I238 計算の理論

上原 隆平

2018年I-1期（4-5月）

I238 Computation Theory

by

Prof. Ryuhei Uehara

Term I-1, April-May, 2018

計算(量)の理論

- ゴール1:
  - “計算可能な*関数/問題/言語/集合*”
    - 関数には2種類存在する;
      1. 計算不能(!)な関数
      2. 計算可能な関数
- ゴール2:
  - 「問題の困難さ」を示す方法を学ぶ
    - 計算可能な問題であっても、*手におえない*場合がある！
      - 計算に必要な資源(時間・領域)が多すぎるとき

Computation Theory/ Computational Complexity

- Goal 1:
  - "*Computable Function/Problem/Language/Set*"
    - We have two functions;
      1. Functions that are not computable!
      2. Functions that are computable.
- Goal 2:
  - How can you show "*Difficulty of Problem*"
    - There are *intractable* problems even if they are computable!
      - because they require too many resources (time/space)!

3.マシンモデルと計算可能性

3.「計算」とは何か？

チューリングマシンの形式的定義：

チューリングマシンMが受理する文字列の集合をL(M)と書く.

– チューリングマシンMが言語Lを「認識する」

⇔L(M)=Lである.

[注意]チューリングマシンはいつでも停止するとは限らない．いつでも停止するチューリングマシンで認識できる言語だけを考えることもあるが，本講義ではここは区別しない．

今日はここの違いを真面目に考えるのがテーマ

3. Machine model & computability

3. Studies on what is a computation.

Formal definition of Turing machine :

We denote by L(M) the set of strings accepted by a Turing machine M.

– A Turing machine M recognizes a language L⇔L(M)=L.

[Note] In general, a Turing machine does not halt necessarily. We sometimes consider Turing machines that always halt for any inputs. We will not distinguish between these models in this class.

Today's Topic: We will consider this part seriously.

4. 計算不能性と対角線論法

4. 計算不能な問題

以下の問題を解くチューリングマシンは存在しない:

停止性判定問題HALT (停止するかどうかを決定する問題)
入力:チューリングマシン T と
それへの入力 x を符号化した文字列$<T,x>$
出力: T に入力 x を与えると，停止するか？
Yes: $T(x)$ は(有限時間内に)停止する
No: 停止しない(無限ループ)

正確に言えば，停止性判定問題を解くチューリングマシン U' は存在しない.

…証明は「対角線論法」を用いて行う

4. Undecidability and Diagonalization

4. Undecidable problem

The following problem cannot be solved by any Turing machine:

The problem HALT (Problem of deciding halting)

input : a code <*T*,*x*> of Turing machine *T* and an input *x*

output: *T* will terminates for the input *x*?

Yes: if *T*(*x*) terminates

No: otherwise.

Precisely, we can show that there is no Turing machine *U'* that computes the halting problem.

…Proof is done by "diagonalization" essentially…

4.計算不能性と対角線論法

4. 2.対角線論法

演習問題2: ここで通常の辞書式順序を使わないのは，なぜか？

定義:
集合が有限であるか、自然数と同じ濃度を持つとき、これを「可算」集合という.

例 1.3'.
0/1文字列の集合は長さ優先辞書式順序により可算集合:

ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, …

観測:
可算集合の部分集合は可算集合

例 1.3.
チューリングマシン（で計算できる関数）は、
各マシンが2進文字列で符号化できることから可算集合
(別の言い方をすれば $T_0, T_1, T_2, \ldots$ と列挙できる)

自然な疑問: 可算でない集合なんて存在するのだろうか？

4. Undecidability and Diagonalization

4. 2. Diagonalization

Exercise 2: Why we do not use the ordinary lexicographical ordering?

Definition:

A set is *countable* if it is finite or it has the same cardinality of natural numbers.

Ex. 1.3'.

The set of 0/1 strings is countable by the lex. ordering:

ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, …

Observation:
Any subset of a countable set is also countable

Ex. 1.3.

Turing machines (and corresponding functions) are countable because each machine can be represented by a binary string.
(In other words, they can be enumerated as T_0, T_1, T_2,…)

Natural question: Is there any uncountable set??

4.計算不能性と対角線論法

4. 3.対角線論法による計算不能性の証明

[定理] 停止性判定問題HALTは決定不能である.

[対角線論法による証明]

計算可能な(1入力)関数すべてからなる集合を Φ とする.
集合 Φ の各要素は一つのチューリングマシンに対応し、
それはΣ^*の2進文字列で表現される.
これらの2進文字列は辞書式順序で

$b_1, b_2, \ldots, b_k \ \ldots$

と列挙できる.
したがってΦのすべての関数は次のように列挙できる:

$f_1, f_2, \ldots, f_k, \ldots$

簡単にいえば Φ は可算集合!

4. Undecidability and Diagonalization

4. 3. Proof of undecidability via Diagonalization

[Theorem] The problem HALT is undecidable.

