I216 Computational Complexity and Discrete Mathematics

by
Prof. Ryuhei Uehara
and
Prof. Atsuko Miyaji

I216 計算量の理論と離散数学

上原隆平、宮地充子

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)!

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

Computational Complexity

- Goal 1:
  - "*Computable Function/Problem/Language/Set*"
    - Technical terms;

      computability, diagonalization
- Goal 2:
  - How can you show "*Difficulty of Problem*"
    - Technical terms;

      The class NP, P≠NP conjecture, NP-hardness, reduction

Computational Complexity

- ゴール1:
  - “計算可能な*関数/問題/言語/集合*”
    - 関連する専門用語;

      計算可能性、対角線論法
- ゴール2:
  - 「問題の困難さ」を示す方法を学ぶ
    - 関連する専門用語;

      クラスNP, P≠NP予想, NP困難性, 還元

Text book

"Introduction to the Theory of Computation"

Michael Sipser, PWS Publishing Company, 1997.

Part 1 (Chaps. 1, 2)
Automata and Languages
… I118 Graphs & Automata

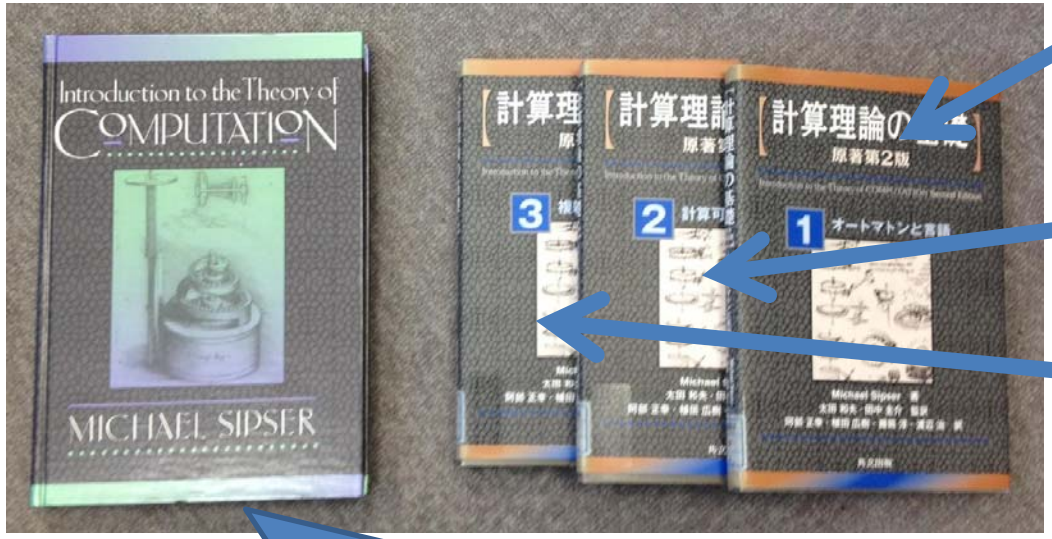
Part 2 (Chaps. 3, 4, 5, 6)
Computability Theory
… 1st Goal

Part 3 (Chaps. **7**, 8, 9, 10)
Complexity Theory
… 2nd Goal

教科書

“Introduction to the Theory of Computation”

Michael Sipser, PWS Publishing Company, 1997.



英語版は3rd editionが出ているけれど…

第1部 (1, 2章)
オートマトンと言語
… I118 グラフとオートマトン

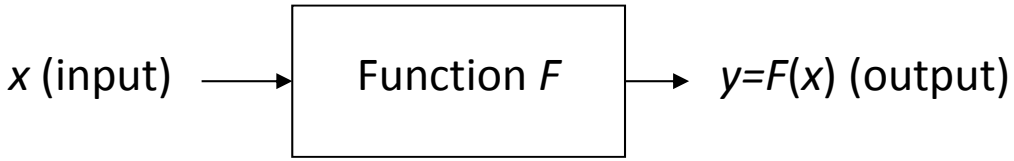
第2部 (3, 4, 5, 6章)
計算可能性の理論
… ゴール1

第3部 (**7**, 8, 9, 10章)
複雑さの理論
… ゴール2

1. Function/Problem/Language/Set

1. What are *functions/problems/languages/sets*?

Function: correspondence between Input & Output



Point: For the same $x=x_1$ and $x=x_2$, we always have $F(x_1)=F(x_2)$.

Problem: computation of a function

E.g. **Sorting Problem**

input: sequence of natural numbers $a_1, a_2, \dots, a_n$

output: increasing order $a_{i1}, a_{i2}, \dots, a_{in}$.

1. 関数・問題・言語・集合

1. 関数・問題・言語・集合とは何か？

関数: 入力と出力の対応関係

ポイント: 同じ入力 $x=x_1$ と $x=x_2$ に対して同じ出力 $F(x_1)=F(x_2)$.

問題: 関数を計算すること

例.　ソート問題（並べ替え）

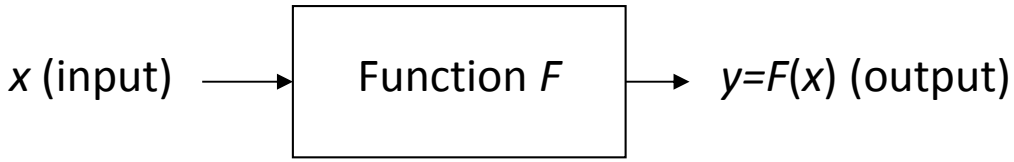
入力: 自然数の列 $a_1, a_2, \dots, a_n$

出力: 小さい順に並べ替えた列 $a_{i1}, a_{i2}, \dots, a_{in}$.

