アナウンス
レポート提出期限：5月8日（火曜日）10:50am
中間試験：5月10日（木曜日）講義時間かチュートリアルアワー

どちらがいいか，今日の途中で聞きます．

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)!

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. 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

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

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] 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.

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

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;

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

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

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、$\forall a \in \Sigma^*[\mathrm{Zero}(a) \Leftrightarrow \mathrm{Total}(\mathrm{replace}(a))]$

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)$\neq\perp$ 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.

Yes/Noタイプのプログラム

3.2. 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

$x \in L \quad \leftrightarrow \quad$ 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 $\subsetneqq$ semi-recursive sets

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

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

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

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
$L=\{e(\varepsilon), e(0), e(1), e(00), e(01), e(10), e(11), e(000), \ldots\}$
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 $\Longrightarrow$ the characteristic predicate $R_A(x)$ is computable

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

B: enumerable set $\Longrightarrow$ 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 $\longrightarrow$ 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？

⟶ 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 $\equiv\{L$: L は枚挙可能$\}$
co-RE $\equiv$ $\{L$: $\overline{L}$ 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 $\Longleftrightarrow$ L が半帰納的
$\Longleftrightarrow$ L を半認識するプログラムAが存在.
$x \in S^*$,　$x \in L \Longleftrightarrow \mathrm{A}(x) = \mathrm{accept}$
$x \notin L \Longleftrightarrow \mathrm{A}(x) = \perp$

クラスco-REはクラスREの補クラス$\overline{\mathrm{RE}}$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.

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

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

RE $\equiv$ {L: L is enumerable}

co-RE $\equiv$ {L: $\overline{L}$ 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.

HALT $\in$ RE, $\overline{\text{HALT}}$ $\in$ 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

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

B : arbitrary co-RE set

we can write a program X such that

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

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) $L \in \mathrm{REC} \longleftrightarrow \overline{L} \in \mathrm{REC}$

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

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 $\longleftrightarrow$ $\overline{L} \in$ REC

(2) $L \in$ RE $\longleftrightarrow$ $\overline{L} \in$ 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 $\subsetneqq$ RE (2) REC $\subsetneqq$ 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 ∩ 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 $\cap$ co-RE

Proof:

By Theorem 2,7 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 $\overline{L} \in$ RE

$\rightarrow$ 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($\langle A_1 \rangle$, x, t) then accept end-if;
    if HaltInTime($\langle A_2 \rangle$, x, t) then reject end-if
  end-for
end.

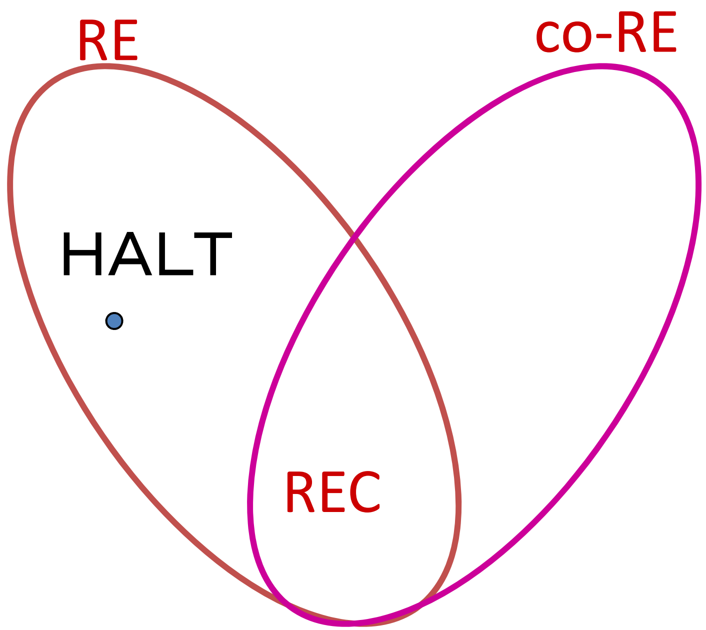
Q.E.D.

