Chap. 9 Undecidability

9.1 A Language that is Not recursively enumerable

9.1.2 Code for Turing Machine

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

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

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

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

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

$\leftrightarrow 0^i 1 0^j 1 0^k 1 0^l 1 0^m$

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

$(M, w) \leftrightarrow code(M)111code(w) \in \{0, 1\}^*$ $\quad$ 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, $\exists i \in \mathbb{N}$.э. $M = M_i$.

*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**)*

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

halting problem

*power set of integer is **uncountable***

Cantor's diagonal arguments

Russel's paradox

$S = \{x | \; x \notin x\}$ $\quad x \in x$, iff $x \notin x$. $\quad$ *But* $S \in S$, 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 posess 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 accepts****.*

subclass *of RE languages*

type 1 in Chomsky's hieratchy

Recursivly enumerable languages(RE languages)

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

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

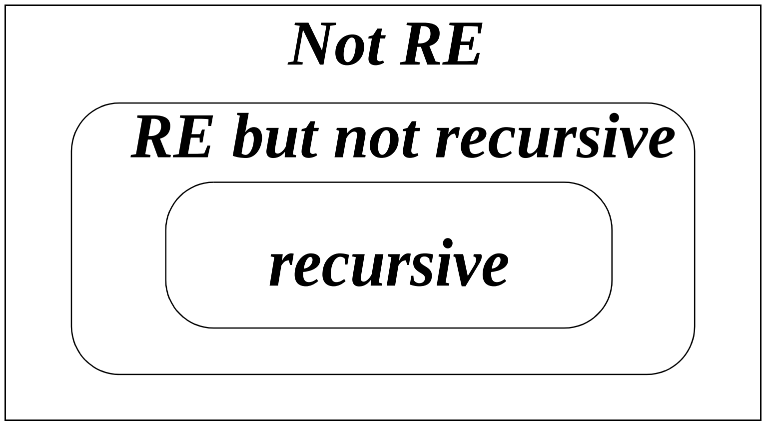
Problem P is called ***decidable****, if P is* ***recursive***

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

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

Three ***classes*** *of languages(problems)*

recursive	***decidable***	***countable***
RE *but* ***not recur.***	***undecidable***	***countable***
not RE	***undecidable***	***uncountable***



Recursive	***Decidable(algorithm)***
	total (recursive) function
Recursive enumerable	***Turing computable***
	partial recursive function
	programmable(computable)

Languages and problems

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

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)

complement *of L_d* $\overline{L_d} = \{w_i \in \Sigma^* \mid w_i \in L(M_i)\} = L_u$

universal language in

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

Complement of recursive and recursively enumerable languages

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

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

Consider $\overline{M}$

***accept** and halt → **not accept** and halt.*

***not accept** and halt → **accept** and halt.*

$\exists \overline{M}$.϶. $\overline{L} = L(\overline{M})$ *and always **halts**.*

More detail

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

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

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

no transition in M → transition to f

*Read "**Why 'Recursive'**?" in page 385*

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

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

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****.*

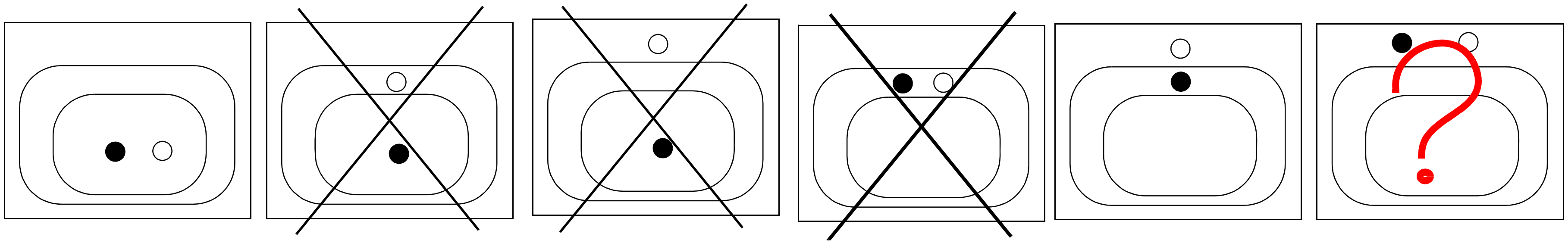
If $x \notin L \rightarrow M_2$ ***accept*** *and halt $\rightarrow$ M* ***not*** *accept x and* ***halt****.*

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

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

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

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



Universal language: L_u: complement of L_d.

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

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.*

$L_d = \overline{L}_u$ *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**.*

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

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

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

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

*A **problem** P is **decidable**,*

*if there exists a **decider** (**program**, **algorithm**, **TM**) that*

*<u>**always**</u> tells **yes** or **no** correctly.*

***Undecidable**, otherwise.*

***Assume** D is **countable**.*

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

*But **decider** is **countable**.*

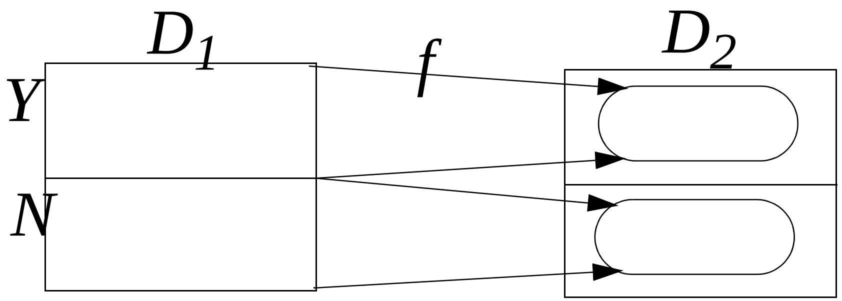
*$\therefore \exists P$, .∋. P is **undecidable**.*

Reducing one problem to another

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

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

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



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

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

P_2 *is* ***not easier*** *than* P_1*.*

Theorem 9.7 *If there is a* ***reduction*** *from* P_1 *to* $P_2 (P_1 \le P_2)$, *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) Supccccpose* 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 .э. $L(M_e) = L_e$.

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

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}.($L_u \le L_{ne}$)

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 accepts w* ($w \notin L(M)$), *then code for* $M \notin L_{ne}$.

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

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 $\{\emptyset\}$.*

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

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

*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****.*

Let P be a ***nontrivial*** *property of RE languages.*

1. *Assume $P(\emptyset)$ = false (or $\emptyset \notin P$).*

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

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

   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 accepts w, do nothing.*

2. *Assume $P(\emptyset)$ = true (or $\emptyset \in P$).*

   Consider complement property $\overline{P}$, $L_{\overline{P}} = \overline{L_P}$.

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

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

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

Three class of languages(problems)

1. *Not recursively enumerable*
   Not computable
2. *Recursively enumerable but not recursive*
   *Computable 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

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

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