1. Function/Problem/Language/Set

1. What are *functions/problems/languages/sets*?

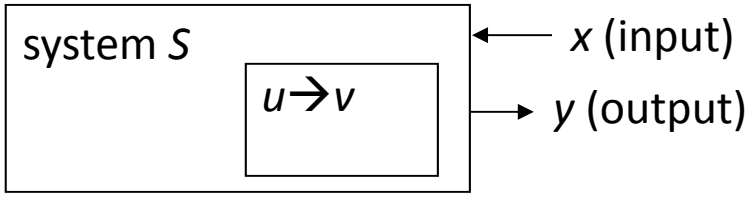
Function: correspondence between Input & Output



Point: For the same $x=x_1$ and $x=x_2$, we always have $F(x_1)=F(x_2)$.

Problem: computation of a function

E.g. **Performance of a computer system S**



problem of computing a function to map (x,u) to (y,v)

1.関数・問題・言語・集合

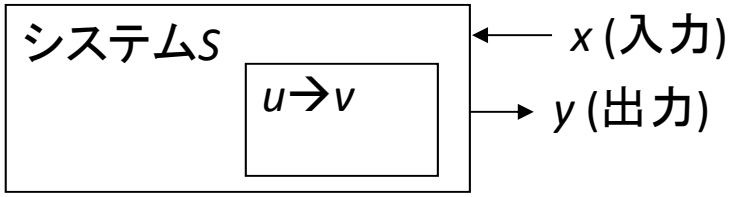
1. 関数・問題・言語・集合とは何か？

関数: 入力と出力の対応関係

ポイント: 同じ入力 $x=x_1$ と $x=x_2$ に対して同じ出力 $F(x_1)=F(x_2)$.

