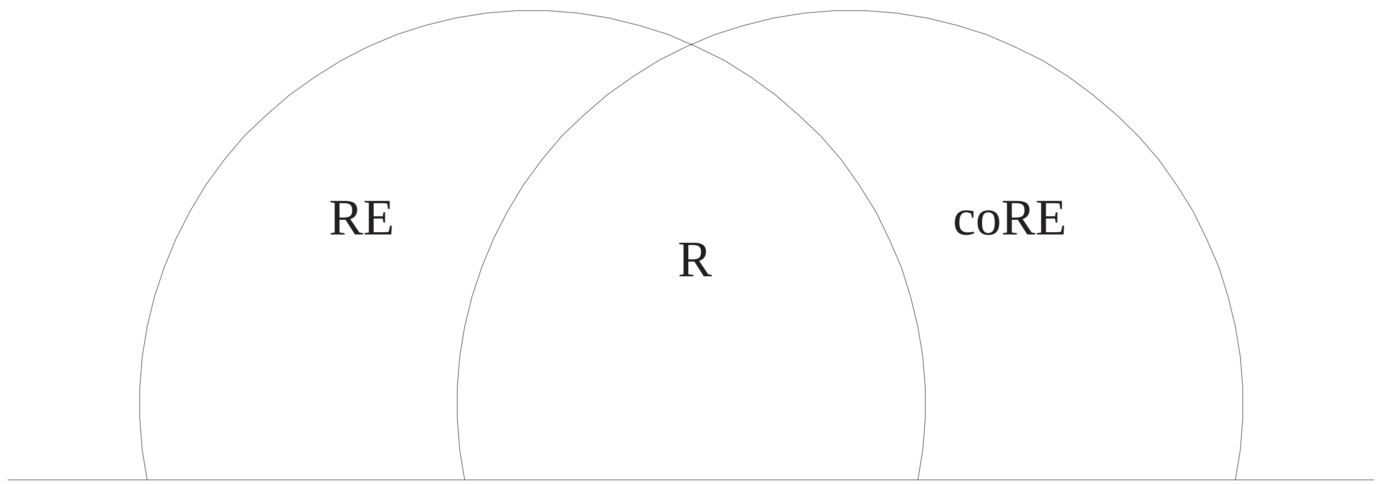
coRE: The set of all languages whose complements are recursively enumerable.

R: The set of all recursive languages.

- Note that coRE is not $\overline{\text{RE}}$.
  - $\text{coRE} = \{ L : \overline{L} \in \text{RE} \} = \{ \overline{L} : L \in \text{RE} \}$.
  - $\overline{\text{RE}} = \{ L : L \notin \text{RE} \}$.

R, RE, and coRE (concluded)

- R = RE $\cap$ coRE (p. 154).
- There exist languages in RE but not in R and not in coRE.
  - Such as H (p. 135, p. 136, and p. 155).
- There are languages in coRE but not in RE.
  - Such as $\bar{H}$ (p. 155).
- There are languages in neither RE nor coRE.


RE
R
coRE

H Is Complete for RE[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**.

[a]Post (1944).

Notations

- Suppose M is a TM accepting L.
- Write $L(M) = L$.
  - In particular, if $M(x) = \nearrow$ for all x, then $L(M) = \emptyset$.
- If $M(x)$ is never "yes" nor $\nearrow$ (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 $\mathcal{C}$ of all the recursively enumerable languages that satisfy it.
  - The property of *finite* recursively enumerable languages is

$$\{ L : L = L(M) \text{ for a TM } M, L \text{ is finite} \}.$$

- A property is **trivial** if $\mathcal{C} = \text{RE}$ or $\mathcal{C} = \emptyset$.
  - Answer to a trivial property is always "yes" or always "no."

Nontrivial Properties of Sets in RE (concluded)

- Here is a trivial property (always yes): Does the TM accept a recursively enumerable language?[a]
- A property is **nontrivial** if $\mathcal{C} \neq \text{RE}$ and $\mathcal{C} \neq \emptyset$.
  - In other words, answer to a nontrivial property is "yes" for some TMs and "no" for others.
- Here is a nontrivial property: Does the TM accept an empty language?[b]
- Up to now, all nontrivial properties (of recursively enumerable languages) are undecidable (pp. 151–152).
- In fact, Rice's theorem confirms that.

[a] Or, $L(M) \in \text{RE}$?
[b] Or, $L(M) = \emptyset$?

Rice's Theorem

Theorem 13 (Rice, 1956) *Suppose $\mathcal{C} \neq \emptyset$ is a proper subset of the set of all recursively enumerable languages. Then the question "$L(M) \in \mathcal{C}$?" is undecidable.*

- Note that the input is a TM program M.
- Assume that $\emptyset \notin \mathcal{C}$ (otherwise, repeat the proof for the class of all recursively enumerable languages *not* in $\mathcal{C}$).
- Let $L \in \mathcal{C}$ be accepted by TM M_L (recall that $\mathcal{C} \neq \emptyset$).
- Let M_H *accept* the undecidable language H.
  - M_H exists (p. 135).

The Proof (continued)

- Construct machine $M_x(y)$:

$$\textbf{if } M_H(x) = \text{“yes”} \textbf{ then } M_L(y) \textbf{ else } \nearrow$$

- On the next page, we will prove that

$$L(M_x) \in \mathcal{C} \text{ if and only if } x \in H. \qquad (1)$$

  - As a result, the halting problem is reduced to deciding $L(M_x) \in \mathcal{C}$.
  - Hence $L(M_x) \in \mathcal{C}$ must be undecidable, and we are done.

The Proof (concluded)

- Suppose $x \in H$, i.e., $M_H(x) =$ "yes."
  - $M_x(y)$ determines this, and it either accepts y or never halts, depending on whether $y \in L$.
  - Hence $L(M_x) = L \in \mathcal{C}$.
- Suppose $M_H(x) = \nearrow$.
  - M_x never halts.
  - $L(M_x) = \emptyset \notin \mathcal{C}$.

Comments

- $\mathcal{C}$ must be arbitrary.
- The following $M_x(y)$, though similar, will not work:

$$\textbf{if } M_L(y) = \text{“yes” } \textbf{then } M_H(x) \textbf{ else } \nearrow.$$

- Rice's theorem is about properties of the languages accepted by Turing machines.
- It then says any nontrivial property is undecidable.
- Rice's theorem is *not* about Turing machines themselves, such as "Does a TM contain 5 states?"

Consequences of Rice's Theorem

Corollary 14 *The following properties of recursively enumerative sets are undecidable.*

- *Emptiness.*
- *Finiteness.*
- *Recursiveness.*
- Σ^*.
- *Regularity.*
- *Context-freedom.*

Undecidability in Logic and Mathematics

- First-order logic is undecidable (answer to Hilbert's (1928) *Entscheidungsproblem*).[a]
- Natural numbers with addition and multiplication is undecidable.[b]
- Rational numbers with addition and multiplication is undecidable.[c]

---

[a] Church (1936).
[b] Rosser (1937).
[c] Robinson (1948).

Undecidability in Logic and Mathematics (concluded)

- Natural numbers with addition and equality is decidable and complete.[a]
- Elementary theory of groups is undecidable.[b]

---

[a] Presburger's Master's thesis (1928), his only work in logic. The direction was suggested by Tarski. Mojžesz Presburger (1904–1943) died in a concentration camp during World War II.

[b] Tarski (1949).

Julia Hall Bowman Robinson (1919–1985)

Alfred Tarski (1901–1983)