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



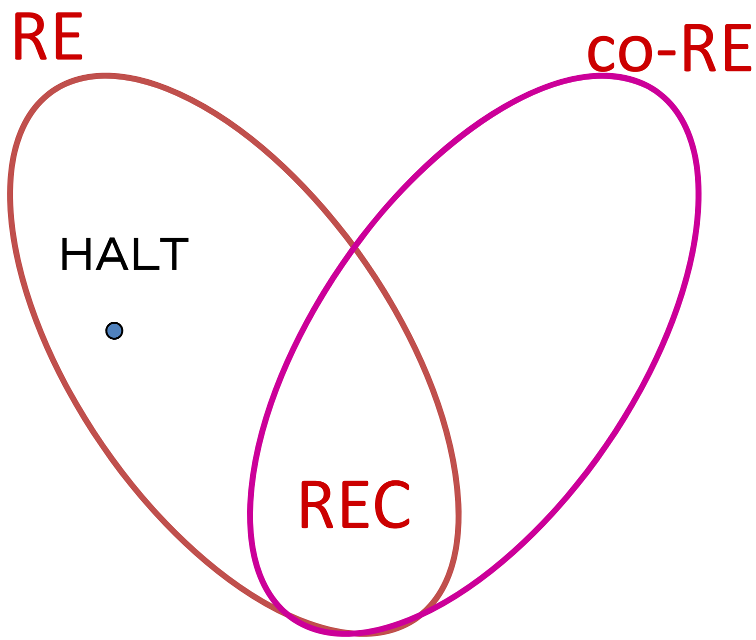
Theorem 3.9. RE $\neq$ co-RE

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.



Information

- 5月10日（木曜日）は中間テスト
  - 9:00-10:40 or 13:30-15:10（30分以上遅刻したら入室禁止）
  - 範囲は5月8日の授業分まで
  - テキスト、資料は{持ち込み禁止|持込OK}
  - 場所をちょっと考えます（「隣は空席」が本来の規則）

- Mid-term exam will be on May 10th, Thu.
  - Time: 9:00-10:40 or 13:30-15:10 (You cannot take it after 30 minutes)
  - About: up to May 8th
  - Texts and other materials are {not allowed|allowed} to bring
  - Place: to be announced (by rule, your have to have no neighbor)

レポートを集めて解答と解説

3.4. 還元可能性と完全性

- 問題の還元可能性
  …問題の相対的な難しさを測る方法
- 問題のあるクラスに関する完全性
  …そのクラス内で最も難しいことを示す方法

クラスREに属している集合の“難しさ”の比較

Aは帰納的だがBは帰納的でないとき,
BはAより難しいと言える.

では, AとBが共に帰納的でない場合は？
← 帰納的還元性による比較

A, B : 集合
AをBへ還元する ← Aの認識問題をBの認識問題に言い換えること.
（AはBへ還元可能）

3.4. Reducibility and Completeness

- Reducibility of a problem
  …Measure of relative hardness of the problem
- Completeness of a problem in a class
  …Most difficult problem in the class

Comparison of sets in the class RE by their "hardness"

If A is recursive but B is not recursive, then we can say that B is *harder* than A.

Then, what about if neither A nor B is recursive?

$\leftarrow$ comparison based on reducibility

A, B : sets

Reduce A to B $\Leftarrow$ Replace the recognition problem of A with the recognition problem of B.

(A is reducible to B)

定義3.4:
A, *B* : 任意の集合
(1) 次の条件を満たす関数 *h* を*A*から*B*への帰納的還元という.
(a) *h* は Σ^* からΣ^* への関数（全域的）
(b) $\forall x \in \Sigma^* [x \in A \leftrightarrow h(x) \in B]$
(c) *h* は計算可能
(2) *A*から*B*への帰納的還元が存在するとき,
*A*は*B*へ帰納的に還元可能という.

なお, *A*が*B*へ帰納的還元可能であることを$A \leq_m B$と記述する.
(*m* は, recursive **m**any-one reduction の *m*)

Definition 3.4:

A, B: arbitrary sets

(1) A function h is recursive reduction from A to B if

(a) h is a total function from Σ^* to Σ^*

(b) $\forall x \in \Sigma^* [x \in A \leftrightarrow h(x) \in B]$

(c) h is computable.

(2) If there is a recursive reduction from A to B, we say that A is recursively reducible to B.

