

1/18

## Chap.4 Computational Complexity

### 4.1. Survey on Theory of Computational Complexity

“Computable?” → “How much cost is required for computation?”  
 Computational Complexity Theory

- (1) Studies on upper bound of computational cost
- (2) Studies on lower bound of computational cost
- (3) Structural studies on hardness of computation

**(1) Studies on upper bound of computational cost**  
 Algorithm Theory: design of efficient algorithms  
 Suppose we have an algorithm  $A$  which solves a problem  $X$  in at most time  $T(n)$  for any input of size  $n$ . Then, an **upper bound** on the time complexity of the algorithm  $A$  is  $T(n)$ .  
 (asymptotic worst case time complexity)

1/18

## 第4章 計算の複雑さ入門

### 4.1. 計算の複雑さの理論概観

「計算可能か？」 → 「どの程度の計算コストで計算可能か？」  
 計算の複雑さの理論 (Computational Complexity Theory)

- (1) 計算量の上限に関する研究
- (2) 計算量の下限に関する研究
- (3) 計算の難しさについての構造的な研究

**(1) 計算量の上限に関する研究**  
 効率のよいアルゴリズムの設計 (アルゴリズム理論)  
 ある問題  $X$  に対して、それを解くアルゴリズム  $A$  があり、サイズ  $n$  の **どんな問題例** に対しても  $A$  の時間計算量が  $T(n)$  以内であるとき、アルゴリズム  $A$  の時間計算量の **上限** は  $T(n)$   
 (最悪時の漸近的な時間計算量)

2/18

**(2) Studies on lower bound of computational cost**  
 If any algorithm for a problem  $X$  takes time  $T(n)$  in the worst case, a lower bound on the time complexity of the problem  $X$  is  $T(n)$ .  
 •  $\mathcal{P} \neq \mathcal{NP}$  conjecture  
 • Robustness of crypto system

**(3) Structural studies on hardness of computation**  
 Studies to characterize hardness in the level of “xx-hardness”  
 hierarchical structure depending on the hardness

2/18

**(2) 計算量の下限に関する研究**  
 問題  $X$  に対する **どんなアルゴリズム** も最悪の場合には  $T(n)$  時間だけ必ずかかってしまうとき、問題  $X$  の時間計算量の下限は  $T(n)$ .  
 •  $\mathcal{P} \neq \mathcal{NP}$  予想  
 • 暗号システムの強さ

**(3) 計算の難しさについての構造的な研究**  
 “xx程度の難しさ”がもつ特徴について調べること。  
 難しさの程度による階層構造。

3/18

## 4.2 Measuring Computation Time

### 4.2.1 Revisiting Programs in the Standard form

**Programs in the standard form**

```

prog program name (input ...);
var pc:  $\Sigma^* \rightarrow \Sigma; \dots; \Sigma^*$ ;
begin
  pc:=1;
  while pc≠0 do
    case pc of
      1: (statement);
      2: (statement);
      3: (statement);
      .....
      k: (statement);
    end-case
  end-while;
  halt(variable of type  $\Sigma^*$ );
end.
    
```

Program consists of a while-loop

Each statement must be either if comparison then  $pc:=k_1$  else  $pc:=k_2$  end-if or substitution;  $pc:=k$ ;

3/18

## 4.2. 計算時間の計り方

### 4.2.1. 標準形プログラム再考

**標準形プログラム**

```

prog プログラム名 (input ...);
var pc:  $\Sigma^* \rightarrow \Sigma; \dots; \Sigma^*$ ;
begin
  pc:=1;
  while pc≠0 do
    case pc of
      1: (文);
      2: (文);
      3: (文);
      .....
      k: (文);
    end-case
  end-while;
  halt( $\Sigma^*$ 型の変数);
end.
    
```

全体は大きな while-loop

各(文)の形は  
 - if 比較文 then  $pc:=k_1$  else  $pc:=k_2$  end-if  
 - 代入文;  $pc:=k$ ;  
 のいずれか。

4/18

### 4.2 Measuring Computation Time

**Definition 4.1 (Computation time)**  
 A: program with  $k$  inputs in the standard form  
 $x_1, x_2, \dots, x_k$ : inputs to A

Single execution of while loop in A is “one step” in A.  
 The number of iterations of the while loop required before A halts is called **the computation time of A for inputs  $x_1, x_2, \dots, x_k$**  (in short, **computation time of  $A(x_1, x_2, \dots, x_k)$** ).