[Proof by diagonalization]

Let Φ be a set of all computable functions (with one argument) .

Each element in Φ corresponds to a Turing machine, that can be represented in a binary string in Σ^*.

Thus we can enumerate all corresponding binary strings as

$b_1, b_2, \ldots, b_k \ \ldots$

in the lexicographical order.

Thus, we can also enumerate all the functions in Φ:

$f_1, f_2, \ldots, f_k, \ldots$

In other words, the set Φ is a countable set!

結論: 停止性判定問題 HALT はコンピュータでは解けない.

[関数]の集合は非可算集合だが,
[計算できる関数]の集合は可算集合である.

対角線論法:

ある要素が可算無限集合に属さないことを示すための論法

ある可算無限集合 G が与えられたとき, その集合に属さない要素 g を構成する方法を与えている.

こうして構成した g は, 対角成分がつねに異なるため可算集合 G に属さない.

Our conclusion: The problem HALT is not computable.

The set of *functions* is uncountable, while
the set of *computable functions* is countable.

Diagonalization

Given a countable set G, construct an element g which does not belong to G.

2.5 計算不可能な関数の例

関数の性質についての述語は計算不可能になることが多い.

例2.19. 与えられたプログラムが計算する関数が恒等的に0か?

Zero(a): aはプログラムAのコードで，任意のxに対していつでもA(x)=0

Zeroは計算不可能.

まず，次のプログラム$Y_{(a,x)}(t)$は作成可能:

- 固定されたプログラムコードaとxに対して,
- A(x)がtステップで止まるかどうかを模倣して
  - 止まるなら1を出力
  - 止まらないなら0を出力

ここで,

Halt(a,x)="yes" ⇔ ∃t $Y_{(a,x)}(t)=1$ ⇔ Zero($y_{(a,x)}$)="no"

Halt(a,x)="no" ⇔ ∀t $Y_{(a,x)}(t)=0$ ⇔ Zero($y_{(a,x)}$)="yes"

である. よって,

もしZeroが計算可能なら，Halt(a,x)は次のように計算できる:

- a,xから$Y_{(a,x)}$のプログラムコード $y_{(a,x)}$ を構築する
- Zero($y_{(a,x)}$)を計算して，その逆を出力する.

「プログラムはいつか止まるか？」という問題は解けないが,
「プログラムはtステップ後に止まるか？」という問題なら解ける.

Haltは計算不可能だったので，Zeroも計算不可能になる.

2.5 Examples of incomputable functions

Predicates concerning on properties of functions are often incomputable.

Ex.2.19. Does a given program always output 0?

Zero(a): a is a code of program A, and A(x)=0 for any x.

Zero is incomputable.

The following program $Y_{(a,x)}(t)$ exists:

- For any fixed program code a and x,
- Simulate A(x) t steps, and
  - Output 1 if it halts
  - Output 0 if it does not halt

Now, we have;

$\text{Halt}(a,x)=\text{“yes”} \Leftrightarrow \exists t\ Y_{(a,x)}(t)=1 \Leftrightarrow \text{Zero}(y_{(a,x)})=\text{“no”}$

$\text{Halt}(a,x)=\text{“no”} \Leftrightarrow \forall t\ Y_{(a,x)}(t)=0 \Leftrightarrow \text{Zero}(y_{(a,x)})=\text{“yes”}$

Therefore, if Zero is computable, Halt(a,x) is also computable as follows:

- Construct a program code $y_{(a,x)}$ of $Y_{(a,x)}$ from a,x, and
- Output the opposite of $\text{Zero}(y_{(a,x)})$.

The problem “does a given program halts?” cannot be solved, while the problem “does a given program halts after t steps?” is solvable.

Since Halt is incomputable, Zero is also incomputable.

例2.20 与えられたプログラムは全域的か?

Total(a): a はプログラムAのコードで，すべてのxについてA(x)≠⊥

プログラムAに対して次の作業を考える.
(1)その中のhalt文を探し出す．→halt(*y*)と仮定
(2)「y≠0なら無限ループ，それ以外なら0を出力して停止」と書き換える.
(3)以上のことをすべてのhalt文について行う.
出来上がったプログラムをBとする.
　　プログラムAが0以外を出力すると必ず無限ループに陥る.
　　i.e., Aが常に0を出力しない限り，Bは全域的にならない.
上記の変換は計算可能➔次の関数が計算可能
　　replace(*a*) = *b*, *a*がプログラムコードのとき,
　　　　= *a*, その他のとき.
　　ただし，*b*はプログラムBのコード

一方、$\forall a \in \Sigma^*$[Zero(*a*)⇔Total(replace(*a*))]

よって，Totalが計算可能なら，Zeroも計算可能
　　i.e., Totalは計算不可能

Ex. 2.20 Is a given program total?

Total(a): a is a code of program A, and A(x)≠⊥ for any x.