問題: 関数を計算すること

例. コンピュータシステム*S*のふるまい

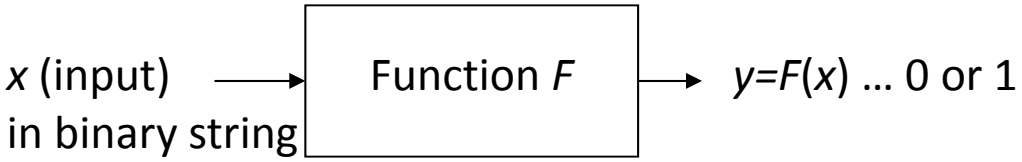


(*x*,*u*)から(*y*,*v*)への関数を計算する問題とみなしてよい

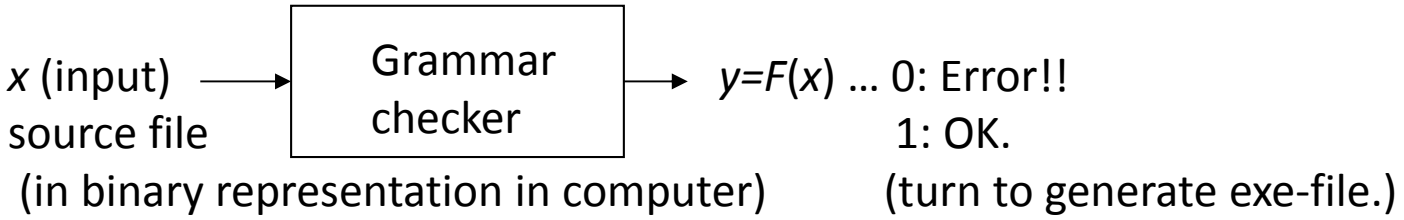
1. Function/Problem/Language/Set

1. What are *functions/problems/languages/sets*?

Typical *Function*: computation of 0/1 for given *x* in binary string



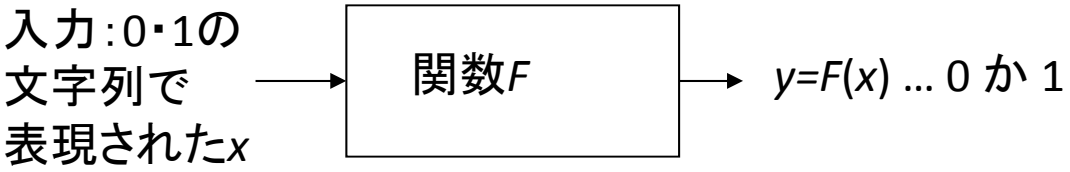
E.g. **Grammar check in C compiler**



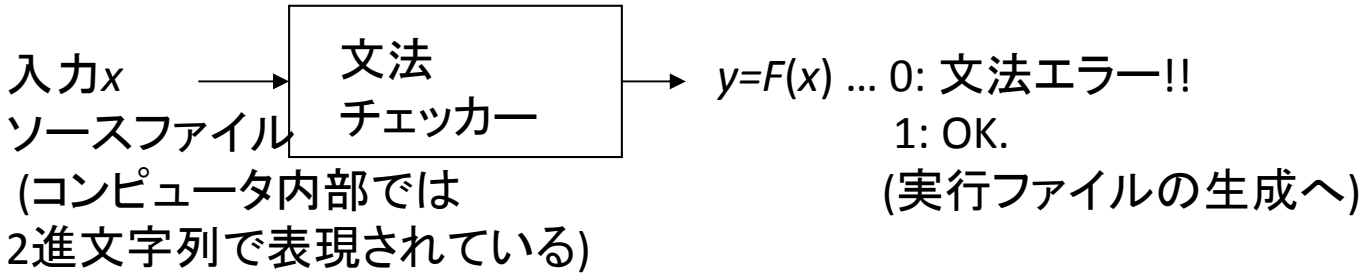
1.関数・問題・言語・集合

1. 関数・問題・言語・集合とは何か？

典型的な*関数*: 2進文字列で表現された x に対して、
0か1を計算する



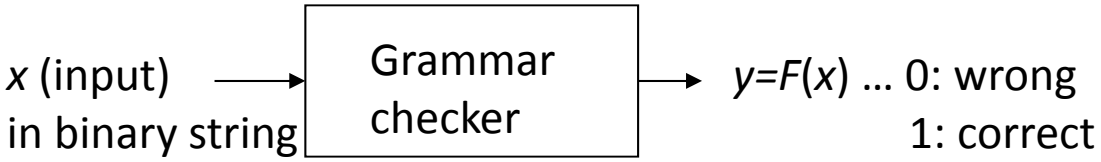
例. **Cコンパイラの文法チェック**



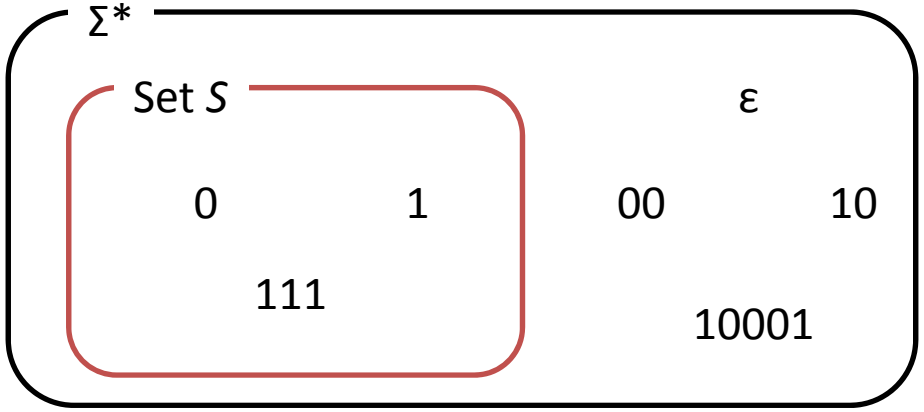
1. Function/Problem/Language/Set

1. What are *functions/problems/languages/sets*?

Language recognition: decide if given *x* is a correct "word" (for given grammar)



Set recognition: decide if given *x* is in some *S* or not



Σ*={ε,0,1,00,01,10,11,…}

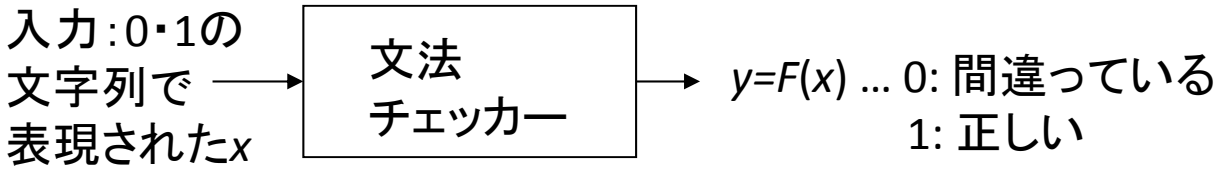
ε: empty string of length 0

S={0,1,111,…}

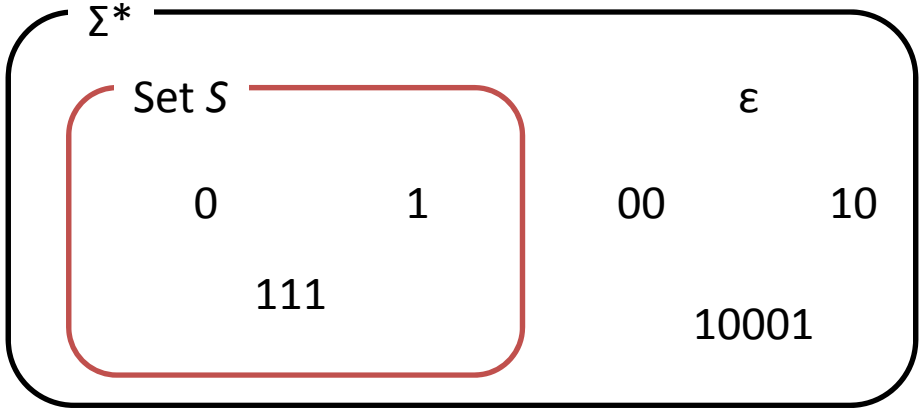
1.関数・問題・言語・集合

1. 関数・問題・言語・集合とは何か？

*言語*の認識問題: 与えられた *x* が（与えられた文法のもとで）正しい「語」であるかどうかを判定する



*集合*の認識問題: 与えられた *x* が集合*S* の要素かどうかを判定する



Σ*={ε,0,1,00,01,10,11,…}

ε: 長さ0の空文字列

S={0,1,111,…}

1. Function/Problem/Language/Set

[Note] In this class, we only consider *binary functions*

$f(x):\Sigma^* \rightarrow \Sigma$, where $\Sigma=\{0,1\}$ (and $\Sigma^*=\{\varepsilon,0,1,00,01,10,11,\ldots\}$)

…Why?

[Reason 1] Any data can be represented by *binary* strings.
Ex.1: Any alphabet (a-z,0-9,…) can be encoded in
8bit binary string by ASCII code.
Ex.2: Any music can be encoded in mp3 format.
Ex.3: Any photo can be encoded in jpeg format.
…

1.関数・問題・言語・集合

[注意] この講義では以下の2値関数だけを考える.

$f(x):\Sigma^* \rightarrow \Sigma$, where $\Sigma=\{0,1\}$ (and $\Sigma^*=\{\varepsilon,0,1,00,01,10,11,\ldots\}$)

…なぜ?

[理由1] どんなデータもバイナリ文字列で表現できる.
例1: アルファベット(a-z,0-9,…) はASCIIコードで8ビットで
表現できる
例2: 音楽は mp3 フォーマットで符号化できる
例3: 写真は jpeg フォーマットで符号化できる
…

1. Function/Problem/Language/Set

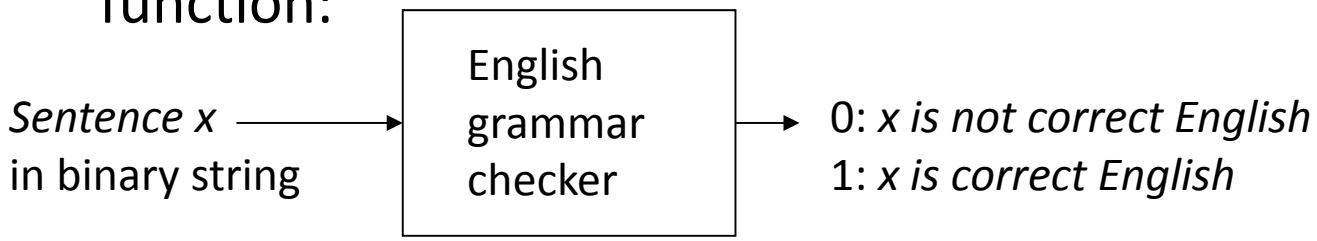
[Note] In this class, we only consider *binary functions*

$f(x)$:$\Sigma^* \rightarrow \Sigma$, where $\Sigma=\{0,1\}$ (and $\Sigma^*=\{\varepsilon,0,1,00,01,10,11,\ldots\}$)

…Why?

