

1238 計算の理論

上原 隆平

2019年I-1期(4-5月)

短期の予定 Schedule for short term

5月7日: レポート1出題(締切: 5月14日10:50)

(Report 1; deadline is 10:50 on May 14)

5月9日: 午後, 補講あり

(Supplemental lecture on Tutorial Hour)

5月14日: 中間テスト (Mid-term Examination)

- 出題範囲(Scope): 今日の講義の範囲まで
- 持ち込んで良いもの(You can/should bring)
 - 筆記用具(Pen/Pencil)
 - 1. スライドのコピー(書き込み可) + 手書きノート(Copies of slides + hand-written notes)
 - 2. 手書きノート(hand-written notes)
 - 3. A4用紙1枚(回収します)(A sheet of paper of A4 size)

I238 Computation Theory

by

Prof. Ryuhei Uehara

Term I-1, April-May, 2019

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)が多すぎる時

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」

$\Leftrightarrow L(M)=L$ 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 $L \Leftrightarrow L(M)=L$.

[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 $\langle T, x \rangle$

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 $\langle T, x \rangle$ of Turing machine T and an input x

output: T will terminate 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. 3. 対角線論法による計算不能性の証明

[定理4.1] 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

$$b_1, b_2, \dots, b_k \dots$$

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:

$$f_1, f_2, \dots, f_k, \dots$$

簡単にいえば Φ は可算集合!

4. Undecidability and Diagonalization

4. 3. Proof of undecidability via Diagonalization

[Theorem 4.1] 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

$$b_1, b_2, \dots, b_k \dots$$

in the lexicographical order.

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

$$f_1, f_2, \dots, f_k, \dots$$



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 .

5. 計算可能性と枚挙可能性

5.0: 準備

TM A を符号化した a とそれへの入力 x

[定理4.1] 停止性判定問題 $\text{HALT}(a, x)$ は決定不能である。

[定理5.1] 次の停止性判定問題の亜種 $\text{HaltInTime}(a, x, t)$ は決定可能である。

$\text{HaltInTime}(a, x, t)$:

TM A に入力 x を与えると t ステップ以内に停止するか？

[証明]

万能TMを使って、TM A を入力 x の元で t ステップ模倣すればよい。

「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。

5. Computability and Enumerability

5. 0: Preliminary

TM A is encoded in a, and its input x

[Theorem 4.1] The problem $\text{HALT}(a, x)$ is uncomputable.

[Theorem 5.1] The following variant of the halting problem
 $\text{HaltInTime}(a, x, t)$ is computable.

$\text{HaltInTime}(a, x, t)$:

Does Turing machine A with input x will halt in t steps?

[Proof]

We can simulate TM A with input x in t steps using the universal Turing machine.

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

5.1. 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5.1. 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:

HaltInTime(a, x, t) と同様

- 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,

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

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

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
- $\text{Zero}(y_{(a,x)})$ を計算して, その逆を出力する.

Haltは計算不可能だったので, Zeroも計算不可能になる.

5.1. Examples of incomputable functions

Predicates concerning on properties of functions are often incomputable.

Ex. 5.1. 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 $\text{Zero}(y_{(a,x)})$.

As same as **HaltInTime**(a, x, t)

Since Halt is incomputable, Zero is also incomputable.

例5.2. 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 $A(x) \neq \perp$

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. \rightarrow halt(y)と仮定
- (2) 「 $y \neq 0$ 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 \rightarrow 次の関数が計算可能

$\text{replace}(a) = b$, 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、 $\forall a \in \Sigma^* [\text{Zero}(a) \Leftrightarrow \text{Total}(\text{replace}(a))]$

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. 5.2. Is a given program total?

Total(a): a is a code of program A , and $A(x) \neq \perp$ for any x .

Given a program A , consider the following computation.

