

Chap. 9 Undecidability

9.1 A Language that is Not recursively enumerable

9.1.2 Code for Turing Machine

TM $M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, \{q_2\}) \leftrightarrow \text{binary string(integer)}$

$$Q = \{q_1, q_2, \dots, q_r\} \quad r \in \mathbb{N}.$$

$$\Gamma = \{X_1, X_2, \dots, X_s\} \quad X_1 = 0, X_2 = 1, X_3 = B \quad s \in \mathbb{N}.$$

$$L = D_1, R = D_2. \quad 2 \in \mathbb{N}.$$

$$\delta(q_i, X_j) = (q_k, X_l, D_m) \quad i, j, k, l, m \in \mathbb{N}.$$

$$\leftrightarrow 0^i 10^j 10^k 10^l 10^m$$

$$\delta \leftrightarrow \delta_1 11 \delta_2 11 \dots 11 \delta_n.$$

$$(M, w) \leftrightarrow \text{code}(M) 111 \text{code}(w) \in \{0, 1\}^* \quad \text{binary string}$$

\therefore number of Turing machines is **countable**.

We can *enumerate* TM M_i for $i \in \mathbb{N}$.

Diagonalization Language: L_d .

Since both of TM's and strings in Σ^* are **countable**,
we can consider (M_i, w_i) pair for $i \in \mathbb{N}$.

Consider $L_d = \{w_i \in \Sigma^* \mid w_i \notin L(M_i)\}$

Theorem 9.2 L_d is **not** recursively enumerable.

Figure 9.1

proof Suppose $L_d = L(M)$ for some TM M .

Since M is a TM, $\exists i \in \mathbb{N} . \exists. M = M_i$.

If $w_i \in L_d$, $w_i \notin L_d$ by definition of L_d . $\therefore M$ does **not** accept w_i .

If $w_i \notin L_d$, $w_i \in L_d$ by definition of L_d . $\therefore M$ **accepts** w_i .

Contradiction! $\therefore M$ does **not exist**.

$\therefore L_d$ is **not** recursively enumerable. **Cantor's diagonal argument**
TM's are **countable** whereas languages are **uncountable!**

The halting problem

```

program halt(P: program, I: input)
  if P(I) will stop then print “halts”
  else print “loops forever” fi

```

Assume the program halt exists and consider a program *H*

```

program H(P: program)
  if halt(P, P) = “halts” then loops forever
  else stop fi

```

Consider *H*(*H*)

if *H*(*H*) loops forever \Rightarrow halt(*H*, *H*) prints “stop”
 \Rightarrow But *H*(*H*) must stop (definiton of *H*).

if *H*(*H*) stop \Rightarrow halt(*H*, *H*) prints “no stop”
 \Rightarrow But *H*(*H*) must loops forever (definiton of *H*).

\therefore Contradiction! halt does **not** exist.

\therefore halting problem does **not** exist.

Languages(sets) that is **not** RE(no TM, no **program**)

$$L_d = \{w_i \in \Sigma^* \mid w_i \notin L(M_i)\}$$

halting problem

power set of integer is **uncountable**

Cantor's diagonal arguments

Russel's paradox

$$S = \{x \mid x \notin x\} \quad x \in x, \text{ iff } x \notin x. \quad \text{But } S \in S, \text{ iff } S \notin S!$$

Some similar examples in the world

A barber who shave everybody who can **not** shave himself.

Shall the barber shave **himself**?

An adjective is heterological, if the adjective does **not** possess
the property it describes.(monosyllabic, polysyllabic)

Is the adjective "**heterological**" **heterological**?

There is a sign that "It is written by **me**(**liar**)".Did you(**liar**) write it?

Self contradiction **denial of self recursion!**

9.2 An undecidable problem that is RE

Recursive languages

If $w \in L$, M halts and accepts.