[Reason 2] Any language can be *recognized* by some binary function as follows.

Ex.1: any *English sentence* be recognized by the following function:



1.関数・問題・言語・集合

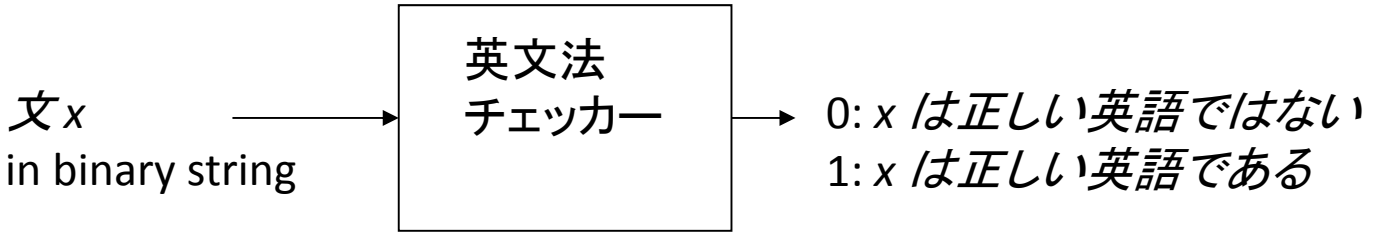
[注意] この講義では以下の2値関数だけを考える.

$f(x)$:$\Sigma^* \rightarrow \Sigma$, where $\Sigma=\{0,1\}$ (and $\Sigma^*=\{\varepsilon,0,1,00,01,10,11,\ldots\}$)

…なぜ?

[理由2] どんな言語も2値関数で認識できる

例1:「英文」は次の関数で認識できる



1. Function/Problem/Language/Set

[Note] In this class, we only consider *binary functions*

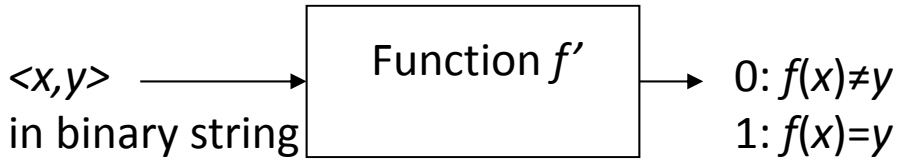
$f(x):\Sigma^* \rightarrow \Sigma$, where $\Sigma=\{0,1\}$ (and $\Sigma^*=\{\varepsilon,0,1,00,01,10,11,\ldots\}$)

…Why?

[Reason 3] Any function can be *interpreted* into another binary function in a sense.

Ex. : for any $f(x)=y$ over integers can be represented as…

- Any integer x can be represented in binary number.
- Computation of "$f(x)=y$" can be represented by the following yes/no type function f'



We consider that f' is as hard as f

1.関数・問題・言語・集合

[注意] この講義では以下の2値関数だけを考える.

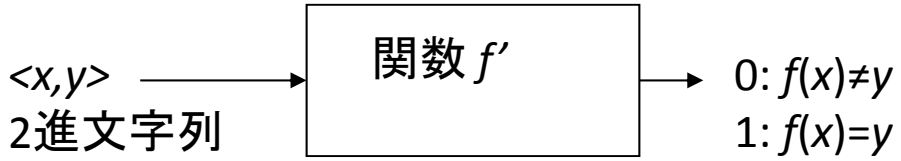
$f(x):\Sigma^* \rightarrow \Sigma$, where $\Sigma=\{0,1\}$ (and $\Sigma^*=\{\varepsilon,0,1,00,01,10,11,\ldots\}$)

…なぜ?

[理由3] どんな関数でも, ある意味で同等の
バイナリ関数に「翻訳」できる

例: 整数上の関数 $f(x)=y$ は次のように翻訳できる:

- 任意の整数 x は2進数で表現できる
- "$f(x)=y$" の計算は次の yes/no タイプの関数 f' に翻訳できる



関数 f' と関数 f は
同程度に難しいと考える

1. Function/Problem/Language/Set

[Note] In this class, we only consider *binary functions*

$f(x):\Sigma^* \rightarrow \Sigma$, where $\Sigma=\{0,1\}$ (and $\Sigma^*=\{\varepsilon,0,1,00,01,10,11,\ldots\}$)

…Why?

[Exercise 1] How can you *interpret* the following sorting problem into binary function?

Sorting Problem

input: sequence of natural numbers $a_1, a_2, \ldots, a_n$

output: increasing order $a_{i1}, a_{i2}, \ldots, a_{in}$.

1.関数・問題・言語・集合

[注意] この講義では以下の2値関数だけを考える.

$f(x)$:Σ* → Σ, where Σ={0,1} (and Σ*={ε,0,1,00,01,10,11,…})

…なぜ?

[演習問題1] 以下のソート問題はどうやって2値関数に翻訳すればよいだろう?

ソート問題(並べ替え)

入力: 自然数の列 $a_1, a_2, \ldots, a_n$

出力: 小さい順に並べ替えた列 $a_{i1}, a_{i2}, \ldots, a_{in}$.

2. Algorithm

2. What are Algorithms?

***Algorithm* for solving a problem**

– method for computing an output specified in the problem.

- *What* to be computed?
- *How* to compute?

> **different**

E.g. Problem of calculating a root of a quadratic equation

input : rational numbers *a, b, c*

output: an *x* that satisfies $ax^2+bx+c=0$

– Output is clearly defined but how can we find it?

