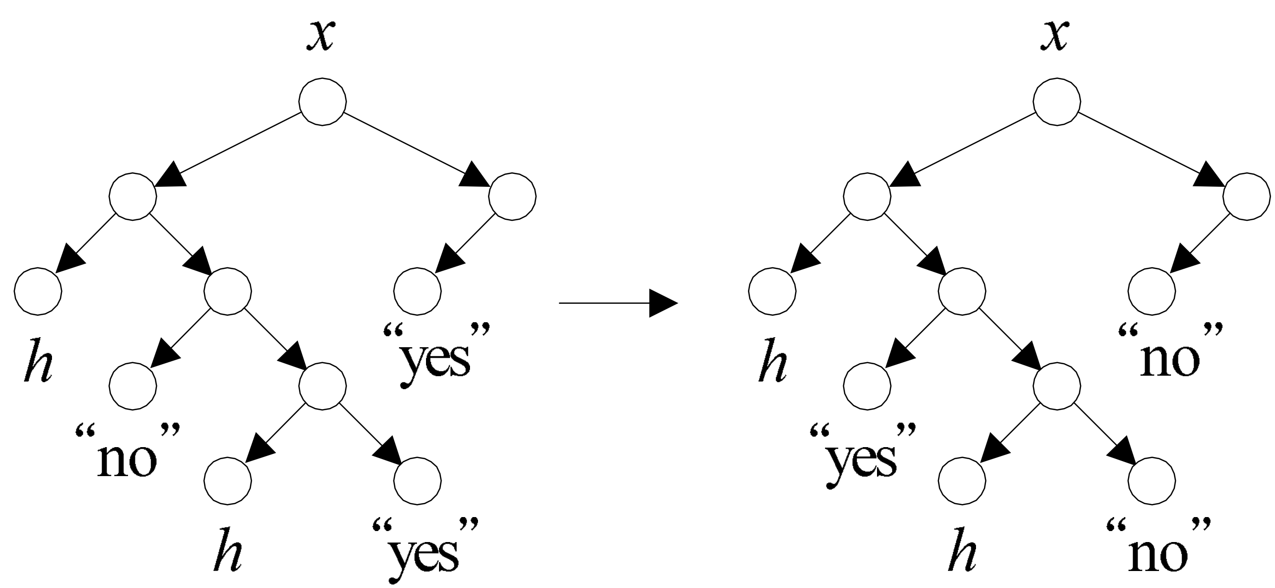
Complementing a TM's Halting States

- 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 complement each other.
- But if M is an NTM, then M' may not decide $\bar{L}$.
  - 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.


x
h
"no"
"yes"
h
"yes"
x
h
"yes"
"no"
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 ("Nondeterministic Polynomial")

- Define
$$\text{NP} \stackrel{\Delta}{=} \bigcup_{k>0} \text{NTIME}(n^k).$$
- Clearly $\text{P} \subseteq \text{NP}$.
- Think of NP as efficiently *verifiable* problems (see p. 337).
  - Boolean satisfiability (p. 121 and p. 196), e.g.
- The most important open problem in computer science is whether $\text{P} = \text{NP}$.

Simulating Nondeterministic TMs

Nondeterminism does not add power to TMs.

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 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 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] 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