If A does not halt, its computation time is infinite.

$$time\_A(x_1, x_2, \dots, x_k) \equiv \text{computation time of } A(x_1, x_2, \dots, x_k)$$

$$time\_A(l) \equiv \max \{ time\_A(x_1, x_2, \dots, x_k) : \sum_{1 \leq i \leq k} |x_i| \leq l \}$$

4/18

### 4.2. 計算時間の計り方

#### 4.2.1. 標準形プログラム再考

**定義4.1. (計算時間の定義)**  
 A:  $k$ 入力標準形プログラム  
 $x_1, x_2, \dots, x_k$ : Aへの入力

Aのwhileループ1回り分の実行をAでの**1ステップ**という。  
 入力 $x_1, x_2, \dots, x_k$ に対してAが停止するまでに回るwhileループの回数を**Aの $x_1, x_2, \dots, x_k$ に対する計算時間**(略して $A(x_1, x_2, \dots, x_k)$ の計算時間)という。ただし、停止しないとき、計算時間は無限大。

$$time\_A(x_1, x_2, \dots, x_k) \equiv A(x_1, x_2, \dots, x_k) \text{の計算時間}$$

$$time\_A(l) \equiv \max \{ time\_A(x_1, x_2, \dots, x_k) : \sum_{1 \leq i \leq k} |x_i| \leq l \}$$

5/18

- Constraints to execute each statement in constant time  
 $u, u'$ : variable of type  $\Sigma$ ,  $v, v'$ : variable of type  $\Sigma^*$   
 $c$ : constant of type  $\Sigma$ ,  $s$ : constant of type  $\Sigma^*$

**(Substitution)**

(1)  $u := c$ ; (2)  $u := u'$ ;  
 (3)  $u := \text{head}(v)$ ; (4)  $u := \text{tail}(v)$ ;  
 (5)  $v := s$ ; (6)  $v := v'$ ; ??  
 (7)  $v := \text{right}(v)$ ; (8)  $v := \text{left}(v)$ ;  
 (9)  $v := u \# v$ ; (10)  $v := v \# u$ ;

**(Comparison)**

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

- comparison of the form  $v = v'$  is forbidden

5/18

- 各文が高々定数時間で実行できるための制約**  
 $u, u'$ :  $\Sigma$ 型の変数,  $v, v'$ :  $\Sigma^*$ 型の変数  
 $c$ :  $\Sigma$ 型の定数,  $s$ :  $\Sigma^*$ 型の定数

**(代入文)** (1)  $u := c$ ; (2)  $u := u'$ ;  
 (3)  $u := \text{head}(v)$ ; (4)  $u := \text{tail}(v)$ ;  
 (5)  $v := s$ ; (6)  $v := v'$ ; ??  
 (7)  $v := \text{right}(v)$ ; (8)  $v := \text{left}(v)$ ;  
 (9)  $v := u \# v$ ; (10)  $v := v \# u$ ;

**(比較文)** (11)  $u = c$  (12)  $v = s$

- $v = v'$ の形の比較は禁止されている。

6/18

### 4.2.2. Time complexity of a program

The time complexity of a program is represented as a **function of input size** (length of an input string)

**Valid Encoding:**  
 Encoding into *at most constant times* larger than the original.

**Ex.4.5: Unary and binary representations**  
 Binary representation is a valid encoding in the standpoint of “size of a number is its number of bits”, but unary one is redundant.

6/18

### 4.2.2. プログラムの時間計算量

プログラムの時間計算量を**入力サイズ**の関数として表現  
 (入力文字列の長さ)

**妥当なコード化:**  
 元の対象のサイズに定数倍の範囲内で忠実なコード化

**例4.5: 1進表記と2進表記**  
 「数のサイズはその桁数」との立場では  
 2進表記は妥当なコード化であるが、  
 1進表記は冗長なコード化

7/18

**Definition 4.3:** For functions  $f$  and  $g$  on natural numbers, if  $\exists c, d > 0, \forall n [f(n) \leq c g(n) + d]$  then we say  $f$  is in the order of  $g$  and denote it by  $f = O(g)$ .