Given a program A, consider the following computation.
(1) Find all halt statements.→assume that it is halt(*y*).
(2) Rewrite as [if y≠0 then loop, otherwise output y and halt].
(3) For each halt statement, do the above.
Let B be the resulting program.

If the program A outputs other than 0 then it enters infinite loop.
i.e., unless A always outputs 0, B is not total.

The above conversion is computable➔So is the following function

replace(*a*) = *b*, if *a* is a program code,
= *a*, otherwise,
where *b* is a code of the converted program B above.

On the other hand, $\forall a \in \Sigma^*$[Zero(*a*)⇔Total(replace(*a*))]

Therefore, if Total is computable, then Zero is also computable, a contradiction. Total is incomputable.

3. 計算可能性の分析

3.1. 関数から集合へ

関数の難しさ→集合の難しさ

⇧

構造的解析

集合$L \subseteq \Sigma^*$の認識問題または決定問題(recognition/decision problem)
⇔与えられた文字列 x が L に属するかどうかを判定

Lの特徴述語 $R_L(x) \leftrightarrow x \in L$

例3.1: EVEN = {<n> : n は偶数}, EQ = {<a, b>: $a=b$}

・EQの認識問題:

「与えられた文字列が<a,b>という形をしていて，かつ$a=b$か？」

「2つの文字列は等しいか？」

正式には，$\mathrm{Eq}(x) \longleftrightarrow \exists a,b[x=<a,b> \wedge\ a=b]$

直観的には，$\mathrm{Eq}'(a, b) \longleftrightarrow [a=b]$

EqとEq'の難しさには殆ど差がないので，同じと思ってよい.

3. Analysis of Computability

3.1. From Functions to Sets

Hardness of a function → hardness of a set

⇧

structural analysis

Problem of recognizing a set（or decision problem）

$L \subseteq \Sigma^*$, recognition problem of a set L

Given a string x, decide whether x belongs to L or not.

Characteristic predicate of L $RL(x) \leftrightarrow x \in L$

Ex.3.1 EVEN = {<n> : n is even}, EQ = {<a, b>: $a=b$}

- Recognition of EQ: “Given a string, is it of the form <a,b> and $a=b$?” or “Are two strings equal to each other?”

Formally, $Eq(x) \longleftrightarrow \exists a,b[x=<a,b> \wedge a=b]$

Intuitively $Eq'(a, b) \longleftrightarrow [a=b]$

There is little difference in hardness between Eq and Eq', so they are considered the same.

以下では,

集合の認識問題の難しさ → 集合の難しさ

問題＝関数の計算問題
問題＝Σ^*上の関数の計算問題
問題＝Σ^*上の集合の認識問題

Yes/Noタイプのプログラムが存在する

定義3.1. Σ^*上の集合は，その特徴述語が計算可能であるとき帰納的(recursive)であるという.

例3.2. HALT≡{<*a*,*x*>: Halt(*a*, *x*)}

HALTが帰納的 ⟷ Haltが計算可能

よって，HALTは帰納的でない.

帰納的集合
計算可能集合
認識可能集合
決定可能集合

Hereafter,

hardness of recognition problem of a set → hardness of a set

Problem = Problem of computing a function

Problem = Problem of computing a function on Σ^*

Problem = Problem of recognizing a set on Σ^*

Def. 3.1: A set *S* is ***recursive*** if its characteristic predicate is computable.

Ex. 3.2. HALT $\equiv$ {<*a*,*x*>: Halt(*a*, *x*)}

HALT is recursive $\longleftrightarrow$ Halt is computable

Thus, HALT is not recursive.

2通りの問題記述方法

文字列等価性判定問題(EQ)	与えられたxに対し
入力：Σ^*上の文字列の組$\langle a,b\rangle$	“$x \in$ EQ?”を判定する問題
質問：$a=b$?	ただし，EQ=$\{\langle a,b\rangle : a=b\}$
直観的	正式

認識プログラム＝ Σ^*型の入力変数をもつ1入力のプログラムで
どんな入力に対しても1または0を出力するもの.

認識プログラムAに対して,

Aが入力xを受理(accept) $\Leftrightarrow$ $A(x)$が1を出力する
記法：$A(x)$ = accept

Aが入力xを却下(reject) $\Leftrightarrow$ $A(x)$が0を出力する
記法：$A(x)$ = reject

Aが集合Lを認識（recognize) $\Leftrightarrow$ $L=\{x: A(x)=\text{accept}\}$

Lが帰納的 $\Leftrightarrow$ **Lを認識するプログラムがある**

Two different ways of problem descriptions

String equivalence(EQ)	Given an x, determine whether
Input : pair $<a, b>$ of strings on Σ^*	"$x \in$ EQ?"
Question : $a=b$?	where, EQ=$\{<a,b>: a=b\}$
Intuitive	Formal

Recognition program = a program of one input on Σ^*
which outputs 1 or 0 for any input.