If $w \notin L$, M halts and does not accept.

subclass of RE languages

type 1 in Chomsky's hierarchy

Recursively enumerable languages (RE languages)

If $w \in L$, M halts and accepts.

If $w \notin L$, M halts and does not accept or loops forever.

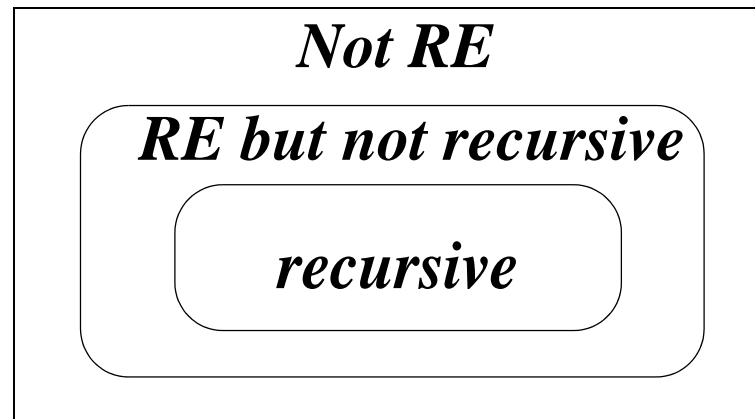
*Problem P is called **decidable**, if P is **recursive**.*

*Problem P is called **undecidable**, if P is **not recursive**.*

*P may be **RE or non-RE**.*

Three classes of languages(problems)

<i>recursive</i>	<i>decidable</i>	<i>countable</i>
<i>RE but not recur.</i>	<i>undecidable</i>	<i>countable</i>
<i>not RE</i>	<i>undecidable</i>	<i>uncountable</i>



Recursive

Decidable(algorithm)

total (recursive) function

Recursive enumerable

Turing computable

partial recursive function

programmable(computable)

Languages and problems

$$L: \Sigma^* \rightarrow \{0, 1\} \qquad P: \mathbb{N} \rightarrow \{0, 1\}$$

*Both of languages and problems are **uncountable**.*

*But TM(program) are **countable**.*

*There are problems(languages) that is **not** recursively enumerable.*

halting problem

Russel's paradox

Diagonalization languages(L_d)

$$\text{complement of } L_d \qquad \bar{L}_d = \{w_i \in \Sigma^* \mid w_i \in L(M_i)\} = L_u$$

universal language in

*RE(but **not recursive**)*

Complement of recursive and recursively enumerable languages

Theorem 9.3 *If L is recursive, \bar{L} is also recursive.*

proof *Let $L = L(M)$ for some TM M that always halts.*

Consider \bar{M}

accept and halt \rightarrow not accept and halt.

not accept and halt \rightarrow accept and halt.

$\exists \bar{M} . \exists . \bar{L} = L(\bar{M})$ and always halts.

More detail

$M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, \{q_2\})$ *no transition from F*

$\bar{M} = (Q \cup \{f\}, \{0, 1\}, \Gamma, \bar{\delta}, q_1, B, \{f\})$

$\bar{\delta} = \delta \cup \{\delta(q, X) = (f, X, S) \mid q \in Q\}$

no transition in $M \rightarrow$ transition to f

Read Why “Recursive”? *in page 385*

Theorem 9.4 If both of L and \bar{L} are RE, L is **recursive** (so is \bar{L}).

proof Let $L = L(M_1)$ and $\bar{L} = L(M_2)$.

Consider a **two** tape TM M .

tape 1 simulates the tape of M_1 and tape 2 simulates the tape of M_2 .
states and state transitions of M simulates M_1 and M_2 in **parallel**.

If $x \in L \rightarrow M_1$ **accept** and halt $\rightarrow M$ accept x and **halt**.

