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 $m \leq n$.}
$x := a$;
for $i = 2, 3, \ldots, m$ **do**
    Guess $y \in \{ v_1, v_2, \ldots, v_m \}$; {The next node.}
    if $(x, y) \notin E$ **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. 237).

[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.

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

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).

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

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 restaurant in Boston: No Name Restaurant.

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

Self-Loop Paradoxes (concluded)

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

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!"

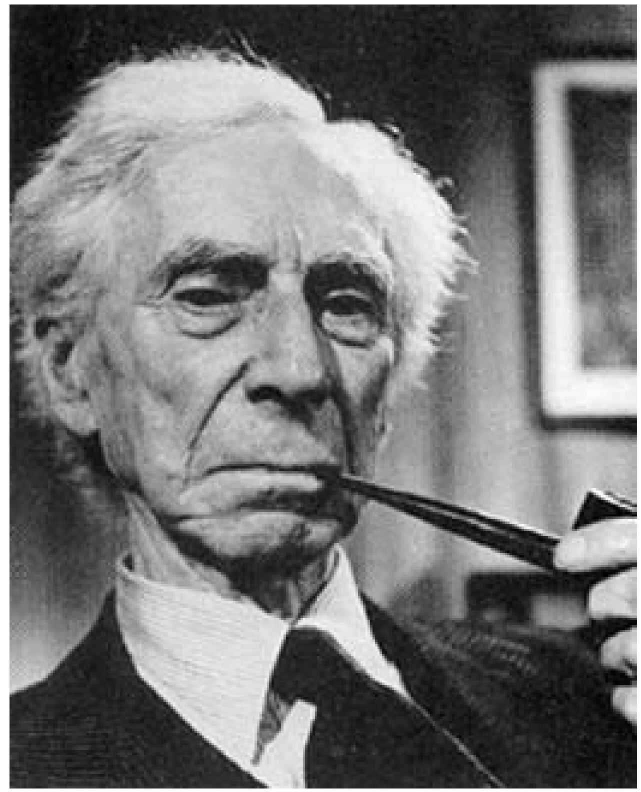
---

[a]See also *John* 14:10 and 17:21.

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

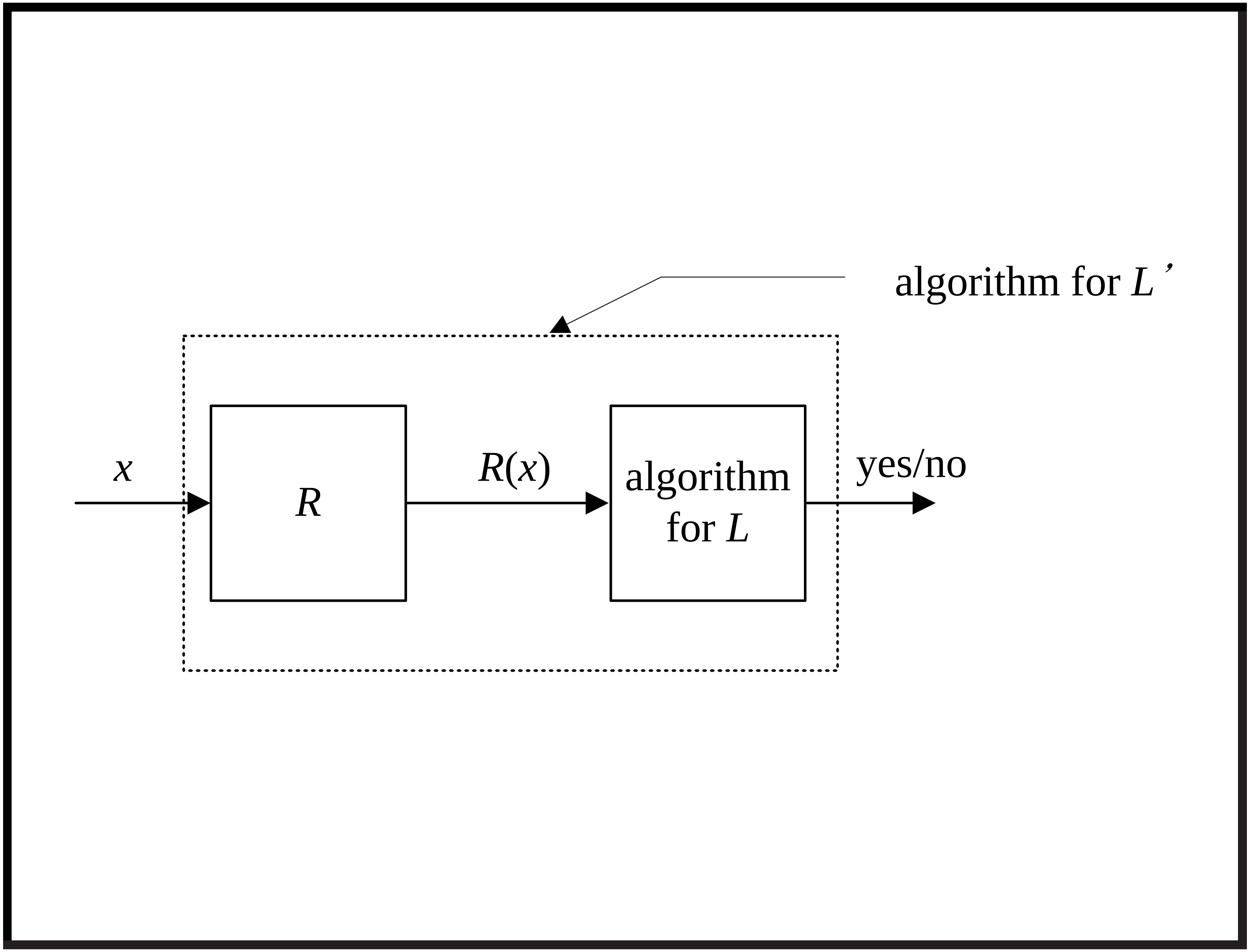
- Suppose we are asked to prove that L is undecidable.
- Suppose L' (such as H) is known to be undecidable.
- Find a computable transformation R (called **reduction**[a]) from L' to L such that[b]