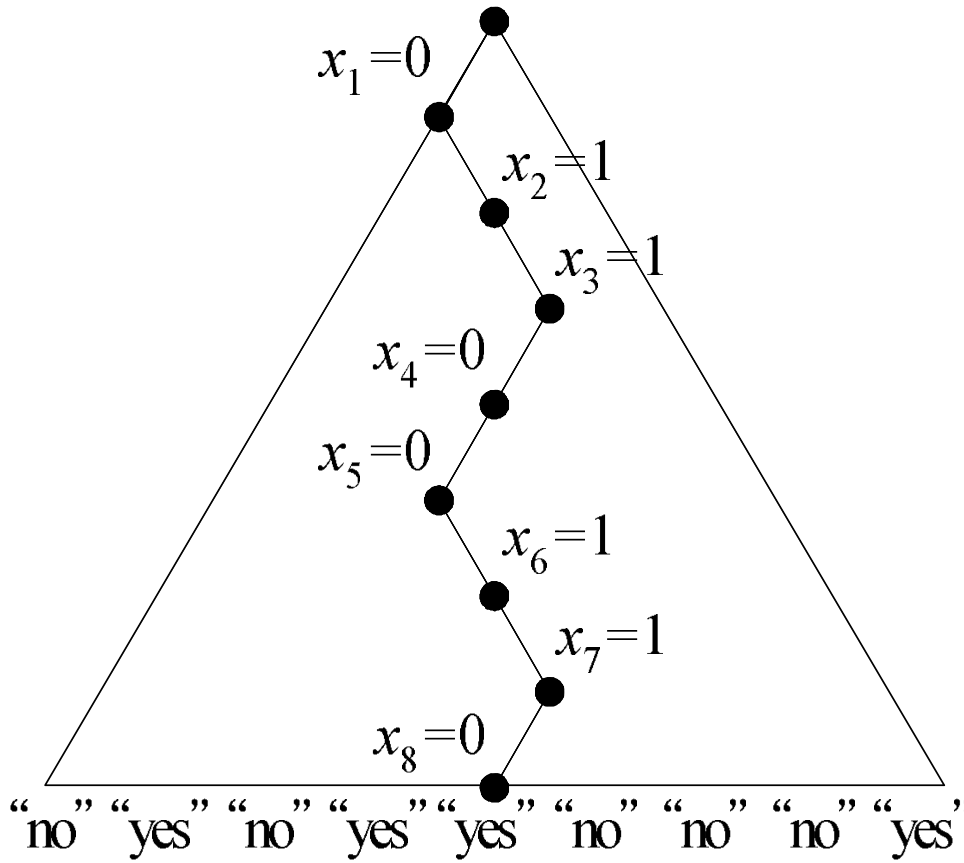
- Does converting an NTM into a TM *require* exploring all computation paths of the NTM as done in Theorem 6 (p. 118)?
- This is a key question in theory with important practical implications.