- (1) Find all halt statements. \rightarrow assume that it is halt(y).
- (2) Rewrite as [if $y \neq 0$ 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 \rightarrow 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, $\forall a \in \Sigma^* [Zero(a) \Leftrightarrow Total(\text{replace}(a))]$

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

5.2. 計算可能性の分析

5.2.1. 関数から集合へ

関数の難しさ \rightarrow 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)

\Leftrightarrow 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5.3. $\text{EVEN} = \{ \langle n \rangle : n \text{ は偶数} \}$, $\text{EQ} = \{ \langle a, b \rangle : a=b \}$

・ EQ の認識問題:

「与えられた文字列が $\langle a, b \rangle$ 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, $\text{Eq}(x) \leftrightarrow \exists a, b [x = \langle a, b \rangle \wedge a=b]$

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

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.

5.2. Analysis of Computability

5.2.1. From Functions to Sets

Hardness of a function \rightarrow 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.5.3. **EVEN** = $\{ \langle n \rangle : n \text{ is even} \}$, **EQ** = $\{ \langle a, b \rangle : a=b \}$

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

Formally, $Eq(x) \leftrightarrow \exists a, b [x = \langle a, b \rangle \wedge a=b]$

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

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5.1. 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5.4. $\text{HALT} \equiv \{ \langle a, x \rangle : \text{Halt}(a, x) \}$

HALTが帰納的 \leftrightarrow 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 \rightarrow 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. 5.1. A set S is *recursive* if its characteristic predicate is computable.

Ex. 5.4. $\text{HALT} \equiv \{ \langle a, x \rangle : \text{Halt}(a, x) \}$

HALT is recursive \leftrightarrow 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)

入力: Σ^* 上の文字列の組 $\langle a, b \rangle$

質問: $a=b?$

直観的

与えられた x に対し

“ $x \in EQ?$ ”を判定する問題

ただし, $EQ = \{\langle a, b \rangle : a=b\}$

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) $\iff A(x)$ が1を出力する
記法: $A(x) = \text{accept}$

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

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

L が帰納的 $\iff L$ を認識するプログラムがある

Two different ways of problem descriptions

String equivalence(EQ)

Input: pair $\langle a, b \rangle$ of strings on Σ^*

Question: $a=b?$

Intuitive

Given an x , determine whether

" $x \in \text{EQ}?$ "

where, $\text{EQ}=\{\langle a,b \rangle : a=b\}$

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 $x \iff A(x)$ outputs 1

notation: $A(x) = \text{accept}$

A rejects an input $x \iff A(x)$ outputs 0

notation: $A(x) = \text{reject}$

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

L is **recursive** \iff **There is a program recognizing L**

5.3. 枚挙可能集合

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

$$x \in L \leftrightarrow A(x) = \text{accept}$$

$$x \notin L \leftrightarrow A(x) = \perp \quad (A(x) \text{ が停止しない})$$

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

5.3. 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 $x \in \Sigma^*$

$x \in L \iff A(x) = \text{accept}$

$x \notin L \iff A(x) = \perp \quad (A(x) \text{ does not stop})$

A set L is semi-recursive \iff semi-recognizing program of a set L

Recursive sets \subsetneq semi-recursive sets

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

例5.5. 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を与えられたとき, 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 $\text{Halt}(a,x)$ なら、あるステップ数 t が存在して、
 $\text{HALT}(\langle a,x \rangle)$ は $\text{HaltInTime}(a,x,t)$ を実行した時点で accept.

Ex.5.5. Halt is not recursive but it is semi-recursive

(Strategy : given a program and an input to it. Execute it to determine whether it 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-while
```

```
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5.2. 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) \rightarrow (b) の証明

L は半帰納的 $\Rightarrow L$ を半認識するプログラム A が存在

c_1 : L の任意の要素
($L \neq \emptyset$ より存在)

A, c_1 より,
プログラム G
を構成

```
prog G(input w:  $\Sigma^*$ ):  $\Sigma^*$ ;  
var x:  $\Sigma^*$  ; t: num;  
begin  
  if w  $\in$   $\Sigma^*xN$  then halt( $c_1$ ) end-if;  
  x:=1st(w); t:=2nd(w);  
  if HaltInTime(<A>, x, t) then halt(x)  
    else halt( $c_1$ ) 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

Theorem 5.2 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) \rightarrow (b)

L is semi-recursive \Rightarrow a program A that semi-recognizes L exists

c_1 : any element of L