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

- 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**:

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

- 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. 147), 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 $M \}$.
- $\{ M; x; y : M(x) = y \}$.

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

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. 111.

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

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. 156), 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.

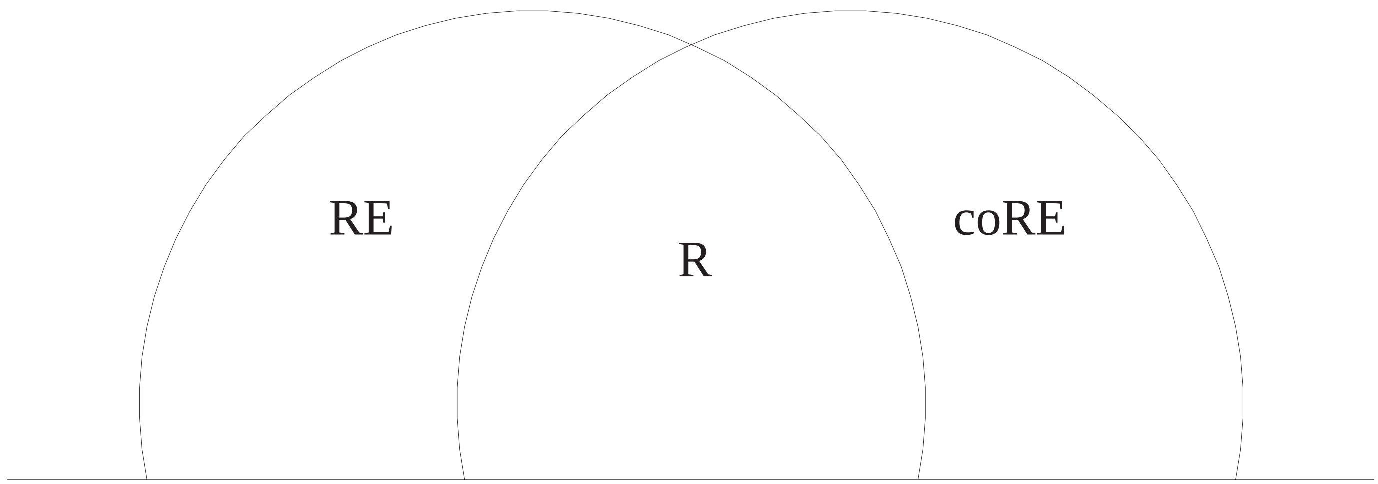
R: The set of all recursive languages.