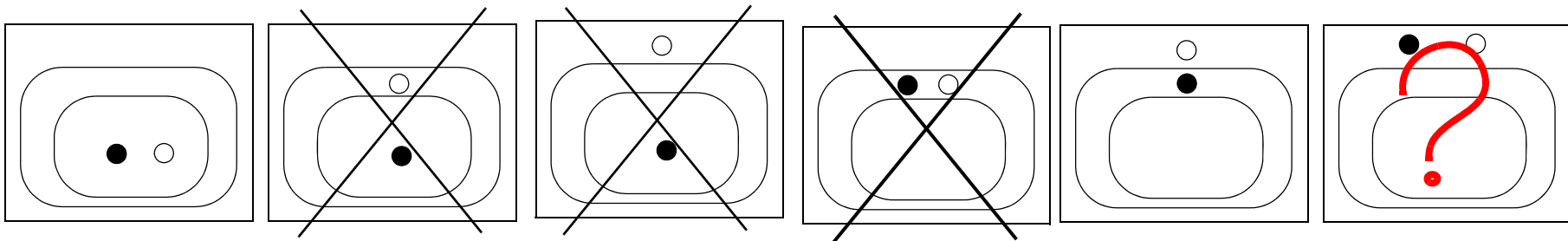
If $x \notin L \rightarrow M_2$ **accept** and halt $\rightarrow M$ **not** accept x and **halt**.

$\therefore L$ is **recursive**.

Only **three** case for the **complement** of language (among $6({}_3P_3)$ cases)

Thm 9.3: If L is **recursive**, \bar{L} is **recursive** and vice versa.

Thm 9.4: L and \bar{L} are **not** both RE and **not recursive**.



Universal language: L_u : complement of L_d .

$$L_u = \{w_i \in \Sigma^* \mid w_i \in L(M_i)\} = \bar{L}_d.$$

Theorem 9.6 L_u is **RE** but **not recursive**.

proof Let U , **universal TM**, be a multi tape TM such that $L(U) = L_u$.

tape 1: (M_i, w_i)

tape 2: simulate the tape of M_i .

If M_i accept w_i , U accept M_i .

$\therefore L_u$ is **RE**. (simplified version of 9.2.3)

Since L_d is **not RE**.

$\therefore L_u$ is **not recursive**. (Three cases for the complement)

$\therefore L_u$ is **RE** but **not recursive**.

$L_d = \bar{L}_u$ is **not RE**. (the fifth case in the diagram of TP p9)

9.3 Undecidable Problems About Turing Machines

P is a *decision problem* on the domain D , if $\forall d \in D$, $P(d)$ is *yes* or *no*.

$P: D \rightarrow \{\text{yes}, \text{no}\}$

$Y_P, N_P \subseteq D$ is called *yes(no) instances* of P , if

$Y_P = \{d \in D \mid P(d) = \text{yes}\}$

$N_P = \{d \in D \mid P(d) = \text{no}\}$, respectively.

A *problem* P is *decidable*,

if there exists a *decider* (program, algorithm, TM) that always tells *yes* or *no* correctly.

Undecidable, otherwise.

Assume D is countable.

Then $|\{P(D)\}|$ is uncountable.

But decider is countable.

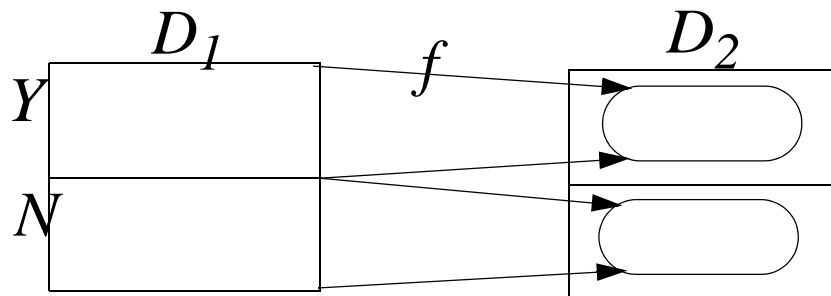
$\therefore \exists P, \exists P$ is undecidable.

