1.2.2. アルゴリズムの記述方法

PASCAL風の手続き型プログラミング言語

例：2進表現で与えられた自然数を通常の自然数に変換

```
1.  prog TR(input x: string on Σ): integer;
2.  label LOOP;
3.  var n: num; c: string;
4.  %単にstringと型指定したときはstring on Γ型を意味する.
5.  begin
6.      if x ≠ 0 ∧ head(x) = 0 then LOOP: goto LOOP: end-if;
7.      %2進表記でないものが入力されると無限ループに入る.
8.      n:=0;
9.     while x > ε do  % εは空列を表す定数
10.        c:=head(x);
11.        if c=1 then n:=2*n+1
12.                else  n:=2*n end-if;
13.        x:=right(x)
14.    end-while;
15.    halt(n)
16. end.
```

1.2.2. Algorithm Description

PASCAL-like procedural programming language

Ex. Conversion from a binary natural number into an ordinary one.

```
1.  prog TR(input x: string on Σ): integer;
2.  label LOOP;
3.  var n: num; c: string;
4.  % string implies a type of string on Γ.
5.  begin
6.     if x ≠ 0 ∧ head(x) = 0 then LOOP: goto LOOP: end-if;
7.     %if non-binary expression is input then goto infinite loop
8.     n:=0;
9.    while x > ε do  % ε is a constant for an empty string
10.       c:=head(x);
11.       if c=1 then n:=2*n+1
12.              else  n:=2*n end-if;
13.       x:=right(x)
14.    end-while;
15.    halt(n)
16. end.
```

注意事項:

- 入出力に関する記述は省く.
- TR: プログラム名 （ ）内が入力変数とその型指定,
  （ ）の右が出力の型
- f_TR: プログラムTRが計算する(部分)関数
- 正常終了と無限ループ
  - 出力が得られるのはhalt文で正しく停止するときのみ.
  - 出力が得られない場合, プログラムが計算する関数値は未定義とみなす.

$$f\_TR(001) = \perp$$

Remarks:

- description concerning input and output are omitted.
- TR: program name (input variable and its type declaration)
  the type of output follows
- *f*_TR: the (partial) function computed by the program TR
- normal termination and infinite loop
  - Output is obtained only when it terminates correctly by a halt sentence.
  - When an output is obtained, the function value computed by the program is considered as "undefined"

$f\_TR(001) = \perp$

変数の型

自然数型： num型
文字列型： string型
文字列を構成する“文字”として許される記号0, 1, 2, … , *a*, *b*, …
の全体を Γ とする.

文字列型データの基本演算

head(*x*)	*x* の最初の1文字
right(*x*)	*x* の2文字目から右の部分
tail(*x*)	*x* の最後の1文字
left(*x*)	*x* の先頭から最後の2文字目までの部分
$x \# y$	*x* と *y* の連接
$x \leq y$	長さ優先の辞書式順序による大小比較

ただし, head(ε)=right(ε)=tail(ε)=left(ε)= ε

Types of variables

natural number type: type num

string type:

Let Γ be a set of all symbols 0, 1, 2, ... , *a*, *b*, ... used in strings

Elementary operations on strings

head(*x*)	the first letter of *x*
right(*x*)	the part of *x* after its first letter
tail(*x*)	the last letter of *x*
left(*x*)	the part of *x* before its last letter
x # *y*	concatenation of *x* and *y*
$x \leq y$	comparison based on lexicographic order with length preferred

where, head(ε)=right(ε)=tail(ε)=left(ε)= ε

自然数の1進表記

自然数n → 0を n 個並べる

$\lceil n \rceil$: 自然数 n の2進表記　$\lceil 4 \rceil$ ➔ 100

$\overline{n}$: 自然数 n の1進表記　$\overline{4}$ ➔ 00000

例2.2. 一般の文字列(Γ上の文字列)もΣ上の文字列で表現可能.

e.g. 8ビットの2進列でのコード化(ASCIIコードなど)

補題2.2. すべての構造型はΣ^*型で表現できる.

Unary representation of a natural number

natural number n → sequence of n 0s

$\lceil n \rceil$: binary representation $\lceil 4 \rceil$ ➔ 100

$\overline{n}$: unary representation $\overline{4}$ ➔ 00000

Ex. 2.2: Ordinary letters are also represented by binary strings

e.g. each letter is coded in 8 bits

Lemma 2.2. All structure types are represented by Σ^* type.

定理2.3. われわれのプログラミング言語のすべてのデータ型と
その上の基本演算はΣ^*型とその上の基本演算だけで実現できる.