By $A \leq_m B$ we express that A is recursively reducible to B.
(the m in the suffix indicates recursive **m**any-one reduction)

例3.10

EVEN＝{ $\lceil n \rceil$: nは偶数}， ODD＝{ $\lceil n \rceil$: nは奇数}

$\lceil n \rceil$ はnの2進表記（n: 自然数）

$$h_1(x) \equiv \begin{cases} \lceil n+1 \rceil & x = \lceil n \rceil \text{となっているとき} \\ x, & \text{その他のとき} \end{cases}$$

この h_1 は明らかに全域的かつ計算可能. また,

$$\forall x \in \Sigma^*[x \in \mathrm{EVEN} \leftrightarrow h_1(x) \in \mathrm{ODD}]$$

よって, h_1 はEVENからODDへの帰納的還元

$$\therefore \mathrm{EVEN} \ \leq_m \ \mathrm{ODD}$$

同じ h_1 がODDからEVENへの帰納的還元にもなっている.

$$\forall x \in \Sigma^*[x \in \mathrm{ODD} \rightarrow h_1(x) \in \mathrm{EVEN}]$$

$$\begin{aligned} \forall x \in \Sigma^*[h_1(x) \in \mathrm{EVEN} &\rightarrow \exists n \geq 0[h_1(x) = \lceil n+1 \rceil \in \mathrm{EVEN}]] \\ &\rightarrow \exists n \geq 1[h_1(x) = \lceil n+1 \rceil \in \mathrm{EVEN}]] \\ &\rightarrow \exists n \geq 1[x = \lceil n \rceil \in \mathrm{ODD}]] \rightarrow [x \in \mathrm{ODD}] \end{aligned}$$

$$\therefore \mathrm{ODD} \leq_m \mathrm{EVEN}$$

Ex.3.10

EVEN = { $\lceil n \rceil$: n is even} ,   ODD = { $\lceil n \rceil$: n is odd}

$\lceil n \rceil$ is binary representation of n (n : natural number)

$$h_1(x) \equiv \begin{cases} \lceil n+1 \rceil & \text{if } x = \lceil n \rceil \\ x, & \textit{otherwise} \end{cases}$$

This h_1 is obviously total and computable. Also,

$$\forall x \in \Sigma^*[x \in \mathrm{EVEN} \leftrightarrow h_1(x) \in \mathrm{ODD}]$$

Therefore, h_1 is a recursive reduction from EVEN to ODD.

$$\therefore \mathrm{EVEN} \ \leq_m \ \mathrm{ODD}$$

The same h_1 is also a recursive reduction from ODD to EVEN.

$$\forall x \in \Sigma^*[x \in \mathrm{ODD} \rightarrow h_1(x) \in \mathrm{EVEN}]$$

$$\begin{aligned} \forall x \in \Sigma^*[h_1(x) \in \mathrm{EVEN} &\rightarrow \exists n \geq 0[h_1(x) = \lceil n+1 \rceil \in \mathrm{EVEN}]] \\ &\rightarrow \exists n \geq 1[h_1(x) = \lceil n+1 \rceil \in \mathrm{EVEN}]] \\ &\rightarrow \exists n \geq 1[x = \lceil n \rceil \in \mathrm{ODD}]] \rightarrow [x \in \mathrm{ODD}] \end{aligned}$$

$$\therefore \mathrm{ODD} \leq_m \mathrm{EVEN}$$

EVENからODDへのもっと単純な還元

$$h_2(x) \equiv \begin{cases} 1 & x \in \mathrm{EVEN}\text{のとき} \\ 10 & \text{その他のとき} \end{cases}$$

自然数の偶奇が判定可能なので, h_2 は計算可能

$1 \in \mathrm{ODD},\ 10 \notin \mathrm{ODD}$だから

$$x \in \mathrm{EVEN} \rightarrow h_2(x) = 1 \in \mathrm{ODD}$$

$$x \notin \mathrm{EVEN} \rightarrow h_2(x) = 10 \notin \mathrm{ODD}$$

$$\therefore x \in \mathrm{EVEN} \leftrightarrow h_2(x) \in \mathrm{ODD}$$

Simpler reduction from EVEN to ODD

$$h_2(x) \equiv \begin{cases} 1 & x \in \text{EVEN} \\ 10 & \text{otherwise} \end{cases}$$

Since odd-evenness of a natural number is computable, so is h_2.