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

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

R, RE, and coRE (concluded)

- $\mathrm{R} = \mathrm{RE} \cap \mathrm{coRE}$ (p. 156).
- There exist languages in RE but not in R and not in coRE.
  - Such as H (p. 137, p. 138, and p. 157).
- There are languages in coRE but not in RE.
  - Such as $\bar{H}$ (p. 157).
- 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 $L(M) = \emptyset$.

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

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. 153–154).
- 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 $\emptyset \not\in \mathcal{C}$ (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. 137).

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

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

  – 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.*[a]
- *Context-freedom.*[b]

---

[a] Is it a regular language?
[b] Is it a context-free language?

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)

Boolean Logic

Both of us had said the very same thing.
Did we both speak the truth
—or one of us did
—or neither?
— Joseph Conrad (1857–1924),
Lord Jim (1900)

Boolean Logic[a]

Boolean variables: $x_1, x_2, \ldots$.

Literals: x_i, $\neg x_i$.

Boolean connectives: $\vee, \wedge, \neg$.

Boolean expressions: Boolean variables, $\neg\phi$ (**negation**), $\phi_1 \vee \phi_2$ (**disjunction**), $\phi_1 \wedge \phi_2$ (**conjunction**).

- $\bigvee_{i=1}^n \phi_i$ stands for $\phi_1 \vee \phi_2 \vee \cdots \vee \phi_n$.
- $\bigwedge_{i=1}^n \phi_i$ stands for $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$.

Implications: $\phi_1 \Rightarrow \phi_2$ is a shorthand for $\neg\phi_1 \vee \phi_2$.

Biconditionals: $\phi_1 \Leftrightarrow \phi_2$ is a shorthand for $(\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$.

---

[a] George Boole (1815–1864) in 1847.

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

Truth Assignments

- A **truth assignment** T is a mapping from boolean variables to **truth values** `true` and `false`.
- A truth assignment is **appropriate** to boolean expression ϕ if it defines the truth value for every variable in ϕ.
  - $\{ x_1 = \texttt{true}, x_2 = \texttt{false} \}$ is appropriate to $x_1 \lor x_2$.
  - $\{ x_2 = \texttt{true}, x_3 = \texttt{false} \}$ is not appropriate to $x_1 \lor x_2$.

Satisfaction

- $T \models \phi$ means boolean expression ϕ is true under T; in other words, T **satisfies** ϕ.
- ϕ_1 and ϕ_2 are **equivalent**, written

$$\phi_1 \equiv \phi_2,$$

  if for any truth assignment T appropriate to both of them, $T \models \phi_1$ if and only if $T \models \phi_2$.

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

Truth Table[a]

- Suppose ϕ has n boolean variables.
- A **truth table** contains 2^n rows.
- Each row corresponds to one truth assignment of the n variables and records the truth value of ϕ under it.
- A truth table can be used to prove if two boolean expressions are equivalent.
  - Just check if they give identical truth values under all appropriate truth assignments.

---

[a] Post (1921); Wittgenstein (1922). Here, 1 is used to denote `true`; 0 is used to denote `false`. This is called the **standard representation** (Beigel, 1993).

A Truth Table

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

A Second Truth Table

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

A Third Truth Table

p	$\neg p$
0	1
1	0

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

Proof of Equivalency by the Truth Table:

$$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$$

p	q	$p \Rightarrow q$	$\neg q \Rightarrow \neg p$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	1	1

De Morgan's Laws[a]

- **De Morgan's laws** state that

$$\begin{aligned} \neg(\phi_1 \land \phi_2) &\equiv \neg\phi_1 \lor \neg\phi_2, \\ \neg(\phi_1 \lor \phi_2) &\equiv \neg\phi_1 \land \neg\phi_2. \end{aligned}$$

- Here is a proof of the first law:

ϕ_1	ϕ_2	$\neg(\phi_1 \land \phi_2)$	$\neg\phi_1 \lor \neg\phi_2$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

[a] Augustus DeMorgan (1806–1871) or William of Ockham (1288–1348).