「われわれのコード化法」

$\lceil x \rceil$: データ x を表すΣ^*の元(x のコード)

$\lfloor w \rfloor$: Σ^*の元 w が表しているデータ

例2.6. プログラムも(改行コード入りの)文字列と見なしてコード化.

```
prog A ...        A  = 0111000  01110010  01101111 ....
begin                        p          r              o ....
  :
end.                    01100101  01101110  00101110 ...
                             e          n          d
```

もっと使いやすい
コード化もあるが,
当面はこれで.

Theorem 2.3. All the data types and elementary operations in our programming language can be realized on Σ^*.

"Our encoding method"

$\lceil x \rceil$: an element of Σ^* representing a data x (a code of x)

$\lfloor w \rfloor$: a data represented by an element w of Σ^*

Ex.2.6. Programs are also coded by considering them as strings

```
prog A ...        A = 0111000  01110010  01101111 ....
  begin                    p          r          o ....
    :
  end.                01100101  01101110  00101110 ...
                           e          n          d
```

We could use a different coding method, but ...

2.2. 計算の基本要素

「データ」や「プログラム」を最小限の資源で表現
…対象を絞ることで議論を単純化する

2.2.2. 制御機構のための基本要素

補題2.4. 関数プログラム(関数定義と関数呼び出し)は,
すべてif文とgoto文によって実現できる.

（略証）

フローチャート → if文とgoto文
再帰呼び出し → スタックを用いて書きなおす

補題2.5. すべての制御構造はif文とgoto文によって実現できる.

定理2.6. すべての制御構造はif文とwhile文によって実現できる.
（例に基づいて証明）

プログラムの「標準形」

2.2.2. Elements for Control Mechanism

Lemma 2.4: A function (definition and call of function) can be implemented by if and goto statements.

（Proof sketch）

flowchart → if statement and goto statement

recursive call → can be rewritten using a stack

Lemma 2.5. All the control mechanisms can be realized by if and goto statements.

Theorem 2.6. All the control structures can be realized by if and while statements.

（Proof based on examples）

“Standard Form” of a program

```
% xが0*かどうかを判定するプログラム
prog A(input x: Σ*): Σ*;
label LOOP;  var a: Σ*;
begin
LOOP: if x= ε then halt(1) end-if;
          a:=head(x); x:=right(x);
          if a=1 then halt(0) else goto LOOP end-if
end.
```

これを次のように変形する.

(1) プログラムの各行は次のいずれか.
  (a) 代入文とgoto文
  (b) if Σ*上の比較 then goto ... else goto ... end-if
  (c) halt(変数)

(2) プログラム本体の各行には, L1から始まり, L2, L3,...と順にラベルづけされている.

(3) ただし, (c)の形の行はプログラムの最後に1箇所しか現れず, それはL0とラベル付けされている.

```
% program to determine whether x is 0* or not
prog A(input x: Σ*): Σ*;
label LOOP;  var a: Σ*;
begin
LOOP: if x= ε then halt(1) end-if;
          a:=head(x); x:=right(x);
          if a=1 then halt(0) else goto LOOP end-if
end.
```

Convert it as follows.

(1) Each line of a program is one of the followings:
  (a) substitution, goto statement
  (b) if comparison on Σ^* then goto ... else goto ... end-if
  (c) halt（variable）

(2) Each line in the program body is labeled as L1, L2, ...

(3) The line of the form (c) above appears only once in the program and it is labeled as L0.

```
prog A(input x: Σ*): Σ*;
label LOOP;  var a: Σ*;
begin
LOOP: if x= ε then halt(1) end-if;
        a:=head(x); x:=right(x);
        if a=1 then halt(0) else goto LOOP end-if
end.
```

```
prog B(input x: Σ*): Σ*;
label L0, L1, L2, L3, L4, L5, L6;
var a,c: Σ*;
begin
L1: if x= ε then goto L5 else goto L2 end-if;
L2: a:=head(x); goto L3;
L3: x:=right(x); goto L4;
L4: if a=1 then goto L6 else goto L1 end-if;
L5: c:=1; goto L0;
L6: c:=0; goto L0;
L0: halt(c)
end.
```

(3-2) goto 文で次に実行する行に移動

(3-1) 通常の処理＋次に実行する行を決める

(2) haltの値を設定

(1) halt文を追加

```
prog A(input x: Σ*): Σ*;
label LOOP;  var a: Σ*;
begin
LOOP: if x= ε then halt(1) end-if;
        a:=head(x); x:=right(x);
        if a=1 then halt(0) else goto LOOP end-if
end.
```

```
prog B(input x: Σ*): Σ*;
label L0, L1, L2, L3, L4, L5, L6;
var a,c: Σ*;
begin
L1: if x= ε then goto L5 else goto L2 end-if;
L2: a:=head(x); goto L3;
L3: x:=right(x); goto L4;
L4: if a=1 then goto L6 else goto L1 end-if;
L5: c:=1; goto L0;
L6: c:=0; goto L0;
L0: halt(c)
end.
```