For a recognition problem A

A accepts an input $x \Leftrightarrow A(x)$ outputs 1
notation : $A(x)$ = accept

A rejects an input $x \Leftrightarrow A(x)$ outputs 0
notation: $A(x)$ = reject

A recognizes a set $L \Leftrightarrow L=\{x: A(x)=\text{accept}\}$

L is ***recursive*** $\Leftrightarrow$ **There is a program recognizing *L***

3.2. 枚挙可能集合

Yes/Noタイプのプログラム

帰納的でない集合を認識するプログラムは存在しない.
しかし弱い意味での“認識”を考えると話は別

プログラムAが集合Lを半認識する
すべての $x \in \Sigma^*$で

$x \in L \quad \leftrightarrow \quad$ A(x)=accept
$x \notin L \quad \leftrightarrow \quad$ A(x)= $\perp$ (A(x)が停止しない)

集合Lは半帰納的 $\leftrightarrow$ 集合Lを半認識するプログラムが存在

帰納的集合 $\subsetneq$ 半帰納的集合

i.e., 認識可能な集合 $\subsetneq$ 半認識可能な集合

3.2 Enumerable set

There is no program for recognizing a non-recursive set,
but we have a different story if we consider weak "recognition"

Program A ***semi-recognizes*** a set *L*

for every $x \in \Sigma^*$

$x \in L \quad \leftrightarrow \quad$ A(*x*)=accept

$x \notin L \quad \leftrightarrow \quad$ A(*x*)= $\perp$ (A(*x*) does not stop)

A set *L* is semi-recursive $\leftrightarrow$ semi-recognizing program of a set *L*

Recursive sets $\subsetneq$ semi-recursive sets

i.e., recognizable sets $\subsetneq$ semi-recognizable sets

例3.5. Haltは帰納的ではないが，半帰納的

（方針：プログラムとそれへの入力が与えられたとき，その
停止性を調べるために実際に実行してみる．
停止する場合には有限時間内に判明する．）

```
prog HALT(input <a, x>);
var t: num;
begin
  t:=0;
  while true do
     if HaltInTime(a, x, t) then accept end-if;
     t:=t+1;
  end-ehile
end.
```

文字列 a が表すプログラムに x を入力すると
t ステップ以内に停止する

上記のプログラムはHALTを半認識する．

もし Halt(a,x) なら、あるステップ数 t が存在して、
HALT(<a,x>) は HaltInTime(a,x,t)を実行した時点で accept.

Ex.3.5. Halt is not recursive but it is semi-recursive

(Strategy : given a program and an input to it. Execute it to
determine whether its stops or not.
If it stops, we know it within finite time.)

```
prog HALT(input <a, x>);
var t: num;
begin
  t:=0;
  while true do
     if HaltInTime(a, x, t) then accept end-if;
     t:=t+1;
  end-ehile
end.
```

when *x* is input to a program represented by a string *a*, it halts within *t* steps.

This program semi-recognizes HALT,
since HALT(<a,x>) will accept for some t
when the program execute HaltInTime(a,x,t) if HALT(*a,x*).

[記法]

RANGE(*g*): 関数 *g* の値域．関数 *g* を計算するプログラムの出力の集合

定理3.2. 集合*L*を空でない任意の集合とする．このとき，次の2条件は同値：
(a) *L*は半帰納的
(b) *L*= *RANGE*(*g*)となるような計算可能関数 *g* が存在する．

(a)→(b)の証明

*L*は半帰納的 ➡*L*を半認識するプログラム A が存在

c_1: *L*の任意の要素
(L≠Φより存在)

A, c_1 より、プログラムGを構成

```
prog G(input w: Σ*): Σ*;
var x: Σ* ; t: num;
begin
   if w ∉ Σ*x N then halt(c1) end-if;
   x:=1st(w); t:=2nd(w);
   if  HaltInTime(<A>, x, t) then halt(x)
                             else halt(c1) end-if
end.
```

プログラムGが計算する関数を *g* とし，*g* が(b)を満たすことを示す．

[Notation]
RANGE(*g*): range of a function *g*, i.e.,
set of all outputs of a program computing a function *g*

Theorem3.2 Let *L* be an arbitrary non-empty set. Then, the following two conditions are equivalent:
(a) *L* is semi-recursive.
(b) There is a computable function *g* such that *L*= *RANGE*(*g*).