```
prog G(input w:  $\Sigma^*$ ):  $\Sigma^*$ ;  
var x:  $\Sigma^*$  ; t: num;  
begin  
  if  $w \notin \Sigma^* x \mathbb{N}$  then halt( $c_1$ ) end-if;  
  x:=1st(w); t:=2nd(w);  
  if HaltIn Time(<A>, x, t) then halt(x) else halt( $c_1$ ) 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 $\langle x, t \rangle$ に対して $\text{HaltInTime}(\langle A \rangle, x, t)$ なら $\text{halt}(x)$ 、
それ以外なら $\text{halt}(c_1)$ 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

$$L(x) = \text{accept} \rightarrow \exists t \in \mathbb{N} [\text{HaltInTime}(\langle A \rangle, x, t)]$$

$$\rightarrow \exists t \in \mathbb{N} [G \text{ は } \langle x, t \rangle \text{ に対して } x \text{ を出力}]$$

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 $L \subseteq \text{RANGE}(g)$.

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

$$L(y) = \perp \rightarrow \forall t \in \mathbb{N} [\neg \text{HaltInTime}(\langle A \rangle, y, t)]$$

$$\rightarrow \forall t \in \mathbb{N} [G \text{ は入力 } \langle y, t \rangle \text{ に対して } c_1 \text{ を出力}]$$

つまり、 \bar{L} のどの要素 y も G の出力として現れない。よって

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

\Rightarrow (2),(3)より $L = \text{RANGE}(g)$

▪ g is computable and total

▪ Since any $x \in L$ is accepted by A ,

$$L(x) = \text{accept} \rightarrow \exists t \in \mathbb{N} [\text{HaltInTime}(\langle A \rangle, x, t);$$

$$\rightarrow \exists t \in \mathbb{N} [G \text{ outputs } x \text{ for an input } \langle x, t \rangle]$$

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

that is, $L \subseteq \text{RANGE}(g)$

every element of L appears as an output of G .

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

$$L(y) = \perp \rightarrow \forall t \in \mathbb{N} [\neg \text{HaltInTime}(\langle A \rangle, y, t)]$$

$$\rightarrow \forall t \in \mathbb{N} [G \text{ outputs } c_1 \text{ for an input } \langle y, t \rangle]$$

$$\rightarrow \forall y' \in \Sigma^*, \forall t \in \mathbb{N} [g(\langle y', t \rangle) \neq y]$$

$$y \notin L, c_1 \in L \text{ thus } y \neq c_1$$

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

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

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

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

$$L = \text{RANGE}(g)$$

(b)→(a)の証明: $L = \text{RANGE}(g)$ 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 $\rightarrow g$ 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:  $\Sigma^*$  ;  
begin  
  w:= $\varepsilon$  ;  
  while true do  
    if  $G(w) = x$  then accept end-if;  
    w:=next(w)  
  end-while  
end.
```

\rightarrow 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, $B(x)=\text{accept}$
(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) \rightarrow (a)

i.e., there is a computable function g such that $L = \text{RANGE}(g)$

$\longrightarrow L$ is semi-recursive

g is computable \implies there is a program G that computes g .

Using this, we have the following program B.

```
prog B(input x);
var w: ;
begin
   $\varepsilon$  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 .

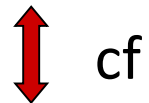
(Q.E.D.)

定理5.3. 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5.3の証明は省略



定理5.2. 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

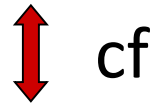
(b) $L=RANGE(g)$ となるような計算可能関数 g が存在する.

Theorem 5.3. 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 $L = \text{RANGE}(e)$.

Proof of Theorem 5.3 is omitted.



Theorem 5.2 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 $L = \text{RANGE}(g)$.

定理5.3

→ 半帰納的集合 L には

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

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5.2. 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 $L = \text{RANGE}(e)$ 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5.4 すべての集合 L に対し,
 L が半帰納的 \longleftrightarrow L が枚挙可能

Theorem 5.3

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

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

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

Def.5.2. 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 $L = \text{RANGE}(e)$.