"Algorithm = method"

➔algorithm＝a method that can be realized as a *program*

program … depends on "machine model"

→ history of "computability"

2. アルゴリズム

2. アルゴリズムとは何か？

問題を解くための「アルゴリズム」

- 問題で指定された出力を計算するための方法
  - *何を*計算するか？
  - *どのように*計算するか？

＞ この二つは異なる

例．2次方程式の根を求める問題

入力：有理数 a, b, c

出力：方程式 $ax^2+bx+c=0$ を満たす x

- 出力の定義は明確だが、どのようにして求めたらよいのか？

“アルゴリズム = 方法"

➔アルゴリズム＝「*プログラム*」として記述できる「方法」

プログラム …「マシンモデル」に依存

→ “計算可能性”の歴史

3. Machine model & computability

3. Studies on what is a 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":

This is the set of "computable" function!

3. マシンモデルと計算可能性

3. 「計算」とは何か？

“「計算可能」な関数とは?”

クリーネによる帰納的関数理論

チューリングによるチューリング機械の理論

➔帰納的関数全体の集合

＝チューリング機械で計算できる関数全体の集合

チャーチの提唱による「計算可能」の定義:

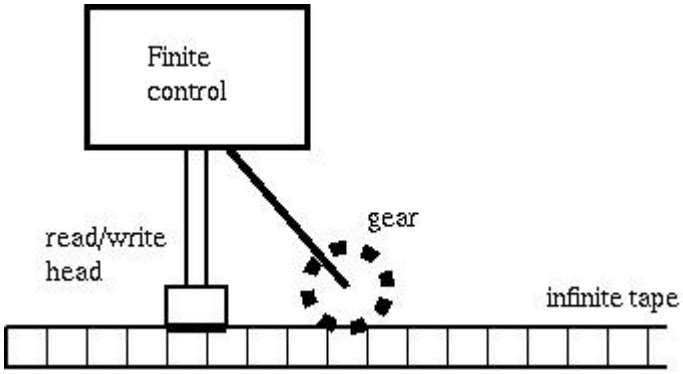
この集合を「計算可能」な関数としよう!

3. Machine model & computability

3. Studies on what is a computation.

Turing machine model consists of

- finite control
- infinitely long tape
  with read/write head



Initially,

1. tape consists of “letters”
2. the head is located at the leftmost position

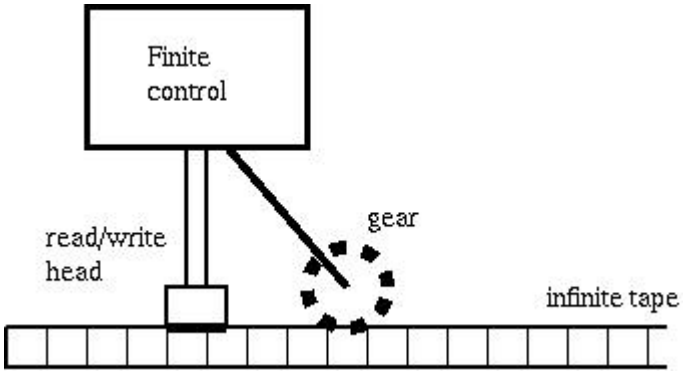
3. マシンモデルと計算可能性

3.「計算」とは何か？

チューリングマシンモデルの構成要素

- 有限制御部
- 無限長のテープ

  読み書きヘッド



初期状態：

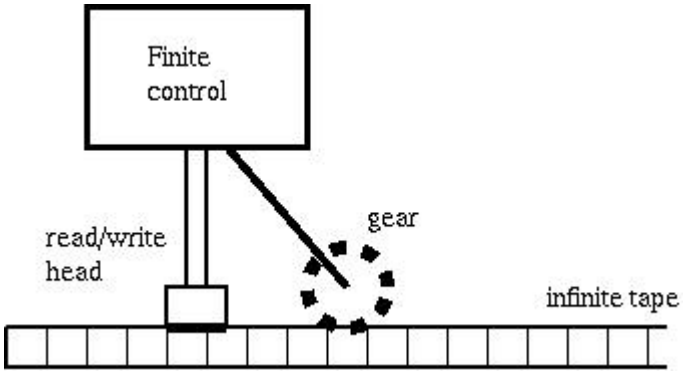
1. テープには「文字」列
2. ヘッドは一番左にある

3. Machine model & computability

3. Studies on what is a computation.

Turing machine model consists of

- finite control
- infinitely long tape
  with read/write head



It moves as follows;

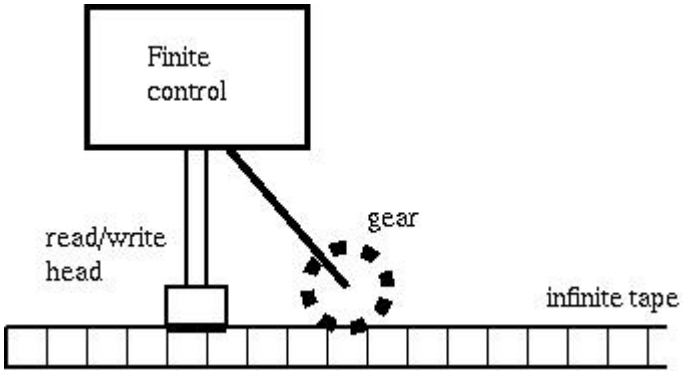
1. r/w head reads the letter
2. <u>according to the letter</u>,
   1. rewrite the letter
   2. move the head to the left or right neighbor
3. change the state of control and go to step 1 until it comes to “accept” state or “reject” state.