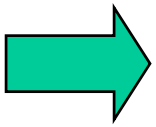
(3-2) Jump to the next line indicated by goto

(3-1) Usual process + goto next line

(2) Set values of halt

(1) Add halt

```
prog B(input x: Σ*): Σ*;
label L0, L1, L2, L3, L4, L5, L6;
var a,c: Σ*;
begin
L1: if x= ε then goto L5 else goto L2 end-if;
L2: a:=head(x); goto L3;
L3: x:=right(x); goto L4;
L4: if a=1 then goto L6 else goto L1 end-if;
L5: c:=1; goto L0;
L6: c:=0; goto L0;
L0: halt(c)
end.
```

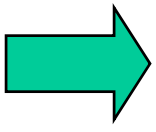


```
prog C(input x: Σ*): Σ*;
var pc: num; a,c:Σ*;
begin
  pc:=1;
  while pc != 0 do
     case pc of
     1:  if x= ε then pc:=5 else pc:=2 end-if;
     2: a:=head(x); pc:=3;
     3: x:=right(x); pc:=4;
     4: if a=1 then pc:=6 else pc:=1 end-if;
     5: c:=1; pc:=0;
     6: c:=0; pc:=0;
     end-case;
  end-while;
  halt(c)
end.
```

goto L*k* ➔ pc:=*k*;

ただし，case文は
実際にはif文の
組み合わせで実現.

```
prog B(input x: Σ*): Σ*;
label L0, L1, L2, L3, L4, L5, L6;
var a,c: Σ*;
begin
L1: if x= ε then goto L5 else goto L2 end-if;
L2: a:=head(x); goto L3;
L3: x:=right(x); goto L4;
L4: if a=1 then goto L6 else goto L1 end-if;
L5: c:=1; goto L0;
L6: c:=0; goto L0;
L0: halt(c)
end.
```



```
prog C(input x: Σ*): Σ*;
var pc: num; a,c:Σ*;
begin
  pc:=1;
  while pc != 0 do
     case pc of
     1:  if x= ε then pc:=5 else pc:=2 end-if;
     2: a:=head(x); pc:=3;
     3: x:=right(x); pc:=4;
     4: if a=1 then pc:=6 else pc:=1 end-if;
     5: c:=1; pc:=0;
     6: c:=0; pc:=0;
     end-case;
  end-while;
  halt(c)
end.
```

goto L*k* ➔ pc:=*k*;

Remark: case statement is realized by combination of if and goto

単純プログラム： 下の要素のみで構成されるプログラム

データ型： Σ上の文字列型（Σ型，Σ*型）

基本演算： 文字列型の基本演算

実行文： 代入文，if文(case文)，while文，halt文

定理2.7. どんなプログラムもそれと同値な単純プログラムに書換えることができる. しかも次のような標準形プログラムに書き直せる

```
prog プログラム名(input ...) ;
var pc: Σ*; ... Σ; ... Σ*; %pcの値は自然数の2進表記
begin
  pc:=1;
  while pc != 0 do
      case pc of
      1: (文) ;
      2: (文) ;
        :
      k: (文) ;
     end-case
   end-while;
  halt(c)
end.
```

各(文)の形は

- if 比較文 then pc:=k1 else pc:=k2 end-if
- 代入文 ; pc:=k;

のいずれか

Simple program: a program consisting only of the following elements.

data type: string type on Σ（Σ type, Σ^* type）

elementary operations: elementary operations on strings

execution statements: substitution, if (case), while, halt

Theorem 2.7 Any program can be rewritten into its equivalent simple program of the following form:

```
prog  Program name(input ...) ;
var pc: Σ*; ... Σ; ... Σ*; % value of pc is a binary representation of an integer
begin
  pc:=1;
  while pc != 0 do
      case pc of
      1: (statement) ;
      2: (statement) ;
        :
      k: (statement) ;
     end-case
    end-while;
  halt(c)
end.
```

each statement is one of the two:

- if comparison then pc:=k1 else pc:=k2 end-if
- substitution; pc:=k;

定理2.8. すべての計算可能関数に対し，
それを計算する標準形プログラムが存在する.

プログラムカウンタの働きを考えてみよう.

更なる制約（テキスト101ページ）

「各文は高々定数時間で実行できるものだけ」

u, u’: Σ型の変数, v,v’: Σ*型の変数

c: Σ型の定数, s: Σ*型の定数

（代入文）

(1) u:=c; (2) u:=u’;

(3) u:=head(v); (4) u:=tail(v);

(5) v:=s; ~~(6) v:=v’;~~ ?