**Remark:** the constants  $c$  and  $d$  must be determined independently of  $n$ .

**Theorem 4.1:** The followings hold for any functions  $f, g$  and  $h$  on natural numbers:

- $\forall n [f(n) \leq g(n)] \rightarrow f = O(g)$
- $\exists c > 0, \forall n [f(n) \leq c g(n)] \rightarrow f = O(g)$
- $[f = O(g) \text{ and } g = O(h)] \rightarrow f = O(h)$

7/18

**定義4.3:** 自然数上の任意の関数  $f$  と  $g$  に対して、もし  $\exists c, d > 0, \forall n [f(n) \leq c g(n) + d]$  ならば  $f$  はオーダー  $g$  であるとい  $f = O(g)$  と書く。

**注意:** 定数  $c$  と  $d$  は  $n$  と独立でなければならない。

**定理4.1:** 自然数上の任意の関数  $f, g, h$  に対し、以下が成立:

- $\forall n [f(n) \leq g(n)] \rightarrow f = O(g)$
- $\exists c > 0, \forall n [f(n) \leq c g(n)] \rightarrow f = O(g)$
- $[f = O(g) \text{ and } g = O(h)] \rightarrow f = O(h)$

8/18

**4.2.3. Time complexity of a problem**

**Def.4.4.** Let  $\Phi$  be a computing problem and  $t$  be a function over natural numbers. If we have a program  $A$  to compute  $\Phi$  and some constants  $c$  and  $d > 0$  such that  $\forall l [time\_A(l) \leq ct(l) + d]$  then we say that  $\Phi$  is computable in  $O(t)$  time, or time complexity of  $\Phi$  is  $O(t)$ .

Note: We assume here that a computing problem is that of recognizing a set.

Intuitively

- problem  $\Phi$  is computable within time  $t$
- time complexity of  $A$  may be less than  $t$ .
- there may be a faster program to compute  $\Phi$  than  $A$  does.

8/18

**4.2.3. 問題の時間計算量**

**定義4.4.**  $\Phi$  を計算問題とし、 $t$  を自然数上の関数とする。いま  $\Phi$  を計算するプログラム  $A$  と定数  $c, d > 0$  が存在して、 $\forall l [time\_A(l) \leq ct(l) + d]$  ならば、 $\Phi$  は  $O(t)$  時間計算可能、あるいは  $\Phi$  の時間計算量は  $O(t)$  であるという。

注意: ここでは計算問題として、集合の認識問題を想定している。

直観的には「問題  $\Phi$  は  $t$  時間以下で計算可能」という意味。

(注1)  $A$  の時間計算量は  $t$  より低いかもしれない。  
(注2)  $A$  よりも速く  $\Phi$  を計算するプログラムがあるかもしれない。

9/18

**Ex.4.7. Time complexity of the problem determining primes**

**Prime-determining problem (PRIME)**  
Input: a natural number  $n$  (binary representation)  
Question: Is  $n$  prime?  
PRIME  $\equiv \{ [n] : n \text{ is prime} \}$

Stirling's Formula:  
 $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

```

prog Naive(input n);
begin
  for each i := 1 < i < n do
    if n mod i = 0 then reject end-if
  end-for;
  accept
end.
    
```

try to divide by numbers between 2 - n-1

$O(l^6)$  time algorithm has been developed in 2002!!

$\log n \cdot \log i$  time

$$time\_Naive(n) \leq \sum_{1 < i < n} (c \log n \log i + d) = c \log n \log n! + dn = O(n(\log n)^2)$$

When the length of  $n$  is  $l$ ,  $l$  is approximately  $\log n$ . So,  $time\_Naive = O(l^2)$ . Thus, time complexity of PRIME is  $O(l^2)$ .