Reducing one problem to another

We say a problem P_1 on D_1 **reduces** (**subset**) to P_2 on D_2 , if

$$\exists f: D_1 \rightarrow D_2, \text{ s.t. } \forall d_1 \in Y_{P_1}(D_1), f(d_1) \in Y_{P_2}(D_2)$$

$$\forall d_1 \in N_{P_1}(D_1), f(d_1) \in N_{P_2}(D_2)$$



If P_1 (on D_1) **reduces** to P_2 (on D_2), **BUT!!!**

P_2 is at least as **hard** as P_1 . ($P_1 \leq P_2$) (i**subset**)

P_2 is **not easier** than P_1 . ($P_1 \not\leq P_2$) \equiv ($P_1 \leq P_2$)

Theorem 9.7 If there is a **reduction** from P_1 to P_2 ($P_1 \leq P_2$), then:

a) If P_1 is **undecidable**, then so is P_2 . (If P_2 is **decidable**, so is P_1 .)

b) If P_1 is **non-RE**, then so is P_2 . (If P_2 is **RE**, so is P_1 .)

proof (대우) a) Suppose P_2 is **decidable**. Then

$\exists D_2 . \forall x \in P_1, f(x) \in P_2, D_2$ halts “yes”, D_1 halts “yes”.

$\forall x \notin P_1, f(x) \notin P_2, D_2$ halts “no”, D_1 halts “no”.

$\therefore P_1$ is **decidable**.

b) Assume P_2 is **RE**. Then

$\exists M_2 . \forall x \in P_1, f(x) \in P_2, M_2$ halts “yes”, M_1 halts “yes”.

$\forall x \notin P_1, f(x) \notin P_2, M_2$ halts “no” or loops forever,

M_1 halts “no” or loops forever.

$\therefore P_1$ is **RE**.

If L_u reduces to P , P is **not recursive**. (RE or not RE)

9.3.2 Turing Machine that Accepts the Empty Language

$$L_e = \{M \mid L(M) = \emptyset\}$$

$$L_{ne} = \{M \mid L(M) \neq \emptyset\}$$

Theorem 9.8 L_{ne} is recursively enumerable.

proof Consider a NTM M_{ne}

1. Guess a TM M and an input string w
2. A TM U test if M accepts w . (U simulates M for w)
3. If M accepts w then M_{ne} accepts M .

$$\therefore L(M_{ne}) = L_{ne}.$$

But it is not so easy to find a TM M_e \exists . $L(M_e) = L_e$.

Actually there is no TM M_e \exists . $L(M_e) = L_e$.

We shall prove that in Thm. 9.9 and 9.10.

Theorem 9.9 L_{ne} is not recursive.

proof Reduce L_u to L_{ne} . ($L_u \leq L_{ne}$)

Consider a TM M_{ne} .

1. U simulates M for w . (guess (M, w) pair)
2. If U accepts w ($w \in L(M)$), then code for $M \in L_{ne}$.
3. If U does not accept w ($w \notin L(M)$), then code for $M \notin L_{ne}$.

Transform (M, w) pair to M_{ne} . $\exists. L(M_{ne}) = \{M \mid w \in L(M)\}$

If $w \in L(M)$, $M \in L_{ne}$.

If $w \notin L(M)$, $M \notin L_{ne}$.

\therefore We Reduced L_u to L_{ne} .

$\therefore L_{ne}$ is not recursive.

Theorem 9.10 L_e is not RE.

proof Since $L_e = \overline{L_{ne}}$ and L_{ne} is RE but not recursive. Case 2 of p7.

Rice's Theorem and Properties of RE Languages

Consider a **property** P of a set of languages.

property of being **context free** is set of all CFL's.

property of being **empty** is $\{\emptyset\}$.

$P: 2^{\Sigma^*} \rightarrow \{\text{true}, \text{false}\}$

$P = \{L \subseteq \Sigma^* \mid P(L)\} = \{L \in 2^{\Sigma^*} \mid P(L)\}.$

A **property** is **trivial**, if it is either **empty** or is **all** of the languages.
nontrivial otherwise.