A Nondeterministic Algorithm for Satisfiability

ϕ is a boolean formula with n variables.

```
for i = 1, 2, ..., n do
    Guess x_i ∈ { 0, 1 }; {Nondeterministic choices.}
end for
{Verification:}
if φ(x_1, x_2, ..., x_n) = 1 then
    "yes";
else
    "no";
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[a] out of 2^n.
- Recall that ϕ is satisfiable if and only if there is a truth assignment that satisfies ϕ.

[a]Equivalently, a sequence of nondeterministic choices.

Analysis (concluded)

- The algorithm decides language

$$\{ \phi : \phi \text{ is satisfiable} \}.$$

  - Suppose ϕ is satisfiable.
    * There is a truth assignment that satisfies ϕ.
    * So there is a computation path that results in "yes."
  - Suppose ϕ is not satisfiable.
    * That means every truth assignment makes ϕ false.
    * So every computation path results in "no."
- General paradigm: Guess a "proof" then verify it.

The Traveling Salesman Problem

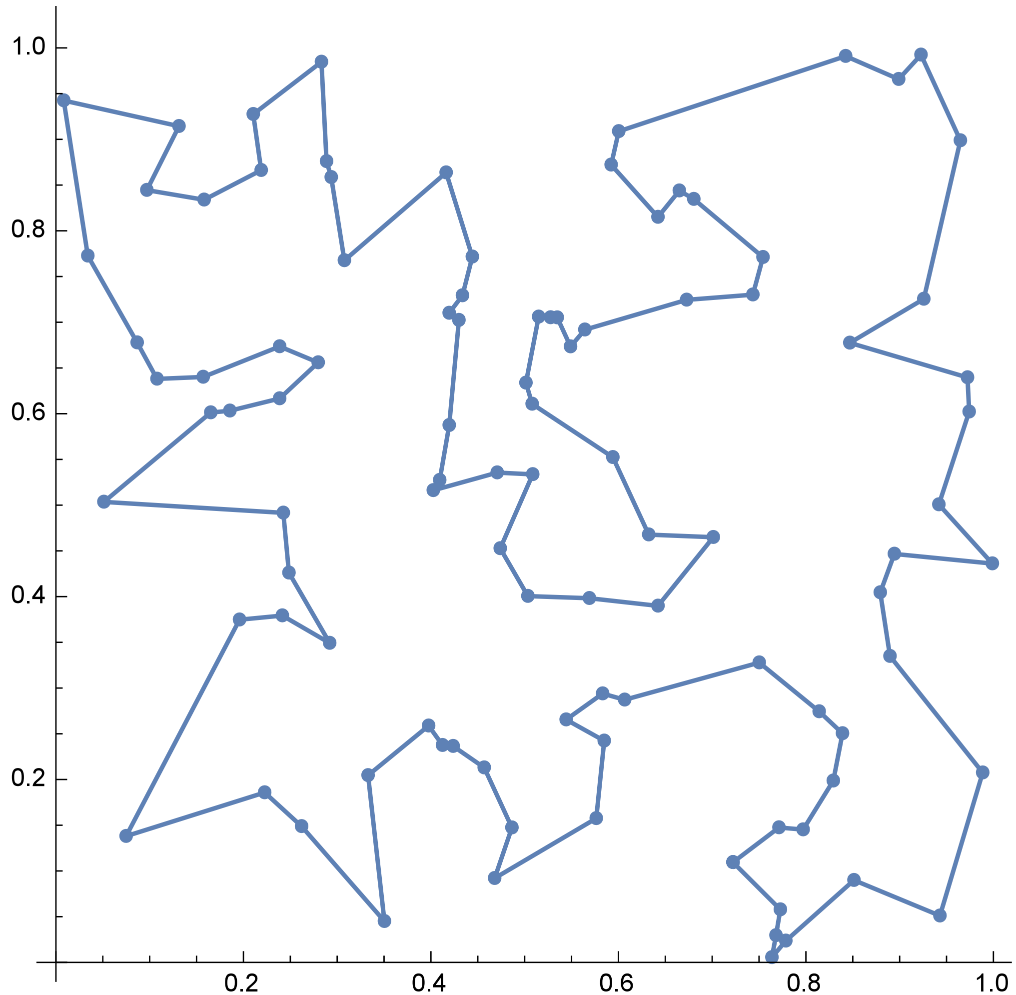
- We are given n cities $1, 2, \ldots, n$ and integer distance d_{ij} between any two cities i and j.
- Assume $d_{ij} = d_{ji}$ for convenience.
- The **traveling salesman problem** (TSP) asks for the total distance of the shortest tour of the cities.[a]
- The decision version TSP (D) asks if there 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. 409 and p. 509).

A Shortest Path



A Nondeterministic Algorithm for TSP (D)

```
for i = 1, 2, ..., n do
    Guess x_i ∈ { 1, 2, ..., n }; {The ith city.}[a]
end for
{Verification:}
if x_1, x_2, ..., x_n are distinct and Σ_{i=1}^{n-1} d_{x_i, x_{i+1}} ≤ 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 $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ Open Problem[a]

- Many practical applications depend on answers to the $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ 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 $f(n)$.
- NSPACE$(f(n))$ is a set of languages.
- As in the linear speedup theorem,[a] constant coefficients do not matter.

[a] Theorem 5 (p. 95).

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

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

  - REACHABILITY with more than one terminal node also has the same complexity.
  - In fact, REACHABILITY for *undirected* graphs is in SPACE$(\log n)$.[a]
- REACHABILITY $\in$ P (see, e.g., p. 240).