Since $1 \in$ ODD, $10 \notin$ ODD

$$x \in \text{EVEN} \rightarrow h_2(x) = 1 \in \text{ODD}$$

$$x \notin \text{EVEN} \rightarrow h_2(x) = 10 \notin \text{ODD}$$

$$\therefore x \in \text{EVEN} \leftrightarrow h_2(x) \in \text{ODD}$$

定理3.12: $A \leq_m B$という関係にある任意の集合A,Bを考える.
このとき, Bが帰納的→Aも帰納的.

証明:

$A \leq_m B$ ➔ AからBへの帰納的還元 h が存在する.

よって, $x \in A$という判定問題 ➔ $h(x) \in B$?

つまり, 次のプログラムはAを認識する.

```
prog A(input x);
begin
   if h(x)∈ B then accept else reject end-if
end.
```

Bが帰納的なら, Bを認識するプログラムが存在する.

→$h(x) \in B$を判定するプログラム

これで上記のプログラムAが完成.

よって, Aは帰納的.　　　　　　　　　　証明終

Theorem 3.12: Consider any sets A and B such that $A \leq_m B$.
Then, B is recursive → A is also recursive.

Proof:

$A \leq_m B$ ➔ there is a recursive reduction h from A to B.

So, the decision problem of $x \in A$ ➔ $h(x) \in B$?

That is, the following program recognizes A.

```
prog A(input x);
begin
   if h(x) ∈ B then accept else reject end-if
end.
```

If B is recursive, there is a program that recognizes B.

→ a program that determines $h(x) \in B$

Now, we have a complete program A.

Thus, A is recursive. Q.E.D.

与えられた集合が“手に負えない”ことを示すための方法を示唆

(i) $A \le_m B$ かつ

(ii) Aは帰納的でない.

このような集合Aを示せれば，Bは帰納的でない

例3.11:

IsProgram(a): a はプログラム A のコードか？
これは計算可能な関数である.

ZERO $\equiv$ {a: IsProgram(a) $\wedge$ $\forall$x[f_a(x) = 0]}

TOTAL $\equiv$ {a: IsProgram(a) $\wedge$ $\forall$x[f_a(x) $\neq \perp$]}

まとめると

関係	したがって,
$\overline{\mathrm{HALT}} \le_m \mathrm{ZERO}$	ZERO $\notin$ REC （$\overline{\mathrm{HALT}} \notin$ RECより）
$\mathrm{ZERO} \le_m \mathrm{TOTAL}$	TOTAL $\notin$ REC （ZERO $\notin$ RECより）

It suggests a method to show that a given set is "intractable"

(i) $A \leq_m B$ and

(ii) A is not recursive.

If we can show such a set A, then B is not recursive.

IsProgram(a): Is a the program code of A?
This problem is computable.

Ex.3.11:

$$\mathrm{ZERO} \equiv \{a: \mathrm{IsProgram}(a) \wedge \forall x[f\_a(x) = 0]\}$$

$$\mathrm{TOTAL} \equiv \{a: \mathrm{IsProgram}(a) \wedge \forall x[f\_a(x) \neq \perp]\}$$

Summarizing,

relation	what follows
$\overline{\mathrm{HALT}} \leq_m \mathrm{ZERO}$	$\mathrm{ZERO} \notin \mathrm{REC}$ (by $\overline{\mathrm{HALT}} \notin \mathrm{REC}$)
$\mathrm{ZERO} \leq_m \mathrm{TOTAL}$	$\mathrm{TOTAL} \notin \mathrm{REC}$ (by $\mathrm{ZERO} \notin \mathrm{REC}$)

定理3.13. $A \leqq_m B$という関係にある任意の集合A, Bを考える.
このとき、次のことが成り立つ。
（1）$B \in$ RE $\longrightarrow$ $A \in$ RE　（Bが枚挙可能 $\longrightarrow$ Aも枚挙可能）
（2）$B \in$ co-RE $\longrightarrow$ $A \in$ co-RE

(補注) 対偶を考えると、
（1）$A \notin$ RE $\rightarrow$ $B \notin$ RE
（2）$A \notin$ co-RE $\rightarrow$ $B \notin$ co-RE

例3.11, 定理3.13 → ZERO、TOTALは
REにもco-REにも属さない。

性質	理由
ZERO $\notin$ RE	$\overline{\mathrm{HALT}} \notin$ RE、$\overline{\mathrm{HALT}} \leqq_m$ ZERO
ZERO $\notin$ co-RE	HALT $\notin$ co-RE、HALT $\leqq_m$ ZERO
TOTAL $\notin$ RE	ZERO $\notin$ RE、ZERO $\leqq_m$ TOTAL
TOTAL $\notin$ co-RE	ZERO $\notin$ co-RE、ZERO $\leqq_m$ TOTAL