$P = \emptyset$ or 2^{Σ^*} are trivial.

But $P = \{\emptyset\}$ is nontrivial.

P may be represented as set of TM's, L_P

$P \leftrightarrow L_P = \{M \in TM \mid L(M) = L\}$

Theorem 9.11 (Rice's Theorem) Every nontrivial property of the recursively enumerable languages are **undecidable**.

proof

Let P be a nontrivial property of RE languages.

1. Assume $P(\emptyset) = \text{false}$ (or $\emptyset \notin P$).

$$\therefore \exists L . \exists . P(L) = \text{true} \text{ or } \exists M_L . \exists . M_L \in L_P$$

We shall **reduce** L_u to L_P . ($L_u \leq L_P$)

1. U simulate w for M

2. If U accepts w , $L = L(M) \leftrightarrow \exists M_L \in L_P$

2.1 M_L simulate for x

2.2 If M_L accepts x , M accepts M_L .

3. If U does not accept w , do nothing.

2. Assume $P(\emptyset) = \text{true}$ (or $\emptyset \in P$).

Consider complement property \bar{P} , $L_{\bar{P}} = \bar{L}_P$

$\therefore \bar{P}$ is undecidable(above).
 $\therefore P$ is undecidable.(Thm 9.3)

Every **nontrivial** property of the **recursively enumerable** languages are either **RE but not recursive** or **not RE**.

Two examples including complements

L_u is **RE but not recursive**

$L_d = \bar{L}_u$ is not **not RE**.

L_{ne} is **RE but not recursive**

$L_e = \bar{L}_{ne}$ is not **not RE**.

Every **nontrivial** property of the **recursively enumerable** languages are either **RE but not recursive** or **not RE**.

9.3.4 Problems about Turing-Machine Specifications

- 1. Whether the language accepted by a TM is empty(L_e , L_{ne}).*
- 2. Whether the language accepted by a TM is finite.*
- 3. Whether the language accepted by a TM is regular.*
- 4. Whether the language accepted by a TM is context-free.*

Undecidable!

Problem is a language

<i>language</i>	<i>problem</i>	<i>function</i>
<i>Not R.E.</i>	<i>Not computable</i>	
<i>recursively enumerable</i>	<i>computable</i>	<i>partial function</i>
<i>recursive</i>	<i>decidable(alway halt)</i>	<i>(total) function</i>

Three class of languages(problems)

1. Not recursively enumerable

*Neither **computable** nor **programmable** (**undecidable**)*

2. Recursively enumerable but not recursive

***Computable** or **programmable** but **not** decidable (**undecidable**)*

partial μ -recursive function

3. recursive

decidable

total μ -recursive function

Turing Church's Thesis

Turing machine

Turing(1930)

μ -recursive function(partial recursive) function

Gödel(1934; lecture at Princeton), Herbrand, Kleene(1936)

λ -calculus

Church(1933-41), Kleene(1935), Rosser(1935)

Equivalence of μ -recursive function and λ -calculus

Church(1936) attributes to Kleene

Equivalence of TM, μ -recursive function, and λ -calculus

Turing(1937)

Turing-Church's Thesis

Following systems are equivalent

<i>Turing machine</i>	<i>Turing(1930)</i>
<i>μ-recursive function(partial recursive) function</i>	<i>Gödel(1934)</i>
<i>λ-calculus</i>	<i>Church(1933-41)</i>
<i>combinatory logic</i>	<i>Schönfinkel(1924), Curry(1929)</i>
<i>Post correspondence system</i>	<i>Post(1936)</i>
<i>type 0 grammar</i>	<i>Chomsky(1959)</i>
<i>while programs</i>	<i>Meyer, Ritchie(1967)</i>

*Turing-Church's thesis **still** works*