Theorem 5.4 For any set L we have

L is semi-recursive \iff L is enumerable

5.4. 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

5.4. Comparison between enumerability and recursiveness

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

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

B: enumerable set \implies a function that enumerates B is computable
that is, we can enumerate all the elements of B

定理5.5. 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) \rightarrow (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 $\rightarrow e$ 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

Theorem 5.5. 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) \rightarrow (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 ,

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

e is computable \longrightarrow 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5.5. 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) \rightarrow (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:  $\Sigma^*$ ;  
begin  
  w :=  $\epsilon$ ;  
  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

Theorem 5.5 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) \rightarrow (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:  $\Sigma^*$ ;  
begin  
  w :=  $\epsilon$ ;  
  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, $x \in L \iff \exists w \in \Sigma^*[R(x, w)]$ 」

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 $x \in L \iff \exists w \in \Sigma^*[R(x, w)]$.”

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 ‘ $x \in L$ ’

5.5. クラスRECとクラスRE

クラスREC \equiv $\{L: L \text{ は帰納的}\}$: 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?

→ 枚挙可能集合.

5.5. Class REC and Class RE

Class REC $\equiv \{L: L \text{ is recursive}\}$: 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?

→ enumerable sets.

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

RE \equiv {L: L は枚挙可能}
co-RE \equiv {L: \overline{L} が枚挙可能}

注: L: 集合

Lが枚挙可能 \iff L が半帰納的

\iff L を半認識するプログラムAが存在.

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

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

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

例5.6. クラスRE, co-REに入る集合の例.

HALT \in RE,

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

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

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

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

Note: L : set

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

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

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

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

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

Ex. 5.6. Examples of sets belonging to class RE and class co-RE.

HALT \in RE,

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

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5.6. すべての集合 L に対し, 次の関係が成り立つ.

$$(1) L \in \text{REC} \leftrightarrow \bar{L} \in \text{REC}$$

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

証明:

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

accept \rightarrow reject, reject \rightarrow accept

と変更すると, \bar{L} を認識するプログラムを得る.

よって, $\bar{L} \in \text{REC}$

(2)はco-REの定義より明らか.

証明終

Theorem 5.6. For every set L , the followings hold:

(1) $L \in \text{REC} \leftrightarrow \bar{L} \in \text{REC}$

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

Proof:

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

If we exchange accept with reject

accept \rightarrow reject, reject \rightarrow accept

then, the resulting program recognizes \bar{L} .

So, $\bar{L} \in \text{REC}$

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

Q.E.D.

定理5.7. (1) $REC \subsetneq RE$ (2) $REC \subsetneq co-RE$

証明：演習問題かレポート問題か試験問題

Theorem 5.7. (1) $\text{REC} \subsetneq \text{RE}$ (2) $\text{REC} \subsetneq \text{co-RE}$

Proof: Exercise or Report Problem or Examination

定理5.8. $REC = RE \cap co-RE$

証明:

定理5.7より, $REC \subseteq RE \cap co-RE$

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 $\bar{L} \in RE$

→ L を半認識するプログラム A_1 と

\bar{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  $\infty$  do
```

```
    if HaltInTime( $\langle A_1 \rangle$ , x, t) then accept end-if;
```

```
    if HaltInTime( $\langle A_2 \rangle$ , 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 5.8 $REC = RE \cap co-RE$

Proof:

By Theorem 5.7 we have $REC \subseteq RE \cap co-RE$

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

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

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

Then, the following program B recognizes L .

```
prog B(input x);
```

```
var t: num;
```

```
begin
```

```
  for t:=0 to  $\infty$  do
```

```
    if HaltInTime( $\langle A_1 \rangle$ , x, t) then accept end-if;
```

```
    if HaltInTime( $\langle A_2 \rangle$ , x, t) then reject end-if
```

```
  end-for
```

```
end.
```

Q.E.D.

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 .