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モデルの構成要素

- 有限制御部
- 無限長のテープ
  読み書きヘッド



動作手順は以下の通り:

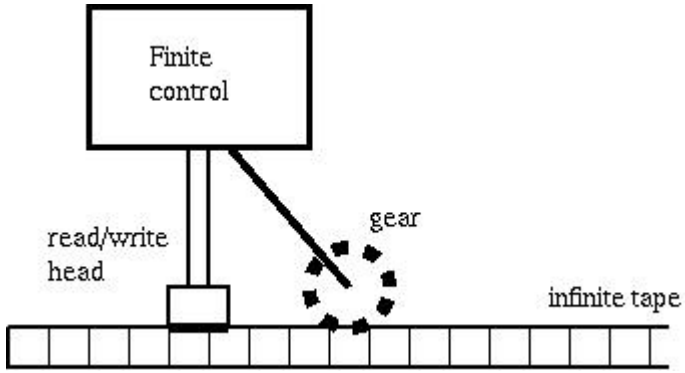
1. 読/書ヘッドが1文字読む
2. 文字の内容に応じて
   1. 文字を上書き
   2. ヘッドを右か左に一つずらす
3. 有限制御部の状態を変える.「受理」状態か「拒否」状態になっていなかったらステップ1に戻る.

3. Machine model & computability

3. Studies on what is a computation.

Turing showed that Turing machine is *universal*

- it can simulate *any computation*
- it has the same computation power as recent supercomputers! (if you do not mind the *speed*)

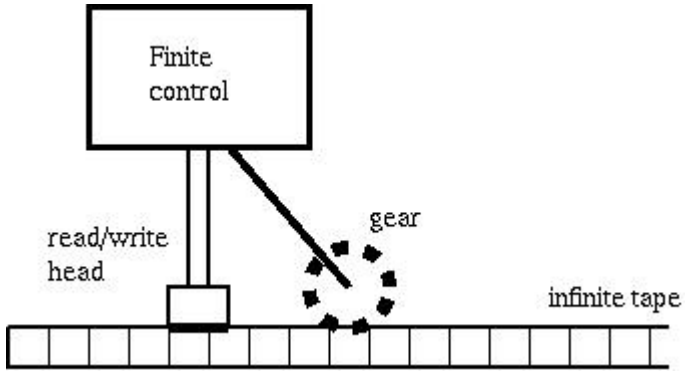


3.マシンモデルと計算可能性

3.「計算」とは何か？

チューリングはチューリングマシンの万能性を証明

- *どんな計算*でも模倣できる
- 計算能力は昨今のスーパーコンピュータとも本質的に同じである! (計算時間を無視すれば)



3. Machine model & computability

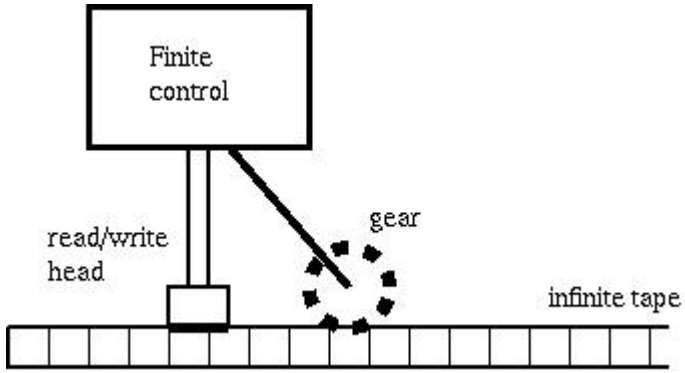
3. Studies on what is a computation.

Turing showed that Turing machine is *universal*

E.g. 1.

k letters tape = binary tape;

each letter can be encoded by a binary string.



3.マシンモデルと計算可能性

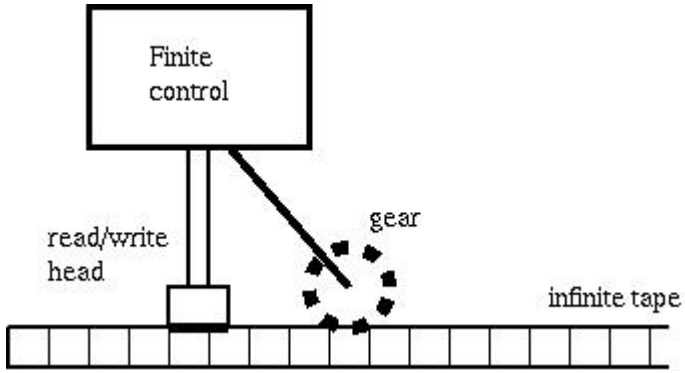
3.「計算」とは何か？

チューリングはチューリングマシンの万能性を証明

例1.

テープ上の文字の種類がk = 文字は2進;

それぞれの文字を2進文字列で符号化.



3. Machine model & computability

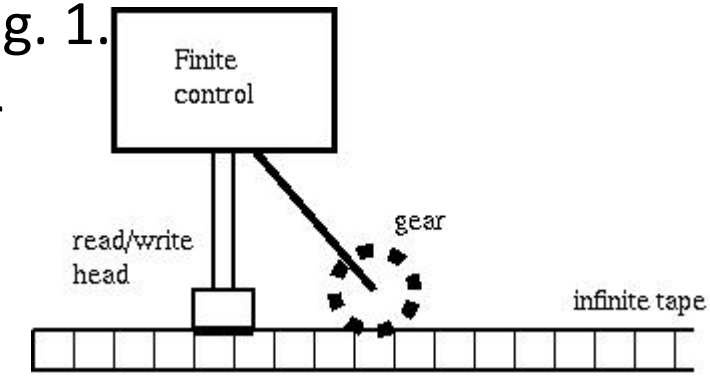
3. Studies on what is a computation.

Turing showed that Turing machine is *universal*

E.g. 2.

infinite *both* sides = infinite just right side;

1. "fold" the tape at the center
2. for the four letters, apply E.g. 1.
3. finite control has a state for "which tape?"