9/18

**例4.7. 素数判定問題の時間計算量**

**素数判定問題 (PRIME)**  
入力: 自然数  $n$  (ただし、2進表記)  
質問:  $n$  は素数か?  
PRIME  $\equiv \{ [n] : n \text{ は素数} \}$

スターリングの公式:  
 $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

```

prog Naive(input n);
begin
  for each i := 1 < i < n do
    if n mod i = 0 then reject end-if
  end-for;
  accept
end.
    
```

2 ~ n-1 の数で割ってみる

$\log n \cdot \log i$  時間

$$time\_Naive(n) \leq \sum_{1 < i < n} (c \log n \log i + d) = c \log n \log n! + dn = O(n(\log n)^2)$$

$n$  の長さを  $l$  とすると、 $l$  はほぼ  $\log n$  だから、 $time\_Naive = O(l^2)$  故に、素数判定問題の時間計算量は (高々)  $O(l^2)$

余談:  
2002年に  $O(l^6)$  のアルゴリズムが考案された!!

10/18

**Def.4.5.**  
 For a function  $t$  over natural numbers, the set of all sets (i.e. recognition problems) with time complexities  $O(t)$  is called  **$O(t)$ -time complexity class**, and it is denoted by  $\text{TIME}(t)$ . And such a function  $t$  is called a **time limit**.  
 For example, a class of sets recognizable in time  $O(l^2)$  is  $\text{TIME}(l^2)$ , and the set PRIME is one element.  
 $\text{PRIME} \in \text{TIME}(l^2)$

10/18

**定義4.5.**  
 自然数上の関数  $t$  に対し、時間計算量が  $O(t)$  となる集合 (i.e., 認識問題) の全体を  **$O(t)$  時間計算量クラス** といい、そのクラスを  $\text{TIME}(t)$  と表す。  
 また、 $t$  のような関数を制限時間と呼ぶ。  
 たとえば、 $O(l^2)$  時間で認識可能な集合を集めたクラスが  $\text{TIME}(l^2)$  であり、集合 PRIME はその一要素。  
 $\text{PRIME} \in \text{TIME}(l^2)$

11/18

**Chapter 5**  
**Representative Complexity Classes**

**5.1. Representative time complexity classes**

$$\mathcal{P} \equiv \bigcup_{p:\text{polynomial}} \text{TIME}(p(l))$$

$$\mathcal{E} \equiv \bigcup_{c>1} \text{TIME}(2^{cl})$$

$$\mathcal{EXP} \equiv \bigcup_{p:\text{polynomial}} \text{TIME}(2^{p(l)})$$

$\mathcal{C}$  set: set in the complexity class  $\mathcal{C}$ .  
 $\mathcal{C}$  problem: problem of recognizing a  $\mathcal{C}$  set.

Problems not in  $\mathcal{P}$  are intractable from the practical viewpoint...

11/18

**第5章 代表的な計算量クラス**

**5.1. 代表的な時間計算量クラス**

$$\mathcal{P} \equiv \bigcup_{p:\text{多項式}} \text{TIME}(p(l))$$

$$\mathcal{E} \equiv \bigcup_{c>1} \text{TIME}(2^{cl})$$

$$\mathcal{EXP} \equiv \bigcup_{p:\text{多項式}} \text{TIME}(2^{p(l)})$$

$\mathcal{C}$ 集合: 計算量クラス $\mathcal{C}$ に入る集合。  
 $\mathcal{C}$ 問題:  $\mathcal{C}$ 集合の認識問題

ある問題が $\mathcal{P}$ に入っていないなら、現実的には手に負えない...

12/18

**Ex.5.1:** Polynomial makes no serious difference in the classes  $\mathcal{P}$ ,  $\mathcal{E}$ ,  $\mathcal{EXP}$ .  
 $\mathcal{P}$ : polynomial  $\times$  polynomial  $\rightarrow$  polynomial  
 $\mathcal{E}$ : linear power of 2  $\times$  polynomial  $\rightarrow$  linear power of 2  
 $\mathcal{EXP}$ : poly. power of 2  $\times$  poly.  $\rightarrow$  poly. power of 2

**Ex.5.2:** Complexity class of PRIME  
 Ex.4.7  $\rightarrow \text{PRIME} \in \text{TIME}(2^l)$   
 Thus,  $\text{PRIME} \in \mathcal{E}$

$O(l^6)$  time algorithm puts it into  $\mathcal{P}$ !!