(7) v:= right(v); (8) v:=left(v);

(9) v:=u # v; (10) v:=v # u;

（比較文）

(11) u=c (12) v=s

Theorem2.8 For every computable function, there is a program in the standard form.

Consider a behavior of program counter.

Further constraints（refer to 101 page of the textbook）

“each statement must be implemented in constant time”

u, u’: variables of Σ type, v,v’: variables of Σ^* type

c: constant of Σ type, s: constant of Σ^* type

（Substitution）

（1） u:=c; (2) u:=u’;

(3) u:=head(v); (4) u:=tail(v);

(5) v:=s; ~~(6) v:=v’;~~ **?**

(7) v:= right(v); (8) v:=left(v);

(9) v:=u # v; (10) v:=v # u;

（Comparison）

(11) u=c (12) v=s

2. 計算可能性入門

計算とは何か？

- 「計算できる」ことと「計算できない」ことの違い
  - 「計算」の基本要素(前回)
  - 「計算できない」ことの証明…対角線論法(今回)

2.1. 帰納的関数論概観

帰納的関数論(recursive function theory)

① “計算”とは何かについての研究
② 計算不可能性の証明
③ 計算不可能な関数のクラスの構造的研究
④ 他の数学との関連分野

Chapter 2: Introduction to Computability

What “Computation” is…

- Difference between “computable” and “incomputable”
  - Basic factor of a “computation” (Done)
  - Proof of “incomputable”…diagonalization (Today)

2.1. Studies on recursive functions

recursive function theory

(1) studies on what is "computation"

(2) proof of incomputability

(3) structural studies on a class of incomputable functions

(4) related mathematics fields

2. 計算可能性入門

① 計算とは何かについての研究

「何をもって計算可能な関数というか？」

・クリーネが定義した帰納的関数(recursive function)
・チューリングが考えたチューリング機械(Turing machine)

➔帰納的関数全体＝チューリング機械で計算可能な関数全体

⇔

計算可能性の定義…チャーチの提唱（Church's Thesis)

Chapter 2: Introduction to Computability

(1) Studies on what is computation.

"When do we call a function computable?“

- recursive function theory by Kleene
- Turing machine theory by Turing

➔the whole set of recursive functions
＝the whole set of functions computable by Turing machines

Church's Thesis on the definition of “computability”

② 計算不可能性の証明

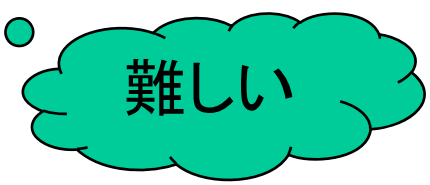
・計算可能性の証明ではプログラムを作ればよい

・計算不可能性の証明では

どんなプログラムも作れないことの証明:

「対角線論法」

「帰納的還元性」



③ 計算不可能な関数のクラスの構造的研究

難しさに応じて階層化されたクラス

→構造的研究

④ 他の数学との関連分野

数理論理学(mathematical logic)など

(2) Proof of incomputability

- Proof of computability is easy: just give a program
- to prove incomputability
  must prove that no program exists…
  proof tools: diagonalization
  recursive reducibility

(3) Structural studies on a class of incomputable functions

hierarchical class depending of hardness
→structural studies

(4) Related mathematics fields

mathematical logic

2. 計算可能性入門

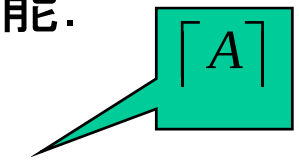
2.4. 計算不可能性の証明と対角線論法

停止問題(停止性判定問題)

入力: プログラム A とそれへの入力 x

出力: Aへ x を与えて実行させると(いつかは)停止するか？

ここでは1入力プログラムの停止問題のみ考えるが，この結果を多入力の場合に拡張することは可能.



(注意)プログラムもΣ^*上にコード化可能.

つまり，A も x もΣ^*上の文字列と考えることができる.

Chapter 2: Introduction to Computability

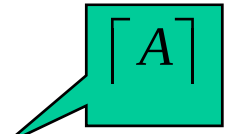
2.4. Incomputability Proof and Diagonalization

Halting Problem（Problem of deciding whether it halts）

Input: a program A and an input x to it.

Output: Whether does it stop if x is given to A?

Here we only consider the problem only for one-input programs, but we can generalize the argument into the cases of multiple inputs.



（Remark）Programs are also encoded into strings on Σ^*.

That is, A and x are also considered as strings on Σ^*.

