3. 計算可能性の分析

3.1. 関数から集合へ

3.2. 枚挙可能集合

- 集合Lが帰納的 ⇔ Lを次の意味で認識するプログラムAが存在する:

$$x \in L \text{ なら } A(x) = accept$$
$$y \notin L \text{ なら } A(y) = reject$$

- 集合Lが半帰納的 ⇔ Lを次の意味で認識するプログラムAが存在する:

$$x \in L \text{ なら } A(x) = accept$$
$$y \notin L \text{ なら } A(y) = \perp$$

無限ループ

3. Analysis of Computability

3.1. From Functions to Sets

3.2. Enumerable sets

- A set L is *recursive* $\Leftrightarrow$ There is a program A that recognizes L in the following sense:

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

$$y \notin L \text{ implies } A(y) = reject$$

- A set L is *semi-recursive* $\Leftrightarrow$ There is a program A that recognizes L in the following sense:

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

$$y \notin L \text{ implies } A(y) = \perp$$

A(y) does not stop

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.

➔ 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。

$e(\varepsilon), e(0), e(1), e(00), e(01), e(10), e(11), e(000), \ldots$
は L の要素を「漏れ」なく「重複」なく列挙する。

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 $L=RANGE(e)$.

➔ for a semi-recursive set L there exists a computable one-to-one function such that
$L=\{e(\varepsilon), e(0), e(1), e(00), e(01), e(10), e(11), e(000), \ldots\}$

We say the function e *enumerates* L.

We can enumerate all elements in L as
$e(\varepsilon), e(0), e(1), e(00), e(01), e(10), e(11), e(000), \ldots$
without *skip* and *duplicate*.

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 $\longleftrightarrow$ Lが枚挙可能

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
- ✓For $x \in \Sigma^*$ it is computable whether $x \in A$
- ✓There is a program that always outputs Yes/No for any $x \in \Sigma^*$

B: enumerable set

- ✓ A function that enumerates B is computable
- ✓ 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$\longrightarrow$ e を計算するプログラムが存在

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: ∑*;
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: ∑*;
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$\mathcal{REC}$とクラス$\mathcal{RE}$

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

- クラス$\mathcal{REC}$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$\mathcal{REC}$
ZEROFT $\notin$ クラス$\mathcal{REC}$

ZEROFTとは，単純for-timesプログラムが常に0を出力するかどうかを判定する述語を特徴述語とする集合のこと．ただし，for-timesに関する説明は省略した．

目標: $\mathcal{REC}$ の外側の領域の構造の解析

$\mathcal{REC}$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 $\mathcal{REC}$ and Class $\mathcal{RE}$

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

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

HALT $\notin$ class $\mathcal{REC}$
ZEROFT $\notin$ class $\mathcal{REC}$

ZEROFT is a set with characteristic predicate that a simple for-times program always outputs 0.
(although the explanation for for-times has been omitted.)

GOAL: Analyzing the structure outside $\mathcal{REC}$

What is the easiest class of sets outside $\mathcal{REC}$?

$\longrightarrow$ enumerable sets.

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

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

注 : L : 集合

Lが枚挙可能 $\Leftrightarrow$ L が半帰納的
$\Leftrightarrow$ L を半認識するプログラムAが存在.

$x \in \Sigma^*,\quad x \in L \Leftrightarrow \mathrm{A}(x) = \mathrm{accept}$
$x \notin L \Leftrightarrow \mathrm{A}(x) = \perp$

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

例3.8. クラス$\mathcal{RE}$, co-$\mathcal{RE}$に入る集合の例.

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

$\overline{\mathrm{ZEROFT}} \in \mathcal{RE}$, $\quad \mathrm{ZEROFT} \in$ co-$\mathcal{RE}$

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

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

Note : L : set

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

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

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

$x \notin L \Leftrightarrow \mathrm{A}(x) = \perp$

Note that the class co-$\mathcal{RE}$ is not complementary of the class $\mathcal{RE}$.

Ex.3.8. Examples of sets belonging to class $\mathcal{RE}$ and class co-$\mathcal{RE}$.

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

$\overline{\mathrm{ZEROFT}} \in \mathcal{RE}$, $\mathrm{ZEROFT} \in$ co-$\mathcal{RE}$

$\mathcal{RE}$とco-$\mathcal{RE}$は同程度の“難しさ”

A: 任意の$\mathcal{RE}$集合

$$\forall x \in \Sigma^*[(x \in A \longrightarrow \mathrm{X}(x)=\text{accept}) \wedge (x \notin A \longrightarrow \mathrm{X}(x)=\perp)]$$

となるプログラムXが作れる

B: 任意のco-$\mathcal{RE}$集合

$$\forall x \in \Sigma^*[(x \in B \longrightarrow \mathrm{X}(x)=\perp) \wedge (x \notin B \longrightarrow \mathrm{X}(x)=\text{accept})]$$