[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**:

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

  - 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:

```
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."

[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)

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

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).

***Psalms* 116:11, Old Testament:** "Everyone is a liar."

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

Self-Loop Paradoxes (continued)

A restaurant in Boston: No Name Restaurant.

U.S. Department of State (March 19, 2020): U.S. citizens who live in the United States should arrange for immediate return to the United States[.]

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)

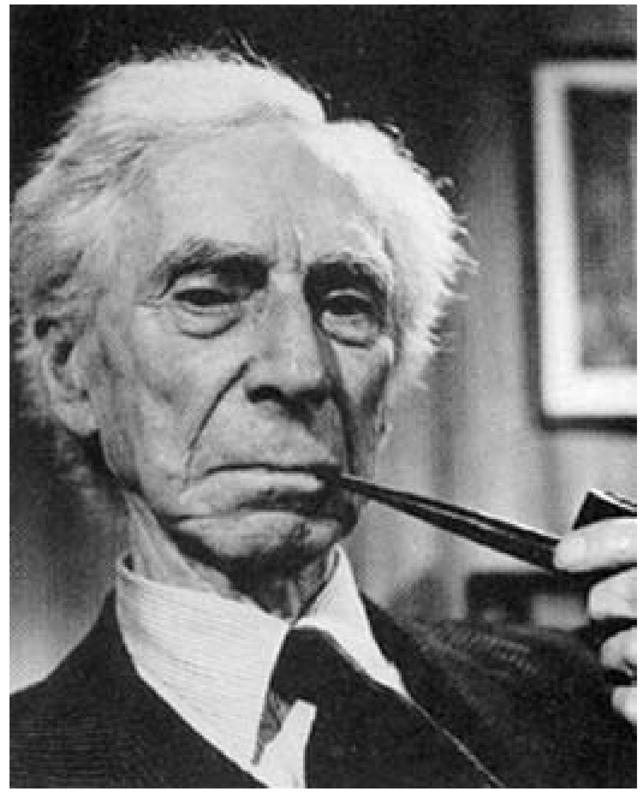
Jerome K. Jerome (1887), *Three Men in a Boat*: "How could I wake you, when you didn't wake me?"

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."

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

Bertrand Russell[a] (1872–1970)

Norbort 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."

[a]Nobel Prize in Literature (1950).

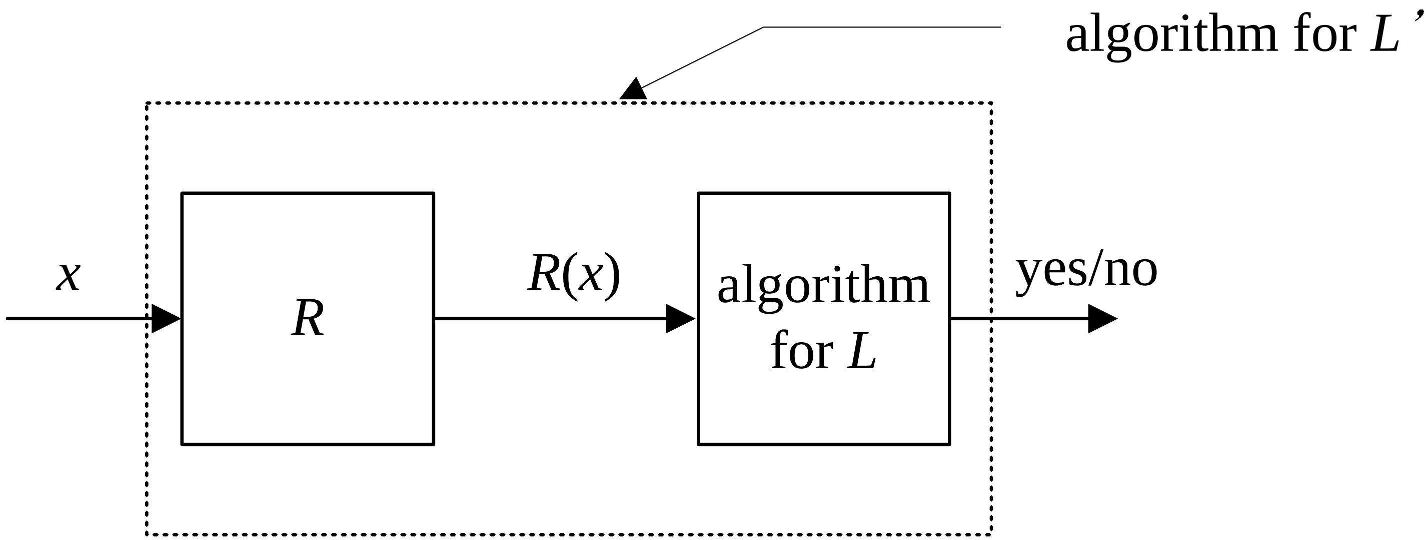
Reductions in Proving Undecidability

- 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.[a]