Proof: (a)→(b)

L is semi-recursive ➡ a program A that semi-recognizes *L* exists
c_1: any element of *L*

```
prog G(input w: Σ*): Σ*;
var x: Σ* ; t: num;
begin
   if w∉ Σ*x N then halt(c1) end-if;
   x:=1st(w); t:=2nd(w);
   if HaltIn Time(<A>, x, t) then halt(x) else halt(c1) end-if
end.
```

Let *g* be a function computed by this program G.
We prove that *g* satisfies (b) as follows:

プログラムG: 入力<x,t> に対して HaltInTime(<A>, x, t)ならhalt(x)、
それ以外なら halt(c_1) を実行する

(1) g は計算可能で全域的

(2) すべての $x \in L$ は A で受理されるから

$L(x)$=accept $\rightarrow \exists t \in \mathrm{N}$ [HaltInTime(<A>, x, t)]

$\rightarrow \exists t \in \mathrm{N}$ [Gは<x,t>に対して x を出力]

よって、Lのすべての要素は G の出力として現れる。

つまり $L \subseteq RANGE(g)$.

(3) 一方, どんな $y \notin L$ に対してもAは停止しないから,

$L(y) = \perp \rightarrow \forall \mathrm{t} \in \mathrm{N}$ [¬HaltInTime(<A>, y, t)]

$\rightarrow \forall \mathrm{t} \in \mathrm{N}$ [G は入力<y, t>に対して c_1 を出力]

つまり, $\overline{L}$のどの要素 y もGの出力として現れない. よって

$$\overline{L} \subseteq \overline{RANGE(g)} \text{ つまり } RANGE(g) \subseteq L$$

$\Rightarrow$(2),(3)より $L = RANGE(g)$

- g is computable and total
- Since any $x \in L$ is accepted by A,

$L(x)$ = accept $\rightarrow \exists t \in N$ [HaltInTime(<A>, x, t);

$\rightarrow \exists t \in N$ [G outputs x for an input <x, t>]

$\rightarrow \exists w (=<x, t>) \in \Sigma^*$ [g(w) = x]

that is, $L \subseteq RANGE(g)$

every element of L appears as an output of G.

- On the other hand, since L does not halt for any $y \notin L$

$L(y) = \perp \rightarrow \forall t \in N$ [$\neg$HaltInTime(<A>, y, t)]

$\rightarrow \forall t \in N$ [G outputs c_1 for an input <y, t>]

$\rightarrow \forall y' \in \Sigma^*, \forall t \in N$ [g(<y', t>) $\neq y$]

$y \notin L$, $c_1 \in L$ thus $y \neq c_1$

$\rightarrow \forall w \in \Sigma^*$ [$g(w) \neq y$]

that is, no element y of $\overline{L}$ appears as an output of G.

$\overline{L} \subseteq \overline{RANGE(g)}$

$L \subseteq RANGE(g) \wedge \overline{L} \subseteq \overline{RANGE(g)}$

$L = RANGE(g)$

(b)→(a)の証明: $L = RANGE(g)$ となるような計算可能関数 g が存在するなら L は半帰納的

g は計算可能⟹g を計算するプログラム G が存在
G を用いて次のプログラムBを作る.

```
prog B(input x);
var w: Σ* ;
begin
   w:=ε ;
   while true do
      if G(w) = x then accept end-if;
      w:=next(w)
   end-while
end.
```

next は長さ優先辞書式順序で次の元を求める関数

- Σ^* の元をすべて列挙して調べている.
- G(w)=xとなる $w \in \Sigma^*$ が存在すれば, $B(x)$=accept
  (存在しなければ停止しない)

よってプログラムBはLを半認識する.

（証明終）

Proof: (b)→(a)

i.e., there is a computable function g such that L = RANGE(g)

→ L is semi-recursive

g is computable ⇒ there is a program G that computes g.

Using this, we have the following program B.

```
prog B(input x);
var w:  ;
begin
 ε w:=    ;
   while true do
      if G(w) = x then accept end-if;
      w:=next(w)
   end-while
end.
```

next is a function that computes the next element in the pseudo-lexicographic order

all the elements of Σ^* are checked in order.

if there is a $w \in \Sigma^*$ such that G(w)=x, then $x \in L$.

The above program semi-recognizes L.

（Q.E.D.）

定理3.3. 任意の無限集合Lに対し，次の2条件は同値．
(a) Lは半帰納的
(b) $L=RANGE(e)$となるような計算可能で1対1の関数 e が存在する．

定理3.3の証明は省略

cf

定理3.2. 集合Lを空でない任意の集合とする．このとき，次の2条件は同値．
(a) Lは半帰納的
(b) $L=RANGE(g)$となるような計算可能関数 g が存在する．

Theorem3.3. For any infinite set L, the following two conditions are equivalent:

(a) L is semi-recursive.

(b) There is a computable one-to-one function e such that $L=RANGE(e)$.

Proof of Theorem 3.3 is omitted.

cf

Theorem3.2 Let L be an arbitrary non-empty set. Then, the following two conditions are equivalent:

(a) L is semi-recursive.

(b) There is a computable function g such that $L=RANGE(g)$.

定理3.3

➔ 半帰納的集合 L には

$L=\{e(\varepsilon), e(0), e(1), e(00), e(01), e(10), e(11), e(000), \ldots\}$

となるような1対1の計算可能関数 e が存在する.