3.マシンモデルと計算可能性

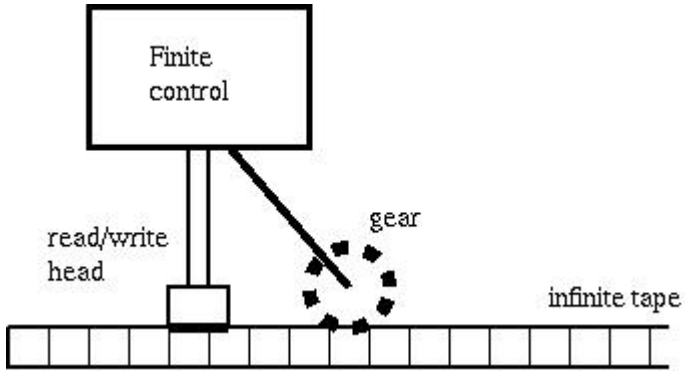
3.「計算」とは何か？

チューリングはチューリングマシンの万能性を証明

例2.

テープは両側無限長 = テープは右側だけ無限長;

1. テープを中央で「折り返」して重ねる
2. 4通りの文字に例1を適用
3. 有限状態部で「どちらの
   テープか」を管理



3. Machine model & computability

3. Studies on what is a computation.

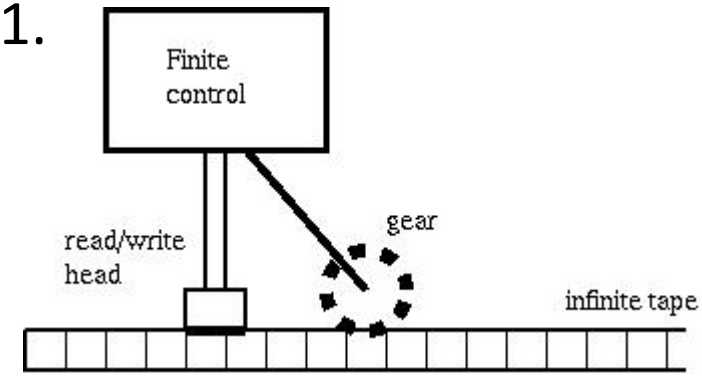
Turing showed that Turing machine is *universal*

E.g. 3.

k (binary) tapes = 1 binary tape;

1. "stack" the *k* tapes onto a tape
2. for the 2^k letters, apply E.g. 1.

(each head position is stored at left end)



3.マシンモデルと計算可能性

3.「計算」とは何か？

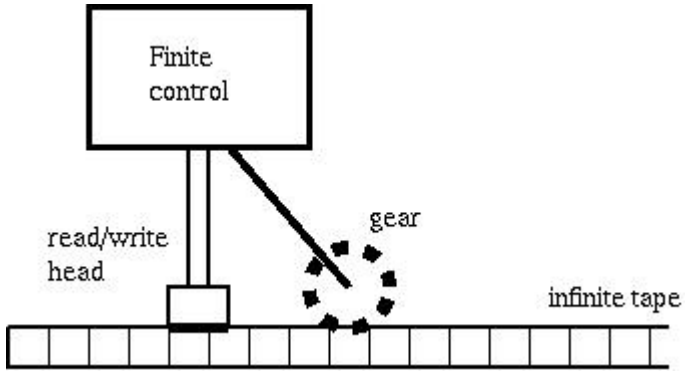
チューリングはチューリングマシンの万能性を証明

例3.

*k*本の(2進)テープ = 1本の2進テープ;

1. *k*本のテープを1本のテープに「積み重ね」る
2. 2^k 種類の文字に例1を適用

各テープヘッドの位置は別途
テープの左端に記録



3. Machine model & computability

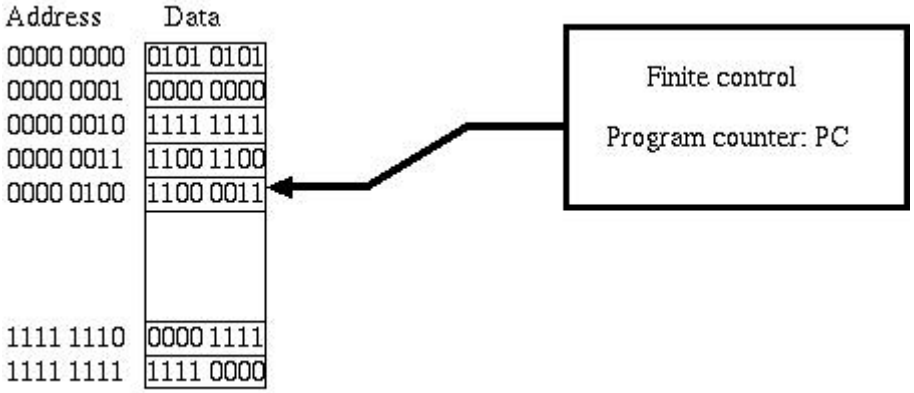
3. Studies on what is a computation.

Turing showed that Turing machine is *universal*

E.g. 3'.

k tapes = so-called "von Neumann computer"

= *k* bit computer on your desktop



3.マシンモデルと計算可能性

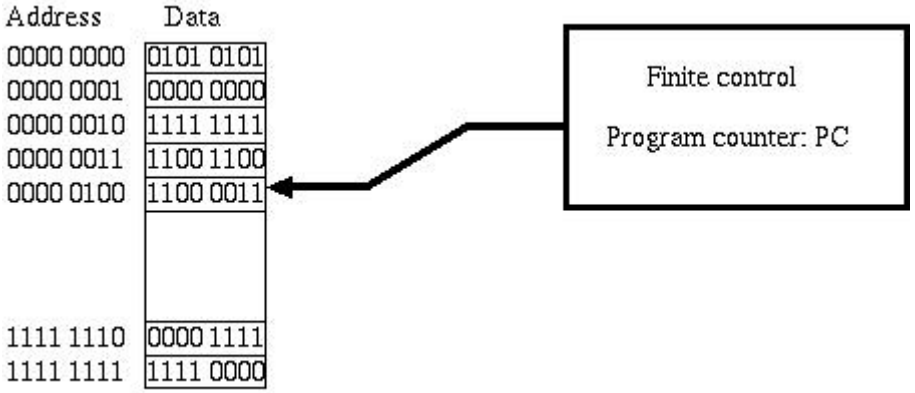
3.「計算」とは何か？

チューリングはチューリングマシンの万能性を証明

例3'.