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．

$\mathcal{RE}$ and co-$\mathcal{RE}$ are equally "hard"

A: arbitrary $\mathcal{RE}$ set

we can write a program X such that

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

B: arbitrary co-$\mathcal{RE}$ set

we can write a program X such that

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

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) $L \in \mathcal{REC} \longleftrightarrow \overline{L} \in \mathcal{REC}$
(2) $L \in \mathcal{RE} \longleftrightarrow \overline{L} \in$ co-$\mathcal{RE}$

証明:
(1) $L \in \mathcal{REC}$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，$\overline{L}$を認識するプログラムを得る.
よって，$\overline{L} \in \mathcal{REC}$

(2)はco-$\mathcal{RE}$の定義より明らか.

証明終

Theorem 3.6. For every set L, the followings hold:
(1) $L \in \mathcal{REC} \longleftrightarrow \overline{L} \in \mathcal{REC}$
(2) $L \in \mathcal{RE} \longleftrightarrow \overline{L} \in$ co-$\mathcal{RE}$

Proof:
(1) $L \in \mathcal{REC}$, 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 $\overline{L}$.
So, $\overline{L} \in \mathcal{REC}$

(2) is obvious from the definition of co-$\mathcal{RE}$.

Q.E.D.

$\boxed{\text{定理3.7.}\quad (1)\ \mathcal{REC} \subsetneqq \mathcal{RE} \quad (2)\ \mathcal{REC} \subsetneqq \text{co-}\mathcal{RE}}$

証明: 略

$\boxed{\textbf{Theorem 3.7. } (1)\ \mathcal{REC} \subsetneqq \mathcal{RE} \quad (2)\ \mathcal{REC} \subsetneqq \text{co-}\mathcal{RE}}$

Proof : Omitted.

定理3.8. $\mathcal{REC}=\mathcal{RE}\cap$ co-$\mathcal{RE}$

証明:

定理3.7より, $\mathcal{REC}\subseteq\mathcal{RE}\cap$ co-$\mathcal{RE}$

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

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

→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 $\mathcal{REC} = \mathcal{RE} \cap \text{co-}\mathcal{RE}$

Proof:

By Theorem 2,7 we have $\mathcal{REC} \subseteq \mathcal{RE} \cap \text{co-}\mathcal{RE}$

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

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

→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.

```
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.
```

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.

Q.E.D.

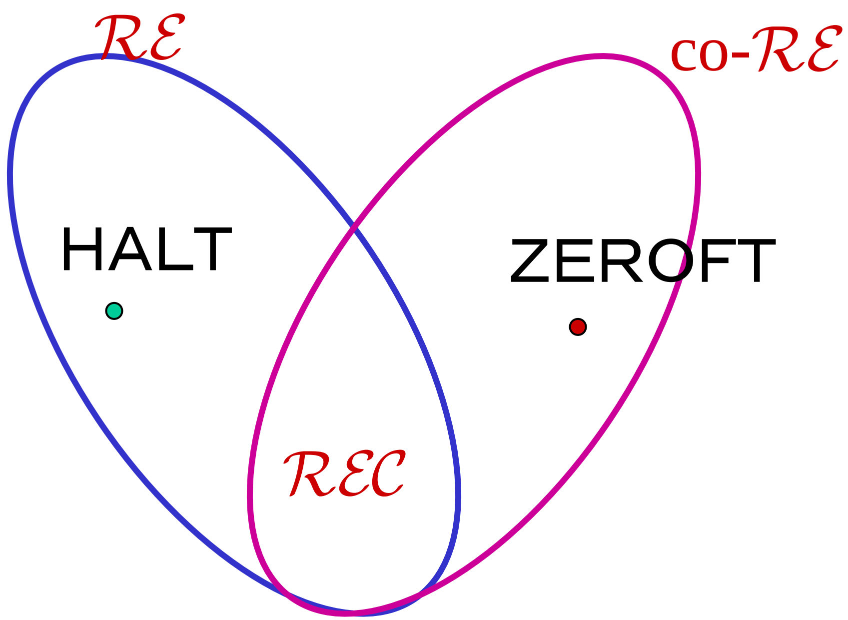
定理3.9. $\mathcal{RE} \neq$ co-$\mathcal{RE}$

証明:

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

定理3.8より, $\mathcal{REC}$＝$\mathcal{RE}$となり, 定理3.7に矛盾.

証明終



Theorem 3.9. $\mathcal{RE} \neq$ co-$\mathcal{RE}$

Proof:

If we assume $\mathcal{RE}$=co-$\mathcal{RE}$, we have $\mathcal{RE}$=$\mathcal{RE}$ $\cap$co-$\mathcal{RE}$.
Hence, by Theorem 3.8 we have $\mathcal{REC}$=$\mathcal{RE}$, contradicts to Theorem 3.7.

Q.E.D.