**Def.5.1:**  $\mathcal{T}$ : set of time limits

$$\bigcup_{t \in \mathcal{T}} \text{TIME}(t): \mathcal{T} \text{ time complexity class}$$

$\rightarrow$  It is denoted by  $\text{TIME}(\mathcal{T})$ .

**Theorem5.1** (1)  $\mathcal{P} = \bigcup_{c>0} \text{TIME}(l^c)$ , (2)  $\mathcal{EXP} = \bigcup_{c>0} \text{TIME}(2^{l^c})$

12/18

**例5.1:** クラス  $\mathcal{P}$ ,  $\mathcal{E}$ ,  $\mathcal{EXP}$  では、多項式時間程度の違いは問題ではない。  
 $\mathcal{P}$ : 多項式  $\times$  多項式  $\rightarrow$  多項式  
 $\mathcal{E}$ : 2の線形乗  $\times$  多項式  $\rightarrow$  2の線形乗  
 $\mathcal{EXP}$ : 2の多項式乗  $\times$  多項式  $\rightarrow$  2の多項式乗

**例5.2:** PRIMEの計算量クラス  
 Ex.4.7  $\rightarrow \text{PRIME} \in \text{TIME}(2^l)$   
 故に、 $\text{PRIME} \in \mathcal{E}$

余談: 2002年に  $O(l^6)$  のアルゴリズムが考案されたので、今では  $\mathcal{P}$

**定義5.1:**  $\mathcal{T}$ : 制限時間の集合

$$\bigcup_{t \in \mathcal{T}} \text{TIME}(t): \mathcal{T} \text{ 時間計算量クラス}$$

$\rightarrow$  これを  $\text{TIME}(\mathcal{T})$  と表す.

**定理5.1:** (1)  $\mathcal{P} = \bigcup_{c>0} \text{TIME}(l^c)$ , (2)  $\mathcal{EXP} = \bigcup_{c>0} \text{TIME}(2^{l^c})$

13/18

**Theorem 5.1:** (1)  $\mathcal{P} = \bigcup_{c>0} \text{TIME}(l^c)$ , (2)  $\mathcal{E}\mathcal{X}\mathcal{P} = \bigcup_{c>0} \text{TIME}(2l^c)^c$

**Proof:** The proof of (2) is omitted.  
 $T_1$ : set of polynomials of the form of  $l^c$ .  
 $T_2$ : set of all polynomials  
 $\rightarrow$  since  $T_1 \subseteq T_2$ ,  $\text{TIME}(T_1) \subseteq \text{TIME}(T_2)$   
 $p$ : arbitrary polynomial ( $p$  is any element of  $T_2$ )  
 if the maximum degree of a polynomial  $p$  is  $k$ ,  $p(l) = O(l^k)$   
 From Theorem 4.3,  
 $\text{TIME}(p(l)) \subseteq \text{TIME}(l^k) \subseteq \text{TIME}(T_1)$   
 Therefore,  $\text{TIME}(T_1) = \text{TIME}(T_2)$  Q.E.D.

**Theorem 4.3:**  
 For any times  $t_1, t_2$ ,  
 $t_1 = O(t_2)$  implies  $\text{TIME}(t_1) \subseteq \text{TIME}(t_2)$

13/18

**定理 5.1:** (1)  $\mathcal{P} = \bigcup_{c>0} \text{TIME}(l^c)$ , (2)  $\mathcal{E}\mathcal{X}\mathcal{P} = \bigcup_{c>0} \text{TIME}(2l^c)^c$

**証明:** (2)の証明は省略。  
 $T_1$ :  $l^c$ という形の多項式の集合。  
 $T_2$ : 多項式の全体  
 $\rightarrow T_1 \subseteq T_2$ なので、 $\text{TIME}(T_1) \subseteq \text{TIME}(T_2)$   
 $p$ : 任意の多項式 ( $p$ は $T_2$ の任意の要素)  
 多項式 $p$ の最大次数を $k$ とすると、 $p(l) = O(l^k)$   
 定理4.3より、  
 $\text{TIME}(p(l)) \subseteq \text{TIME}(l^k) \subseteq \text{TIME}(T_1)$   
 したがって、 $\text{TIME}(T_1) = \text{TIME}(T_2)$  証明終