- As the reduction R maps members of L_1 to a *subset* of L_2,[b] we *may* say L_1 is a "special case" of L_2.[c]
- That is one way to understand the use of the somewhat confusing term "reduction."

---

[a] Intuitively, L_2 can be used to solve L_1.

[b] Because R may not be onto.

[c] 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. 150), 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 $M \}$.
- $\{ M; x; y : M(x) = y \}$.

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. 113.

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. 159), L is recursive, a contradiction.

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

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

R, RE, and coRE

RE: The set of all recursively enumerable languages.

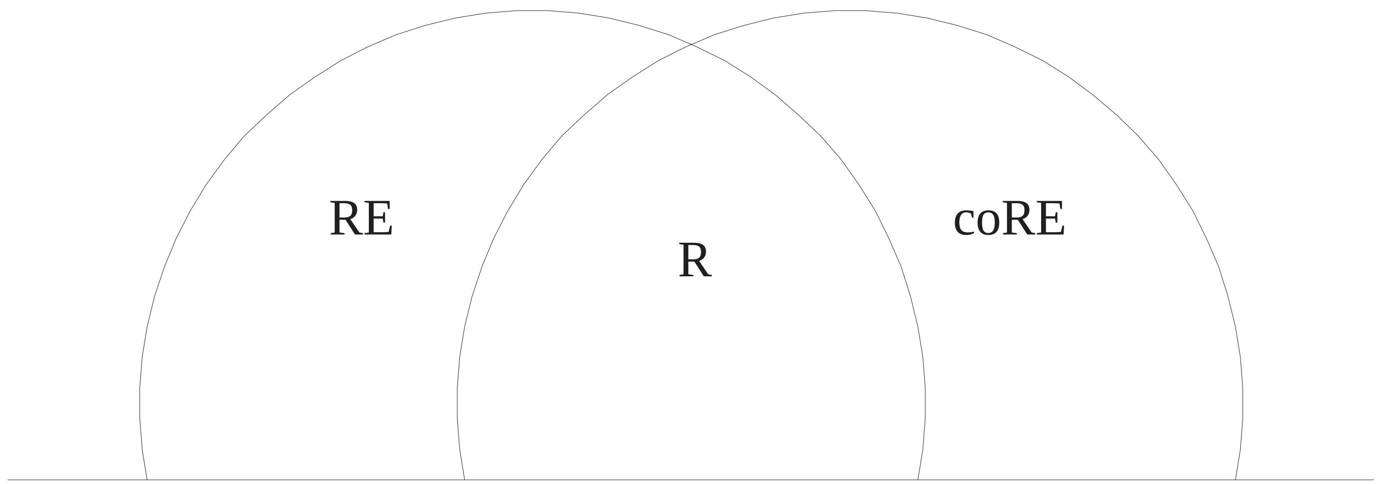
coRE: The set of all languages whose complements are recursively enumerable.

R: The set of all recursive languages.

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

R, RE, and coRE (concluded)

- R = RE $\cap$ coRE (p. 159).
- There exist languages in RE but not in R and not in coRE.
  - Such as H (p. 139, p. 140, and p. 160).
- There are languages in coRE but not in RE.
  - Such as $\bar{H}$ (p. 160).
- 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).