関数 e は L を枚挙 (enumerate)する

定義3.2. 集合 L は次のいずれかが成り立つとき, (帰納的に)
枚挙可能であるという(recursively enumerable).
(a) L は有限集合
(b) L を枚挙する関数で計算可能なものが存在.

注: 有限集合 L に対しては $L = RANGE(e)$となるような
1対1の全域関数 e などあり得ないので, 例外的に扱っている.

定理3.4 すべての集合 L に対し,
L が半帰納的 ⟷ Lが枚挙可能

Theorem3.3

➔ for a semi-recursive set L there exists a computable one-to-one function such that

$L=\{e(\varepsilon), e(0), e(1), e(00), e(01), e(10), e(11), e(000), ...\}$

We say the function e ***enumerates*** L.

Def.3.2 A set L is **(recursively) enumerable** if

(a) L is a finite set, or

(b) there is a computable function that enumerates L.

Remark : Finite sets are exceptional, since for any finite set L there is no total on-to-one function e such that L = RANGE(e).

Theorem3.4 For any set L we have

L is semi-recursive $\Longleftrightarrow$ L is enumerable

枚挙可能性と帰納性の比較

A: 帰納的集合

- ✓ Aの特徴述語 $R_A(x)$ が計算可能.
- ✓ $x \in \Sigma^*$に対し、$x \in A$かどうか判定可能
- ✓どんな入力 $x \in \Sigma^*$に対しても、
  いつも停止してYes/Noを答えてくれるプログラムが存在

B: 枚挙可能集合

- ✓ B を枚挙する関数が計算可能
- ✓すべてのBの要素を順番に出力するプログラムが存在

Comparison between enumerability and recursiveness

A: recursive set ⟹ the characteristic predicate $R_A(x)$ is computable

That is, for $x \in \Sigma^*$ it is computable whether $x \in A$

B: enumerable set ⟹ a function that enumerates B is computable

that is, we can enumerate all the elements of B

定理3.5. すべての集合 L に対し，次の条件は同値
(a) L は枚挙可能.
(b) 適当な計算可能述語 R に対し，$L=\{x:\exists\, w\in \Sigma^* [R(x, w)]\}$

(a)→(b)の証明
L は枚挙可能だから，L を枚挙する計算可能関数 e が存在する.
$R(x,w) \equiv [e(w) = x]$と定義
e が L の枚挙関数なので，
$L = \{x: \exists w \in \Sigma^* [e(w) = x]\}$
$= \{x: \exists w \in \Sigma^* [R(x,w)]\}$
e は計算可能関数 ⟶ e を計算するプログラムが存在
しかも e は全域的なので，そのプログラムは必ず停止して答を出力
よって，述語 R は計算可能

Theorem3.5. For any set L, the following conditions are equivalent.

(a) L is enumerable.

(b) For some computable predicate R, we have

$$L=\{x: \exists w \in \Sigma^* [R(x, w)]\}$$

Proof : (a)→(b)

L is enumerable, so there is a computable function e enumerating L.

Define $R(x,w) \equiv [e(w) = x]$

Since e is a function enumerating L,

$$L = \{x: \exists w \in \Sigma^* [e(w) = x]\}$$
$$= \{x: \exists w \in \Sigma^* [R(x,w)]\}$$

e is computable ⟶ there is a program that computes e

Moreover, e is total, and thus the program always stops and outputs an answer. Thus, the predicate R is computable.

定理3.5. すべての集合 L に対し，次の条件は同値

(a) L は枚挙可能.

(b) 適当な計算可能述語 R に対し，$L=\{x: \exists w \in \Sigma^* [R(x, w)]\}$

(b)→(a)の証明

条件(b)を満たす述語を計算する関数 R(x,w) を使って，L を半認識するプログラムCが作れる.

```
prog C(input x);
var w: Σ*;
begin
  w:= ε;
  while true do
    if R(x, w) then accept end-if;
    w:=next(w)
  end-while
end.
```

したがって，L は半帰納的，つまり枚挙可能.

証明終

Theorem3.5 For any set L, the following conditions are equivalent.

(a) L is enumerable.

(b) For some computable predicate R, $L=\{x: \exists w \in \Sigma^* [R(x, w)]\}$

Proof: (b)→(a)

Using a program that computes a predicate satisfying the condition (b), we have a program that semi-recognizes L.

```
prog C(input x);
var w: Σ*;
begin
  w:= ε;
  while true do
    if R(x, w) then accept end-if;
    w:=next(w)
  end-while
end.
```

Therefore, L is semi-recursive. That is, it is enumerable.

Q.E.D.

どんな枚挙可能集合 L にも次の関係を満たす計算可能な
述語Rが存在

「すべての $x \in \Sigma^*$に対し，$x \in L \Longleftrightarrow \exists w \in \Sigma^*[R(x, w)]$」

Lの認識問題を $\exists w[R(x, w)]$ という形の論理式で判定可能.
逆に，そのような形で認識問題を判定できる集合が枚挙可能集合.