各 $a, x \in \Sigma^*$ に対し,

IsProgram(a)

$\Leftrightarrow$ [aは1入力の文法的に正しい標準形プログラムのコード]

eval(a, x)

$$\equiv \begin{cases} f\_a(x), & \text{IsProgram}(a)\text{のとき,} \\ ?, & \text{その他のとき.} \end{cases}$$

$f\_a(x)$: コード a が表すプログラムに入力 x を加えたときの出力の値. ($f_a(x)$は部分関数)

定理2.16: IsProgram と eval はプログラムで実現可能.

IsProgram : コンパイラ(lint)

eval(a, x) : コード a が表すプログラムに x を入力したときの実行をシミュレートすればよい.
つまり, インタープリタ. (エミュレータ)

詳細は4.3節

for $a, x \in \Sigma^*$

IsProgram(a)

$\Leftrightarrow$ [a is a one-input grammatically correct standard program]

eval(a, x)

$$\equiv \begin{cases} f\_a(x), & \text{if IsProgram}(a), \\ ?, & \text{otherwise.} \end{cases}$$

$f\_a(x)$: output value when an input x is given to the program represented by the code a

Theorem2.16: IsProgram and eval are computable (programmable).

IsProgram : compiler(lint program)

eval(a, x) : it suffices to simulate the behavior of the program for a code a with an input x, i.e. interpreter or emulator

refer to Section 4.3 for detail

述語Haltの定義

コードaが表現するプログラム

各 $a, x \in \Sigma^*$ に対し

$\mathrm{Halt}(a, x)$

$\Leftrightarrow$ [$\mathrm{IsProgram}(a) \wedge$ [入力 x に対し$\lfloor a \rfloor$ は停止する.]]

例2.1 ループを含んでいても停止性を簡単に判定できる場合.

```
prog B(input w: Σ*): Boolean;
label LOOP;
begin
  if w≠ε then LOOP: goto LOOP
         else halt(0) end-if
end.
```

実際のプログラムは
標準形でかかれていると仮定

- $\mathrm{Halt}(\lceil \mathrm{B} \rceil, \varepsilon)$: 入力εに対しプログラム B は停止.
- 任意の $x \in \Sigma^* - \{\varepsilon\}$ に対し, $\neg\mathrm{Halt}(\lceil \mathrm{B} \rceil, x)$

Bの停止性は
容易に判定できる

（注意）$\mathrm{eval}(\lceil \mathrm{B} \rceil, \varepsilon) = 0$ だが, $x \neq \varepsilon$ に対しては

$\mathrm{eval}(\lceil \mathrm{B} \rceil, x) = \perp$ （未定義）

Definition of a predicate Halt

for $a, x \in \Sigma^*$

Halt(a, x)

$\Leftrightarrow$ [IsProgram(a) $\wedge$ [$\lfloor a \rfloor$ stops for an input x]]

Program described by code *a*

Ex.2.1 Halting is sometimes easily checked even with loops

```
prog B(input w: Σ*): Boolean;
  label LOOP;
  begin
    if w ≠ ε then LOOP: goto LOOP
             else halt(0) end-if
  end.
```

Assume that the program is written in the standard form

- Halt($\lceil$ B $\rceil$, ε): program B stops for an input ε
- $\neg$Halt($\lceil$B$\rceil$, x) for any $x \in \Sigma^* - \{\varepsilon\}$

  Thus, we can easily check whether B halts or not.

(Remark) eval($\lceil$B$\rceil$, ε) = 0 but, for $x \neq \varepsilon$

eval($\lceil$B$\rceil$, x) = $\perp$ (undefined)

定理2.17 **Haltは計算不可能**

（証明）

背理法：Haltが計算可能だと仮定して矛盾を導く.

Haltが計算可能➔Haltを計算するプログラムHが存在する.

そのHを用いて，次のようなプログラムXを作る.

```
prog X(input w: Σ*): Σ*;
 label LOOP;
 begin
     if H (w, w) then LOOP: goto LOOP
                 else halt(0) end-if
 end.
```

実際には標準形で書かれていると仮定.

プログラム⌊*w*⌋に*w*を入力したとき停止するかどうかを
プログラムHを呼び出して判定し，
答が *true* なら無限ループに入り，
答が *false* なら0を出力して停止する，というプログラム

H：プログラム，Halt：述語

Theorem 2.17: Halt is incomputable.

(Proof)

By contradiction: Assume that Halt is computable.

Halt is computable ➔ There is a program H to compute Halt.

Using the H, we obtain the following program X.

```
prog X(input w: Σ*): Σ*;
 label LOOP;
 begin                          Assume that it is written in the standard form
    if H (w, w) then LOOP: goto LOOP
                 else halt(0) end-if
 end.
```

Using the function H we check whether the program ⌊*w*⌋ stops for an input *w*. If the answer is "HALT" then the program X enters infinite loop, and if it is "DO NOT HALT" then it stops.