**定理 4.3:**  
 すべての制限時間  $t_1, t_2$  に対し、  
 $t_1 = O(t_2)$  ならば  $\text{TIME}(t_1) \subseteq \text{TIME}(t_2)$

14/18

**Ex.5.3. Problem of evaluating propositional expression (PROP-EVAL)**  
**Input:**  $\langle F, \langle a_1, a_2, \dots, a_n \rangle \rangle$   
 $F$  is an extended prop. expression  
 $(a_1, a_2, \dots, a_n)$  is a truth assignment to  $F$   
**Question:**  $F(a_1, a_2, \dots, a_n) = 1$ ?

$(x,y)$	$x \rightarrow y$ $(\neg x \vee y)$	$x \leftrightarrow y$ $((x \rightarrow y) \wedge (y \rightarrow x))$
(0,0)	1	1
(0,1)	1	0
(1,0)	0	0
(1,1)	1	1

14/18

**例5.3. 命題論理式評価問題 (PROP-EVAL)**  
**入力:**  $\langle F, \langle a_1, a_2, \dots, a_n \rangle \rangle$   
 $F$ は拡張命題論理式  $\wedge \vee \neg \rightarrow \leftrightarrow$   
 $(a_1, a_2, \dots, a_n)$ は $F$ に対する真理値割り当て  
**質問:**  $F(a_1, a_2, \dots, a_n) = 1$ ?

$(x,y)$	$x \rightarrow y$ $(\neg x \vee y)$	$x \leftrightarrow y$ $((x \rightarrow y) \wedge (y \rightarrow x))$
(0,0)	1	1
(0,1)	1	0
(1,0)	0	0
(1,1)	1	1

15/18

**Ex.5.3. Problem of evaluating propositional expression (PROP-EVAL)**  
**Input:**  $\langle F, \langle a_1, a_2, \dots, a_n \rangle \rangle$   
 $F$  is an extended prop. expression  
 $(a_1, a_2, \dots, a_n)$  is a truth assignment to  $F$   
**Question:**  $F(a_1, a_2, \dots, a_n) = 1$ ?

Construct a computation tree from a code  $[F]$  of ext. prop. expression  
 It is built in time  $O(|[F]|^3)$ .  
 If computation tree is available, we can easily obtain the value  
 $F(a_1, a_2, \dots, a_n)$  in a **bottom-up fashion**.

Ex.:  $F(x_1, x_2, x_3) = [x_1 \wedge \neg x_2] \vee [x_1 \rightarrow x_3]$

$F(0,1,0) = 1$   
 $F(1,1,0) = 0$

Hence PROP-EVAL  $\in \mathcal{P}$

15/18

**例5.3. 命題論理式評価問題 (PROP-EVAL)**  
**入力:**  $\langle F, \langle a_1, a_2, \dots, a_n \rangle \rangle$   
 $F$ は拡張命題論理式  $\wedge \vee \neg \rightarrow \leftrightarrow$   
 $(a_1, a_2, \dots, a_n)$ は $F$ に対する真理値割り当て  
**質問:**  $F(a_1, a_2, \dots, a_n) = 1$ ?

拡張命題論理式  $F$  がコード化されたもの  $[F]$  から計算木を作る。  
 計算木は  $O(|[F]|^3)$  時間で構成できる。  
 計算木が得られていれば、**ボトムアップ式**で  
 $F(a_1, a_2, \dots, a_n)$  の値は容易に計算可能。

例:  $F(x_1, x_2, x_3) = [x_1 \wedge \neg x_2] \vee [x_1 \rightarrow x_3]$

$F(0,1,0) = 1$   
 $F(1,1,0) = 0$

よって PROP-EVAL  $\in \mathcal{P}$

Ex. 5.3. 2-Satisfiability (2SAT) 16/18

**Input:**  $\langle F \rangle$   $F$  is 2-conjunctive normal form

**Question:** Is there any assignment such that  $F(a_1, a_2, \dots, a_n) = 1$ ?

Conjunctive Normal Form (CNF)  
 $F = (\bigcirc \vee \bigcirc \vee \dots \vee \bigcirc) \wedge (\bigcirc \vee \dots \vee \bigcirc) \wedge \dots \wedge (\dots)$   
 - described by  $\wedge$  of  $\vee$  of literals.