Theorem 3.13. Consider any sets A and B such that $A \leqq_m B$.
Then, we have:

(1) $B \in$ RE → $A \in$ RE (B is enumerable → so is A)

(2) $B \in$ co-RE → $A \in$ co-RE

(Remark) Their contrapositions:

(1) $A \notin$ RE → $B \notin$ RE

(2) $A \notin$ co-RE → $B \notin$ co-RE

Ex.3.11, Theorem 3.13 → Neither ZERO or TOTAL belongs to RE or co-RE.

property	reason
ZERO $\notin$ RE	$\overline{\text{HALT}} \notin$ RE、$\overline{\text{HALT}} \leqq_m$ ZERO
ZERO $\notin$ co-RE	HALT $\notin$ co-RE、HALT $\leqq_m$ ZERO
TOTAL $\notin$ RE	ZERO $\notin$ RE、ZERO $\leqq_m$ TOTAL
TOTAL $\notin$ co-RE	ZERO $\notin$ co-RE、ZERO $\leqq_m$ TOTAL

還元可能性　：　難しさを比較する手段

$A \leqq_m B \rightarrow A$の認識問題を$B$の認識問題に変換できる。

$\downarrow$

Aの難しさ $\leqq$ Bの難しさ

（Bを認識するプログラムがあればAの認識に使える。）

定理3.14.

任意に与えられた集合 A, B, C に対し、次の関係が成り立つ

(1) $A \leqq_m A$

(2) $A \leqq_m B$ かつ $B \leqq_m C$ ならば $A \leqq_m C$

$A \equiv_m B \overset{\text{def}}{\leftrightarrow} A \leqq_m B$ かつ $B \leqq_m A$

$\equiv_m$ は同値関係（同程度の難しさ）

$A \equiv_m B$ のとき、A と B は $\equiv_m$-同値という。

Reducibility : a means of comparing hardness

$A \leqq_m B$ → We can convert the recognition problem of A into that of B.

↓

hardness of $A \leq$ hardness of B

（A program recognizing B can be used to recognize A.）

Theorem 3.14. For any given sets A, B, C, we have

（1） $A \leqq_m A$

（2） $A \leqq_m B$ and $B \leqq_m C$ implies $A \leqq_m C$

$A \equiv_m B \overset{\text{def}}{\leftrightarrow} A \leqq_m B$ and $B \leqq_m A$

$\equiv_m$ is an equivalence relation (equal hardness)

If $A \equiv_m B$, we say that A and B are $\equiv_m$-equivalent.

例3.13.

ZERO $\notin$ RE　　∴ZERO $\not\leq_m$ HALT

（∵ ZERO $\leq_m$ HALTとすると、HALT$\in$ REなので

ZERO$\in$ REとなり矛盾）

一方、HALT $\leq_m$ ZERO

∴ZEROはHALTより真に難しい。

例3.14.

すべての帰納的集合は互いに帰納的に同値。

たとえば、EVEN（偶数の集合）とPRIME（素数の集合）は

帰納的に同値

$$\mathrm{EVEN} \equiv_m \mathrm{PRIME}$$

（両方とも帰納的という意味で同程度の難しさ）

どちらも計算できるという意味で同程度に難しい

Ex. 3.13.

ZERO $\notin$ RE $\therefore$ ZERO $\not\leq_m$ HALT

($\because$ if ZERO $\leq_m$ HALT we have HALT $\in$ RE and ZERO $\in$ RE, a contradiction)

On the other hand, HALT $\leq_m$ ZERO

$\therefore$ ZERO is strictly harder than HALT.

Ex. 3.14.

All the recursive sets are recursively equivalent to each other.

For example, EVEN (set of even numbers) and PRIME (set of primes) are recursively equivalent

$$\text{EVEN} \equiv_m \text{PRIME}$$

(both of them are equally hard in the sense that they are recursive.)

both computable

“クラスREの中で最も難しい集合”の定義

（one of the most difficult sets in RE）

定義3.5.