H: program or function, Halt: predicate

X(w)
プログラム⌊w⌋ にwを入力したとき停止するか
どうかをプログラムHを呼び出して判定し,
答が *true* なら無限ループに入り,
答が *false* なら0を出力して停止する

$x_1 = \lceil X \rceil$とし, x_1を
プログラムXに入力
(i) ループに入ってしまう, or
(ii) 0を出力して停止.

(i) を仮定すると…
- プログラムがループに入るから, H $(x_1, x_1) = true$
- つまり $X(x_1)$ は停止する: 仮定に矛盾

(ii) を仮定すると…
- プログラムが終了するから, H $(x_1, x_1) = false$
- つまり $X(x_1)$ は停止しない: 仮定に矛盾

どちらの場合も矛盾を生じる。
したがって「Haltは計算可能」という仮定は誤り.
証明終

H:プログラム
Halt:述語

Let $x_1 = \lceil X \rceil$ and input x_1 to the program X

(i) enters an infinite loop, or

(ii) stops normally with the output 0.

Case (i)

- Since it enters infinite loop, $\neg$Halt(x_1, x_1)
- at the if statement in the program X we have H (x_1 , x_1)=false

So, halt(0) is executed（normal termination）: contradiction

Case (ii)

- Since it stops, Halt(x_1, x_1) is true.
- at the if statement in the program X we have H (x_1, x_1)=true

So, it enters an infinite loop: contradiction

In either case we have a contradiction.

That is, the assumption that "Halt is computable" is wrong.

End of proof

H:program or function, Halt:predicate

定理2.18 次の関数 diag は計算不可能

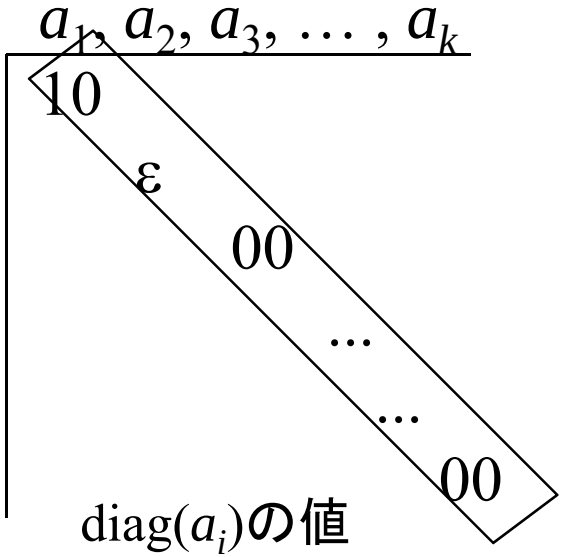
$\mathrm{diag}(a) = f\_a(a) \# 0$, $\mathrm{Halt}(a, a)$のとき
$= \varepsilon$, その他のとき

証明:
計算可能な(1引数の)関数全体の集合をF_1とする.
プログラムのコードはΣ^*の元だから,
"文法的に正しいプログラムのコード"を小さい順に$a_1, a_2, \ldots, a_k, \ldots$
と並べることができる.(長さ優先の辞書式順序)
F_1の関数も$f_a_1, f_a_2, \ldots, f_a_k, \ldots$ と並べることができる.

	a_1	a_2	a_3	…	a_k
f_a_1	1	ε	00		0
f_a_2	0	⊥	1		ε
f_a_3	0	11	0		11
:	.....				
:	.....				
f_a_k	ε	ε	1		0

$f\_a_i(a_j)$

f_a_iの値



$\mathrm{diag}(a_i)$の値

$\mathrm{diag}(a_i) = w \# 0$, $f_a_i(a_i)$の値wが未定義 ⊥でないとき
$= \varepsilon$, その他のとき

Theorem 2.18 The following function diag is incomputable.

$$\text{diag}(a) \begin{cases} = f\_a(a) \# 0, & \text{if Halt}(a, a) \\ = \varepsilon, & \text{otherwise} \end{cases}$$

Proof:

Let F_1 be a set of all computable functions (with one argument) .