*k*テープ = いわゆる「フォン・ノイマン式」計算機

=普通の *k* ビットのコンピュータ



3. Machine model & computability

3. Studies on what is a computation.

Turing showed that Turing machine is *universal*

Two crucial ideas;

1. A Turing machine *T* can be encoded as a (loooong) binary string that consists of
    1. string that represents the finite control
    2. string that represents the contents on the tape
2. A universal Turing machine *U* simulates any Turing machine *T* represented in the binary string.

    (The machine *U* is a kind of “simulator”)

3.マシンモデルと計算可能性

3. 「計算」とは何か？

チューリングはチューリングマシンの万能性を証明

二つの重要なアイデア;

1. チューリングマシン *T* は以下の二つの内容をつないだ(長—い)2進文字列で表現することができる.
    1. 有限制御部を書き下した文字列
    2. テープの内容を書き下した文字列
2. 万能チューリングマシン *U* は2進文字列で表現された任意のチューリングマシン *T* の動作を模倣できる

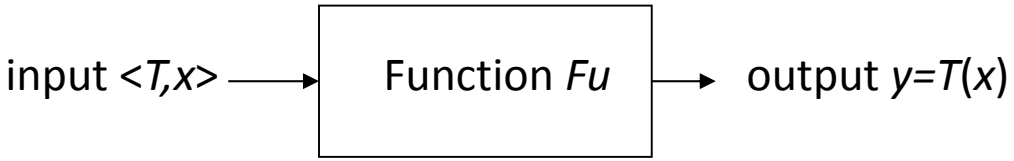
    (マシン *U* は一種の「シミュレータ」)

3. Machine model & computability

3. Studies on what is a computation.

Turing showed that Turing machine is *universal*

In the term of "function" *Fu*:



input <*T,x*>: that represents "the code of *T*" and "the code *x* of the input to *T*"
output *T(x)*: the output of *T* with its input *x*

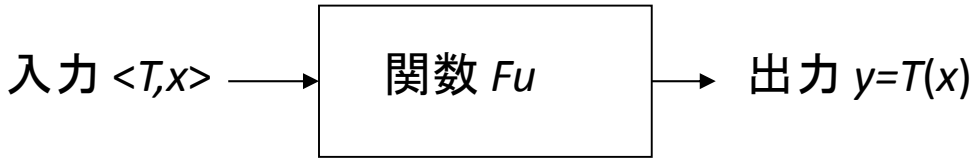
[Theorem] (Turing 1936)
There is a (universal) Turing machine *U* such that it computes *T(x)*
for any given Turing machine *T* and its input *x*.

3.マシンモデルと計算可能性

3. 「計算」とは何か？

チューリングはチューリングマシンの万能性を証明

「関数」*Fu* を使って示せば:



入力 <*T,x*>: 「*T* を符号化したもの」と「*T* への入力*x*」を符号化したものの連結
出力 *T(x)*: マシン *T* に入力 *x* を与えたときの出力

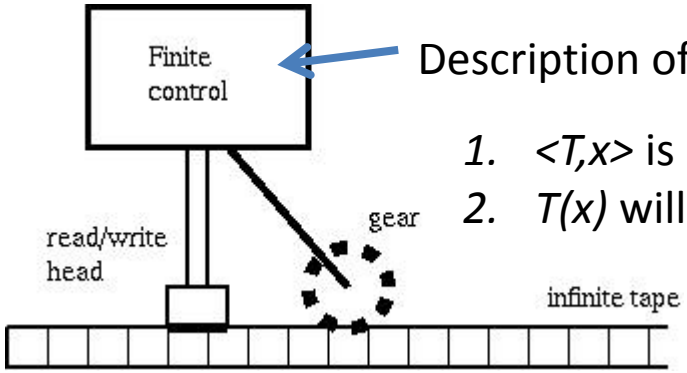
[定理] (チューリング1936)
与えられた任意のチューリングマシン*T* とそれへの入力 *x* に対して,
T(x) を計算(模倣)する(万能)チューリングマシン*U*が存在する.

3. Machine model & computability

3. Studies on what is a computation.

Turing showed that Turing machine is *universal*

In the term of “Turing machine”:



1. *<T,x>* is encoded and written on the tape at first
2. *T(x)* will be written on the tape by *U* in finite time
   (if *T(x)* does not halt, so is U.)

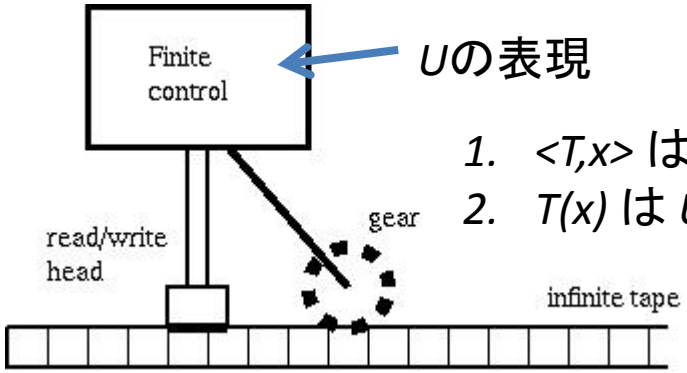
input *<T,x>*: that represents “the code of *T*” and “the code *x* of the input to *T*”
output *T(x)*: the output of *T* with its input *x*

3.マシンモデルと計算可能性

3.「計算」とは何か？

チューリングはチューリングマシンの万能性を証明

「チューリングマシン」モデルで表現すると:



*U*の表現

1. *<T,x>* は符号化されて最初にテープに書かれている
2. *T(x)* は *U* によって有限時間内にテープに書かれる
   (*T(x)* が停止しないなら，*U*も同様．)

入力 *<T,x>*:「*T* を符号化したもの」と「*T* への入力*x*」を符号化したものの連結
出力 *T(x)*: マシン *T* に入力 *x* を与えたときの出力

4. Unsolvability and Diagonalization

4. Incomputable problem

The following problem cannot be solved by any Turing machine:

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"

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