Conjunctive Normal Forms

- A boolean expression ϕ is in **conjunctive normal form** (**CNF**) if

$$\phi = \bigwedge_{i=1}^{n} C_i,$$

where each **clause** C_i is the disjunction of zero or more literals.[a]

  – For example,

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3).$$

- Convention: An empty CNF is satisfiable, but a CNF containing an empty clause is not.

[a] Improved by Mr. Aufbu Huang (R95922070) on October 5, 2006.

Disjunctive Normal Forms

- A boolean expression ϕ is in **disjunctive normal form** (**DNF**) if

$$\phi = \bigvee_{i=1}^{n} D_i,$$

where each **implicant**[a] or simply **term** D_i is the conjunction of zero or more literals.

  - For example,

$$(x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3).$$

[a] D_i implies ϕ, thus the term.

Clauses and Implicants

- The $\bigvee$ of clauses yields a clause.
  - For example,

$$\begin{aligned} & (x_1 \lor x_2) \lor (x_1 \lor \neg x_2) \lor (x_2 \lor x_3) \\ = \;& x_1 \lor x_2 \lor x_1 \lor \neg x_2 \lor x_2 \lor x_3. \end{aligned}$$

- The $\bigwedge$ of implicants yields an implicant.
  - For example,

$$\begin{aligned} & (x_1 \land x_2) \land (x_1 \land \neg x_2) \land (x_2 \land x_3) \\ = \;& x_1 \land x_2 \land x_1 \land \neg x_2 \land x_2 \land x_3. \end{aligned}$$

Any Expression ϕ Can Be Converted into CNFs and DNFs

$\phi = x_j$:

- This is trivially true.

$\phi = \neg\phi_1$ and a CNF is sought:

- Turn ϕ_1 into a DNF.
- Apply de Morgan's laws to make a CNF for ϕ.

$\phi = \neg\phi_1$ and a DNF is sought:

- Turn ϕ_1 into a CNF.
- Apply de Morgan's laws to make a DNF for ϕ.

Any Expression ϕ Can Be Converted into CNFs and DNFs (continued)

$\phi = \phi_1 \vee \phi_2$ and a DNF is sought:

- Make ϕ_1 and ϕ_2 DNFs.

$\phi = \phi_1 \vee \phi_2$ and a CNF is sought:

- Turn ϕ_1 and ϕ_2 into CNFs,[a]

$$\phi_1 = \bigwedge_{i=1}^{n_1} A_i, \quad \phi_2 = \bigwedge_{j=1}^{n_2} B_j.$$

- Set

$$\phi = \bigwedge_{i=1}^{n_1} \bigwedge_{j=1}^{n_2} (A_i \vee B_j).$$

---

[a] Corrected by Mr. Chun-Jie Yang (`R99922150`) on November 9, 2010.

Any Expression ϕ Can Be Converted into CNFs and DNFs (concluded)

$\phi = \phi_1 \wedge \phi_2$ and a CNF is sought:

- Make ϕ_1 and ϕ_2 CNFs.

$\phi = \phi_1 \wedge \phi_2$ and a DNF is sought:

- Turn ϕ_1 and ϕ_2 into DNFs,

$$\phi_1 = \bigvee_{i=1}^{n_1} A_i, \quad \phi_2 = \bigvee_{j=1}^{n_2} B_j.$$

- Set

$$\phi = \bigvee_{i=1}^{n_1} \bigvee_{j=1}^{n_2} (A_i \wedge B_j).$$

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

An Example: Turn $\neg((a \wedge y) \vee (z \vee w))$ into a DNF

$$
\begin{aligned}
& \neg((a \wedge y) \vee (z \vee w)) \\
\stackrel{\neg(\text{CNF}\vee\text{CNF})}{=} \quad & \neg(((a) \wedge (y)) \vee ((z \vee w))) \\
\stackrel{\neg(\text{CNF})}{=} \quad & \neg((a \vee z \vee w) \wedge (y \vee z \vee w)) \\
\stackrel{\text{de Morgan}}{=} \quad & \neg(a \vee z \vee w) \vee \neg(y \vee z \vee w) \\
\stackrel{\text{de Morgan}}{=} \quad & (\neg a \wedge \neg z \wedge \neg w) \vee (\neg y \wedge \neg z \wedge \neg w).
\end{aligned}
$$