Since a code of a program is an element of Σ^*,

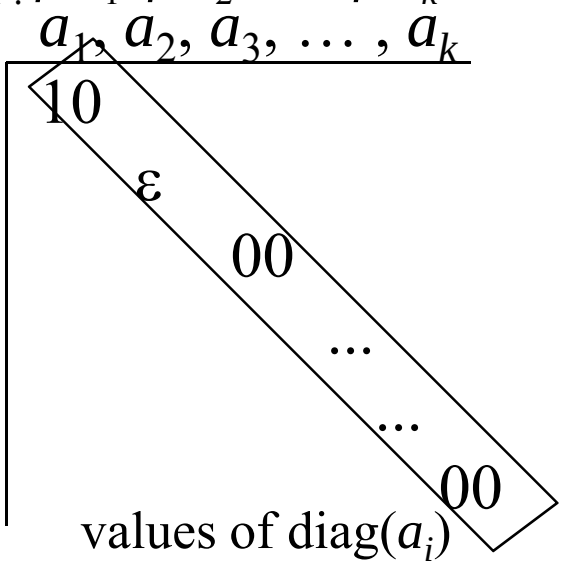
we can enumerate all grammatically correct program codes

$a_1, a_2, \ldots, a_k$ ... in the psuedo-lexicographical order.

We can also enumerate all the functions of F_1 : $f\_a_1, f\_a_2, \ldots, f\_a_k, \ldots$

	a_1	a_2	a_3	$\ldots$	a_k
f_a_1	1	ε	00		0
f_a_2	0	$\perp$	1		ε
f_a_3	0	11	0		11
:	.....				
:	.....				
f_a_k	ε	ε	1		0

values of f_a_i



values of diag(a_i)

diag(a_i) = w#0, if the value w of (f_a_i , a_i) is not undefined $\perp$.

ε, otherwise

diagはどのf_a_iとも異なる.

理由:diag()とf_a_i()は，対角線の所で必ず異なる.

$$\mathrm{diag}(a_i) \neq f\_a_i(a_i)$$

$$\mathrm{diag} \notin F_1$$

つまり，関数diagは計算可能でない.

証明終

[関数]の個数は[計算できる関数]
の個数よりも``多い”

対角線論法:

ある要素が無限集合に属さないことを示すための論法。

ある関数の集合 G が与えられたとき，その集合に属さない関数 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、関数集合 G には属さない。

diag is different from any f_a_i.

Why: diag() is different from $f\_a_i()$ at its diagonal position.

$$\text{diag}(a_i) \neq f\_a_i(a_i)$$

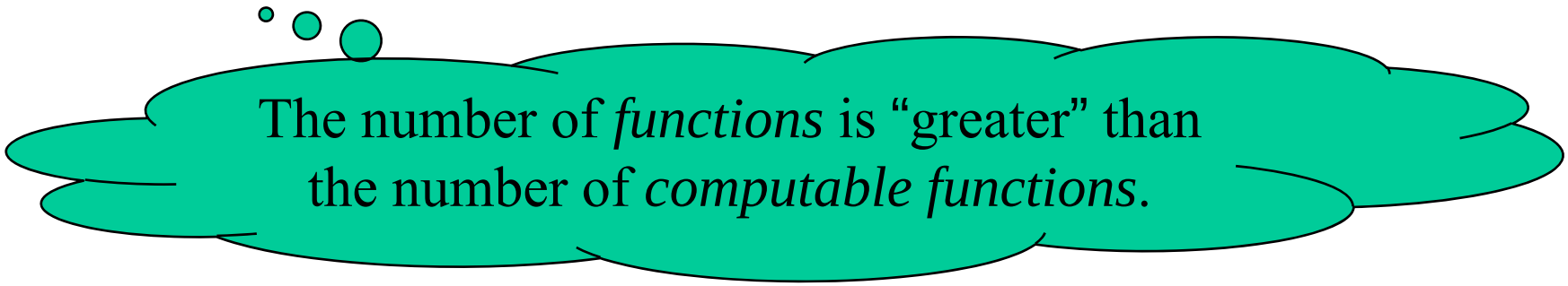
(two functions $f_1()$ and $f_2()$ are different if there exists an input x such that $f_1(x) \neq f_2(x)$.)

$$\Downarrow$$

$$\text{diag} \notin F_1$$

That is, the function diag is not computable.

End of proof



Diagonalization

Given a set G of functions, construct a function g which does not belong to G.

対角線論法

可算無限集合：　自然数全体の集合との間に1対1対応がある集合のこと.
可算集合：有限または可算無限である集合のこと.
　　つまり，1つずつ要素を取り出してきて，もれなく書き並べられるもの

例1. 正の偶数全体の集合Eは可算無限である.
　自然数全体の集合Nの要素 i と，Eの要素 $2i$ を対とする1対1対応がある.
例2. 整数全体の集合Zは可算無限である.
　1対1対応がある. または，$Z=\{0, 1, -1, 2, -2, 3, -3, ...\}$と列挙できる.
例3. 有理数全体の集合は可算無限である.（なぜか？）

定理：実数全体の集合Rは非可算である.

Diagonalization

Enumerable infinite set: **a set with one-to-one correspondence with the set of all natural numbers**

Enumerable set: finite or enumerable infinite set.

that is, a set whose elements are enumerable one by one.