集合 A が次の条件を満たすとき、それを（$\leqq_m$ のもとで）RE-完全（RE-complete）という。

（a）$\forall L \in \mathrm{RE}$ ［$L \leqq_m A$］

(A より真に難しいものはREには存在しない)

（b）$A \in \mathrm{RE}$

集合Aが上記の条件（a）だけを満たすとき、RE-困難（RE-Hard）という。

（すべてのRE集合より難しい集合のこと）

Definition of “**the hardest sets in the class RE**”

Def. 3.5.

A set A is called RE-complete (under $\leqq_m$) if the following conditions hold

(a) $\forall L \in \mathrm{RE} \quad [L \leqq_m A]$

(no element of RE is strictly harder than A).

(b) $A \in \mathrm{RE}$

If a set A satisfies only (a) above, it is called RE-hard.

(meaning sets harder than any RE set)

定理3.15: HALTはRE-完全

（証明）

HALT ∈ REなので、条件(b)はOK。

L: 任意のRE集合とする。

→*L*を半認識するプログラム L が存在する

すべての $x \in \Sigma^*$ に対し、

$x \in L$ ⟺ Halt(<L , *x*>) ⟺ <L , *x*> ∈ HALT

よって、$h(x) \stackrel{\mathrm{def}}{=} \langle \mathrm{L}, x \rangle$ は *L* からHALTへの帰納的還元。

（証明終）

Theorem 3.15 HALT is RE-complete.

(Proof)

Since HALT $\in$ RE, the condition (b) is satisfied.

L : any RE set.

→ a program L that semi-recognizes *L*.

for any $x \in \Sigma^*$

$$x \in L \Leftrightarrow \text{Halt}(\langle L, x \rangle) \Leftrightarrow \langle L, x \rangle \in \text{HALT}$$

Thus, $h(x) \stackrel{\text{def}}{=} \langle L, x \rangle$ is a recursive reduction from *L* to HALT.

Q.E.D.

定理3.16: A, B を任意の集合とする。
（1）[AがRE-困難] かつ [$A \leqq_m B$] ならば B はRE-困難
（2）A がRE-困難　$\leftrightarrow \overline{A}$ がco-RE-困難

例3.15. 定理3.16を用いて、いろいろな集合の困難性（完全性）を示す。

集合	難しさ	主な理由
HALT	RE-完全	定理3．15
$\overline{\text{HALT}}$	co-RE完全	HALTがRE-困難、$\overline{\text{HALT}} \in$ co-RE
ZERO	RE-困難、co-RE困難	HALT $\leq_m$ ZERO、
$\overline{\text{TOTAL}}$	RE-困難、co-RE困難	ZERO $\leq_m$ TOTAL

Theorem 3.16: Let A and B be arbitrary sets.

(1) $[A$ is RE-hard and $A \leqq_m B]$ implies B is RE-hard.

(2) A is RE-hard $\leftrightarrow \overline{A}$ is co-RE-hard.

Ex.3.15 Using Theorem 3.16, we can show hardness of various sets.

Sets	hardness	reasons
HALT	RE-complete	Theorem3.15
$\overline{\text{HALT}}$	co-RE complete	HALT is RE-hard, $\overline{\text{HALT}} \in$ co-RE
ZERO	RE-hard、co-RE hard	HALT $\leq_m$ ZERO、
$\overline{\text{TOTAL}}$	RE-hard、co-RE hard	ZERO $\leq_m$ TOTAL

H: RE-完全集合の集合

H: REの中で“最も難しい集合”

REC : REの中で“最もやさしい集合”

還元 $\leqq_m$ のもとで

定理3.17.

(1) $\mathrm{REC} \cap H = \phi$

(2) $\mathrm{RE} - (\mathrm{REC} \cup H) \neq \phi$

(1) $\mathrm{REC} \subsetneq \mathrm{RE}$

RECは同値関係 $\equiv_m$ のもとで閉じている。

(2)の証明は複雑なので省略。

H: an RE-complete set

H: "hardest set" in RE

REC : "easiest set" in RE

Under the reduction $\leqq_m$

Theorem 3.17.

(1) $\mathrm{REC} \cap \mathrm{H} = \phi$

(2) $\mathrm{RE} - (\mathrm{REC} \cup \mathrm{H}) \neq \phi$

(1) $\mathrm{REC} \subsetneqq \mathrm{RE}$

REC is closed under the equivalence relation $\equiv_m$.

(2) The proof is complicated, and so omitted.