$\exists w[Q(x, w)]$ という形の論理式： 枚挙可能集合のための論理式
（RE論理式）

QをこのRE論理式の核(kernel)という.

LのRE論理式： 枚挙可能集合Lに対するRE論理式

LのRE論理式が $\exists w[R(x,w)]$ のとき，
各$x \in L$に対し，$R(x, w_x)$となるような$w_x \in \Sigma^*$ が存在する.
このw_xを‘$x \in L$’の証拠（witness)と呼ぶ.

For any enumerable set L there is a computable predicate R satisfying

"for any $x \in \Sigma^*$, we have $x \in L \Longleftrightarrow \exists w \in \Sigma^* [R(x, w)]$.

The problem of recognizing L can be determined by the predicate of the form $\exists w[R(x, w)]$.

Conversely, sets whose recognition problem can be determined in this way are enumerable sets.

predicate of the form $\exists w[Q(x, w)]$: predicate for enumerable sets (RE predicate)

Q is a kernel of the RE predicate.

RE predicate for L: the RE predicate for an enumerable set L

If the RE predicate of L is $\exists w[R(x,w)]$,

for each $x \in L$ there is $w_x \in \Sigma^*$ such that $R(x, w_x)$ is true.

Such w_x is called a **witness** for '$x \in L$'

3.3. クラスRECとクラスRE

クラスREC $\equiv \{L: L$ は帰納的$\}$: 帰納的集合のクラス

- クラスRECの外側は帰納的でない集合の領域
- 空でないこと程度しか分かっていない（ここまでの議論では）

HALT $\notin$ クラスREC

目標: REC の外側の領域の構造の解析

RECの外側で最も扱いやすい集合のクラスは何か？

$\longrightarrow$ 枚挙可能集合.

3.3 Class REC and Class RE

Class REC $\equiv$ {L: L is recursive}: a class of recursive sets

- Outside of the class REC is a region for non-recursive sets.
  It is only known that it is not empty (by the argument so far).

HALT $\notin$ class REC

GOAL: Analyzing the structure outside REC

What is the easiest class of sets outside REC?

$\longrightarrow$ enumerable sets.

RE ≡{L: L は枚挙可能}
co-RE≡ {L: L が枚挙可能}

Memo: $\overline{\mathrm{RE}}$ は、
より広いクラスを含むので、
REより難しいと言える。

注：L：集合
Lが枚挙可能⟺L が半帰納的
⟺ L を半認識するプログラムAが存在.
$x \in S^*$, $x \in L \Longleftrightarrow A(x) = \text{accept}$
$x \notin L \Longleftrightarrow A(x) = \perp$

クラスco-REはクラス$\overline{\text{RE}}$の補クラスREではないことに注意.

例3.8. クラスRE, co-REに入る集合の例.

HALT ∈ RE, $\overline{\text{HALT}} \in$ co-RE

Memo: $\overline{\mathrm{RE}}$ is harder than PE since it contains more wide class.

$\mathrm{PE} \equiv \{L: L \text{ is enumerable}\}$

$\text{co-PE} \equiv \{L: \overline{L} \text{ is enumerable}\}$

Note : L : set

L is enumerable $\Leftrightarrow$ L is semi-recursive

$\Leftrightarrow$ there is a program A that semi-recognizes L.

$x \in \Sigma^*$, $x \in L \Leftrightarrow A(x) = \text{accept}$

$x \notin L \Leftrightarrow A(x) = \perp$

Note that the class co-RE is not complementary of the class RE.

Ex.3.8. Examples of sets belonging to class RE and class co-RE.

$\mathrm{HALT} \in \mathrm{RE}$, $\overline{\mathrm{HALT}} \in \text{co-RE}$

REとco-REは同程度の“難しさ”

A：任意のRE集合

$x \in S^*[(x \in A \rightarrow X(x)=\text{accept}) \wedge (x \notin A \rightarrow X(x)=\perp)]$

となるプログラムXが作れる

B：任意のco-RE集合

$x \in S^*[(x \in B \rightarrow X(x)=\perp) \wedge (x \notin B \rightarrow X(x)=\text{accept})]$

となるプログラムXが作れる

上記の2つのプログラムはよく似ており，難しさに差がつけられない．

RE and co-RE are equally “hard”

A : arbitrary RE set

we can write a program X such that

$x \in S^*[(x \in A \rightarrow X(x)=\text{accept}) \wedge (x \notin A \rightarrow X(x)=\perp)]$

B : arbitrary co-RE set

we can write a program X such that

$x \in S^*[(x \in B \rightarrow X(x)=\perp) \wedge (x \notin B \rightarrow X(x)=\text{accept})]$

The above two programs are similar, and there is no difference.