Ex.1. The set *E* of all even positive integers is enumerable infinite.

one-to-one correspondence between an element *i* of the set of all natural numbers and an element *2i* of the set *E*

Ex.2. The set Z of all integers is enumerable infinite.

We can enumerate them as Z={0, 1, -1, 2, -2, 3, -3, ...}.

Ex.3. The set *R* of all rational numbers is enumerable infinite.（Why?）

Theorem: The set *R* of all real numbers is not enumerable.

定理：実数全体の集合Rは非可算である.

0以上1未満の実数全体の集合Sが非可算であることを対角線論法で証明する.
可算であると仮定すると，すべての要素を書き並べることができる：

$0.a_{11}a_{12}a_{13}...$
$0.a_{21}a_{22}a_{23}...$
$0.a_{31}a_{32}a_{33}...$
$0.a_{41}a_{42}a_{43}...$

$0.a_{k1}a_{k2}a_{k3}...$　ただし，$a_{ij} \in \{0,1,...,9\}$

$0.a_{11}a_{12}a_{13}...$
$0.a_{21}a_{22}a_{23}...$
$0.a_{31}a_{32}a_{33}...$
$0.a_{41}a_{42}a_{43}...$

$0.a_{k1}a_{k2}a_{k3}... \ a_{kk}$

上の並びで対角線上にある数に注目し，新たな無限小数

$x = 0.b_1b_2b_3...$

を作る．ここで，

if a_{kk}=1 then $b_k = 2$ else b_k=1

としてb_kを定める.
このように作られた無限小数は明らかに0と1の間の実数である.
しかし，作り方から，上に列挙したどの要素とも等しくない（対角線の所で必ず異なる）.
つまり，xはSに属さないことになり，矛盾である.
したがって，Sが可算であるという仮定に誤りがある.

Theorem: The set *R* of all real numbers is not enumerable.

Using the diagonalization we prove that the set S of all real numbers between 0 and 1 is not enumerable. By contradiction, we assume that it is enumerable:

$0.a_{11}a_{12}a_{13}...$
$0.a_{21}a_{22}a_{23}...$
$0.a_{31}a_{32}a_{33}...$
$0.a_{41}a_{42}a_{43}...$

$0.a_{11}a_{12}a_{13}...$
$0.a_{21}a_{22}a_{23}...$
$0.a_{31}a_{32}a_{33}...$
$0.a_{41}a_{42}a_{43}...$

$0.a_{k1}a_{k2}a_{k3}... \; a_{kk}$

$0.a_{k1}a_{k2}a_{k3}...$ where $a_{ij} \in \{0, 1, ..., 9\}$

Define a new real number x by collecting those digits in the diagonal

$x = 0.b_1b_2b_3...$

where b_k is defined by

if $a_{kk}=1$ then $b_k = 2$ else $b_k=1$

The number x defined above is obviously between 0 and 1, but it is different from any number listed above since it is different at its diagonal position.
That is, x does not belong to S, which is a contradiction.
Therefore, our assumption that S is enumerable is wrong.

例2.17 Haltの計算不可能性の証明の中で用いたプログラムX

```
prog X(input w: Σ*): Σ*;
label LOOP;
begin
   if H (w, w) then LOOP: goto LOOP
                  else  halt(0) end-if
end.
```

f_X: プログラムXが計算する関数

$f\_a_i(a_i) = \perp$ のとき, $\neg\mathrm{Halt}(a_i, a_i)$

$\therefore\ f\_\mathrm{X}(a_i) = 0$

$f\_a_i(a_i) \neq \perp$ のとき, $\mathrm{Halt}(a_i, a_i)$

$\therefore\ f\_\mathrm{X}(a_i) = \perp$

つまり, $f\_\mathrm{X} = f\_a_i$ となる f_a_i は
計算可能な関数の集合 F_1 の中に存在しない.

★プログラムの個数は可算無限だが、関数の個数は非可算無限

Ex.2.17 Program X used in the proof of incomputability of Halt

```
prog X(input w: Σ*): Σ*;
label LOOP;
begin
   if H (w, w) then LOOP: goto LOOP
                  else  halt(0) end-if
end.
```

*f*_X: function computed by the program X

if $f\_a_i(a_i) = \perp$ then $\neg \mathrm{Halt}(a_i, a_i)$

$\therefore f\_\mathrm{X}(a_i) = 0$

if $f\_a_i(a_i) \neq \perp$ then, $\mathrm{Halt}(a_i, a_i)$

$\therefore f\_\mathrm{X}(a_i) = \perp$

That is, there is no function f_a_i in the set F_1 of functions such that $f\_\mathrm{X} = f\_a_i$.

★The number of programs is enumerable, while the number of functions is not.