©2019 Prof. Yuh-Dauh Lyuu, National Taiwan University

Functional Completeness

- A set of logical connectives is called **functionally complete** if every boolean expression is equivalent to one involving only these connectives.
- The set $\{\neg, \vee, \wedge\}$ is functionally complete.
  - Every boolean expression can be turned into a CNF, which involves only $\neg$, $\vee$, and $\wedge$.
- The sets $\{\neg, \vee\}$ and $\{\neg, \wedge\}$ are functionally complete.[a]
  - By the above result and de Morgan's laws.
- $\{$ NAND $\}$ and $\{$ NOR $\}$ are functionally complete.[b]

[a] Post (1921).
[b] Peirce (c. 1880); Sheffer (1913).

Satisfiability

- A boolean expression ϕ is **satisfiable** if there is a truth assignment T appropriate to it such that $T \models \phi$.
- ϕ is **valid** or a **tautology**,[a] written $\models \phi$, if $T \models \phi$ for all T appropriate to ϕ.

---

[a]Wittgenstein (1922). Wittgenstein is one of the most important philosophers of all time. Russell (1919), "The importance of 'tautology' for a definition of mathematics was pointed out to me by my former pupil Ludwig Wittgenstein, who was working on the problem. I do not know whether he has solved it, or even whether he is alive or dead." "God has arrived," the great economist Keynes (1883–1946) said of him on January 18, 1928, "I met him on the 5:15 train."

Satisfiability (concluded)

- ϕ is **unsatisfiable** or a **contradiction** if ϕ is false under all appropriate truth assignments.
  - Or, equivalently, if $\neg\phi$ is valid (prove it).
- ϕ is a **contingency** if ϕ is neither a tautology nor a contradiction.

Ludwig Wittgenstein (1889–1951)

Wittgenstein (1922), "Whereof one cannot speak, thereof one must be silent."

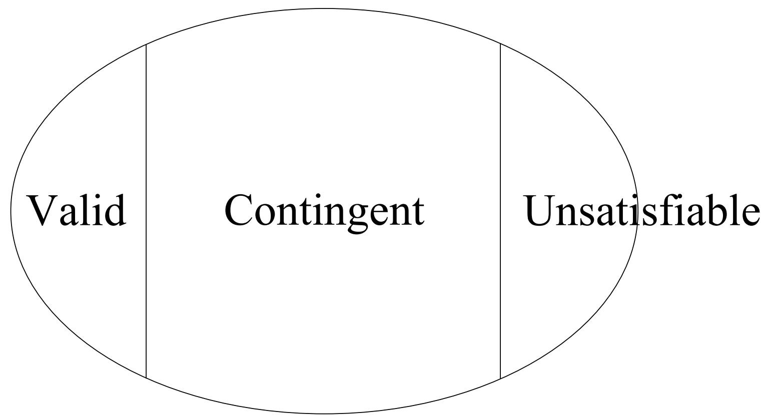
SATISFIABILITY (SAT)

- The **length** of a boolean expression is the length of the string encoding it.
- SATISFIABILITY (SAT): Given a CNF ϕ, is it satisfiable?
- Solvable in exponential time on a TM by the truth table method.
- Solvable in polynomial time on an NTM, hence in NP (p. 119).
- A most important problem in settling the "P $\stackrel{?}{=}$ NP" problem (p. 317).

UNSATISFIABILITY (UNSAT or SAT COMPLEMENT) and VALIDITY

- UNSAT (SAT COMPLEMENT): Given a boolean expression ϕ, is it unsatisfiable?
- VALIDITY: Given a boolean expression ϕ, is it valid?
  - ϕ is valid if and only if $\neg\phi$ is unsatisfiable.
  - ϕ and $\neg\phi$ are basically of the same length.
  - So UNSAT and VALIDITY have the same complexity.
- Both are solvable in exponential time on a TM by the truth table method.

Relations among SAT, UNSAT, and VALIDITY



- The negation of an unsatisfiable expression is a valid expression.
- None of the three problems—satisfiability, unsatisfiability, validity—are known to be in P.