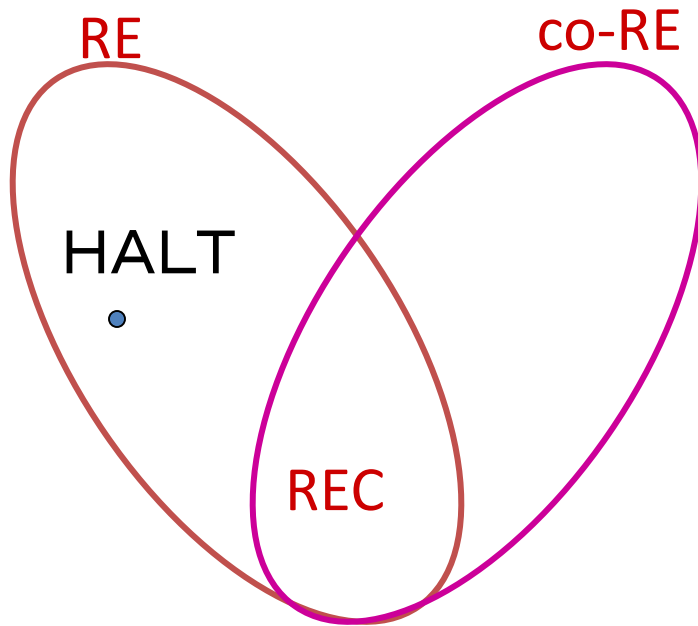
定理5.9. $RE \neq co-RE$

証明:

$RE = co-RE$ と仮定すると, $RE = RE \cap co-RE$

定理5.8より, $REC = RE$ となり, 定理5.7に矛盾.

証明終



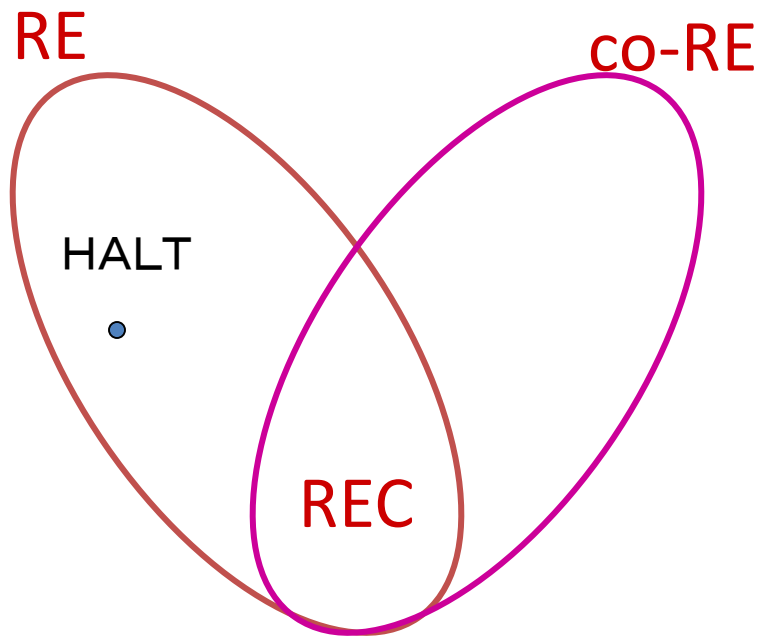
Theorem 5.9. $RE \neq co-RE$

Proof:

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

Hence, by Theorem 5.8 we have $REC = RE$, contradicts to Theorem 5.7.

Q.E.D.



短期の予定 Schedule for short term

5月7日: レポート1出題(締切: 5月14日10:50)

(Report 1; deadline is 10:50 on May 14)

5月9日: 午後, 補講あり

(Supplemental lecture on Tutorial Hour)

5月14日: 中間テスト (Mid-term Examination)

- 出題範囲(Scope): 今日の講義の範囲まで
- 持ち込んで良いもの(You can/should bring)
 - 筆記用具(Pen/Pencil)
 1. スライドのコピー(書き込み可) + 手書きノート(Copies of slides (with your marks)+ hand-written notes)
 2. 手書きノート(hand-written notes)
 3. A4用紙1枚(回収します)(A sheet of paper of A4 size, which will be submitted)