定理3.6. すべての集合Lに対し，次の関係が成り立つ．

(1) $L \in \mathrm{REC} \leftrightarrow \overline{L} \in \mathrm{REC}$

(2) $L \in \mathrm{RE} \leftrightarrow \overline{L} \in \text{co-RE}$

証明：

(1) $L \in \mathrm{REC}$とすると，L を認識するプログラムがある．

accept →reject，reject → accept

と変更すると，$\overline{L}$を認識するプログラムを得る．

よって，$\overline{L} \in \mathrm{REC}$

(2)はco-REの定義より明らか．

証明終

Theorem 3.6. For every set L, the followings hold:

(1) $L \in$ REC $\leftrightarrow$ $\overline{L} \in$ REC

(2) $L \in$ RE $\leftrightarrow$ $\overline{L} \in$ co-RE

Proof:

(1) $L \in$ REC, then there is a program that recognizes L.

If we exchange accept with reject

accept → reject, reject → accept

then, the resulting program recognizes $\overline{L}$.

So, $\overline{L} \in$ REC

(2) is obvious from the definition of co-RE.

Q.E.D.

定理3.7. (1) REC $\subsetneq$ RE (2) REC $\subsetneq$ co-RE

証明: 略

Theorem 3.7. (1) REC $\subsetneq$ RE (2) REC $\subsetneq$ co-RE

Proof : Omitted.

定理3.8. REC＝RE∩co-RE

証明：

定理3.7より，REC⊆RE∩co-RE

任意の $L \in$ RE $\cap$ co-REについて，$L \in$ RECを示したい．

仮定より，$L \in$ RE かつ $\overline{L} \in$ RE

→L を半認識するプログラムA_1と

$\overline{L}$ を半認識するプログラムA_2が存在．

このとき，次のプログラムBは L を認識する．

```
prog B(input x);
var t: num;
begin
   for t:=0 to ∞ do
      if HaltInTime(<A1>, x, t) then accept end-if;
      if HaltInTime(<A2>, x, t) then reject end-if
   end-for
end.
```

$x \in L$のとき，
A_1が先に停止して
acceptとなる．
$x \notin L$のとき，
A_2が先に停止して
rejectとなる．

証明終

Theorem 3.8 REC = RE ∩ co-RE

Proof:

By Theorem 2,7 we have REC ⊆ RE ∩ co-RE

We want to show that L ∈ REC for any $L \in$ RE ∩ co-RE.

By the assumption, $L \in$ RE and $\overline{L} \in$ RE

→there are a program A_1 that semi-recognizes L and
a program A_2 that semi-recognizes $\overline{L}$.

Then, the following program B recognizes L.

if $x \in L$,
A_1 stops before A_2
and accepts x.
if $x \notin L$,
A_2 stops before A_1
and rejects x.

```
prog B(input x);
var t: num;
begin
   for t:=0 to ∞ do
      if HaltInTime(<A1>, x, t) then accept end-if;
      if HaltInTime(<A2>, x, t) then reject end-if
   end-for
end.
```

Q.E.D.

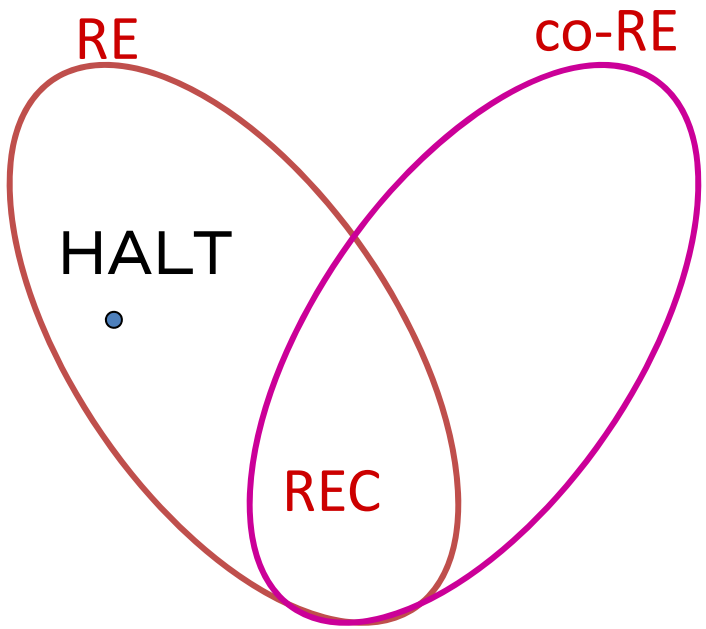
定理3.9. RE ≠ co-RE

証明:

RE=co-REと仮定すると，RE=RE ∩ co-RE

定理3.8より，REC＝REとなり，定理3.7に矛盾．

証明終



Theorem 3.9. RE $\neq$ co-RE

Proof:

If we assume RE=co-RE, we have RE=RE $\cap$ co-RE.

Hence, by Theorem 3.8 we have REC=RE, contradicts to Theorem 3.7.

Q.E.D.