$k$  SAT exactly/at most  
 - Each closure contains  $k$  literals

- We can define 3SAT, 4SAT similarly.
- SAT consists of any CNF.
- ExSAT consists of any extended propositional expression.

例5.3. 命題論理式充足性問題:2和積形(2SAT) 16/18

**入力:**  $\langle F \rangle$   $F$ は2和積形命題論理式

**質問:**  $F(a_1, a_2, \dots, a_n) = 1$ を満たす割り当てがあるか?

和積形:  
 $F = (\bigcirc \vee \bigcirc \vee \dots \vee \bigcirc) \wedge (\bigcirc \vee \dots \vee \bigcirc) \wedge \dots \wedge (\dots)$   
 - リテラルの論理和の論理積で表現されたもの

$k$ 和積形( $k$  SAT) ちょうど/たかだか  
 - 和積形の各論理和が  $k$  個のリテラルを含む

- 3SAT, 4SAT も同様に定義できる。
- SAT: 各論理和のリテラルの個数に制限がないもの
- ExSAT: 入力が拡張命題論理式( $\rightarrow$ や  $\leftrightarrow$ も許す)

Ex. 5.4: Graph reachability problem (ST-CON) 17/18

**Input:**  $\langle G, s, t \rangle$  : an undirectd graph  $G$ ,  $1 \leq s, t \leq n(=|G|)$

**Question:** Does  $G$  have a path from  $s$  to  $t$ ?

- **Cycle** is a path that shares two endpoints.
- **Euler cycle** is a cycle that visits all **edges** once.
- **Hamiltonian cycle** is a cycle that visits all **vertices** once.

Ex. 5.4: Euler cycle problem (DEULER)  
**Input:**  $\langle G \rangle$ : a directed graph  $G$   
**Question:** Does  $G$  have an Euler cycle?

Ex. 5.5 Hamiltonian cycle problem (DHAM)  
**Input:**  $\langle G \rangle$ : a directed graph  $G$   
**Question:** Does  $G$  have a Hamiltonian cycle?

例5.4: 到達可能性問題(ST-CON) 17/18

**入力:**  $\langle G, s, t \rangle$  : 無向グラフ  $G$ ,  $1 \leq s, t \leq n(=|G|)$

**質問:**  $G$ 上で  $s$  から  $t$  への道があるか?

- **閉路**とは、始点と終点と同じである路
- **オイラー閉路**とは、すべての**辺**を一度ずつ通る閉路
- **ハミルトン閉路**とは、すべての**頂点**を一度ずつ通る閉路

例5.4: 一筆書き閉路問題(DEULER)  
**入力:**  $\langle G \rangle$ : 有向グラフ  $G$   
**質問:**  $G$ はオイラー閉路をもつか?

例5.5: ハミルトン閉路問題(DHAM)  
**入力:**  $\langle G \rangle$ : 有向グラフ  $G$   
**質問:**  $G$ はハミルトン閉路をもつか?

It is known that: 18/18

- The following problems are in  $\mathcal{P}$ :
  - ✓ PROP-EVAL, 2SAT, ST-CON, DEULER
- The following problems are in  $\mathcal{E}$ , but...
  - ✓ 3SAT, DHAM

The class  $\mathcal{NP}$  between  $\mathcal{P}$  and  $\mathcal{E}$ ?

以下の事実が知られている: 18/18

- 以下の問題は  $\mathcal{P}$  に属する:
  - ✓ PROP-EVAL, 2SAT, ST-CON, DEULER
- 以下の問題は  $\mathcal{E}$  に属する、が、、、
  - ✓ 3SAT, DHAM

$\mathcal{P}$  と  $\mathcal{E}$  の間(?)のクラス  $\mathcal{NP}$