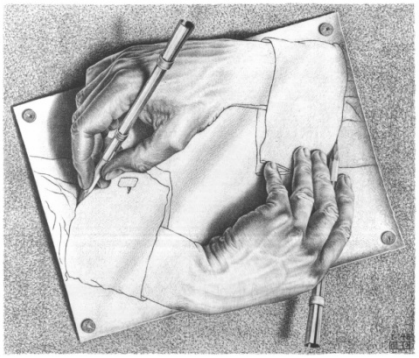
第6章　問題の解き方

本章で説明すること

- アルゴリズム
- 計算量
- Turing machine, 帰納的関数
- 計算可能性



6.1.2 アルゴリズムの実例
平方根の計算　――(2分法まで)

1. アルゴリズム１(反復による平方根の計算)
2. アルゴリズム２(２分法による平方根の計算)
3. アルゴリズム３(ニュートン法による計算)

問題: 平方根の計算

- $\sqrt{x}$ を求める
- 注意:
  - 小数の計算は有限の精度で行われる
  - →近似値しか求められない
- 問題:
  - ある正の実数 x が与えられたときに、2乗すると x に近くなる正の実数 y を精度 δ で求める.
  - つまり, $|\sqrt{x}-y|<\delta$ となるような y を 1 つ求める

平方根のアルゴリズム: 反復法

- x = 90, δ = 1 の場合を考える
- 「y = 0, 1, 2, 3, ... を順に検討してゆき $(y+\delta)^2$ が 90 より大きくなったら、その1つ前が解」

アルゴリズム1
(反復による
平方根の計算)

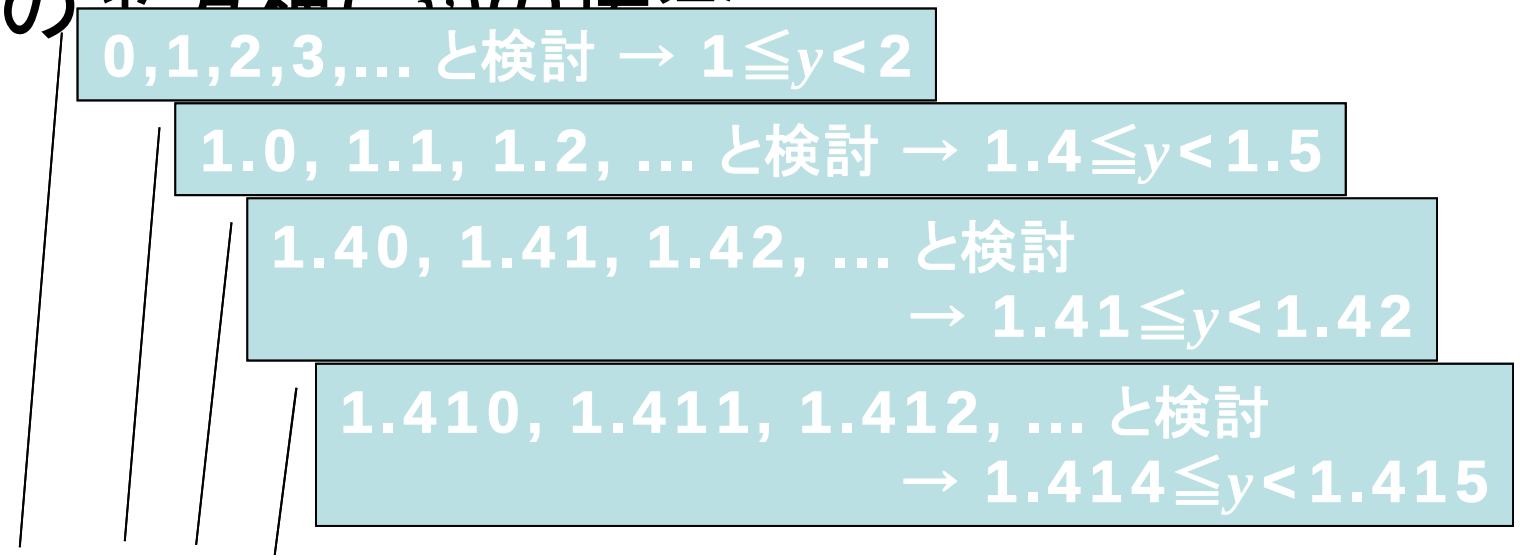
$y \leftarrow 0$
while $(y+\delta)^2 < x$ **do**
    $y \leftarrow y + \delta$
done
return y

- 実際の動作 x =2, δ =0.0001 の場合、

回数	0	1	2	···	14140	14141	14142
候補 (y)	0.0000	0.0001	0.0002	···	1.4140	1.4141	1.4142
$(y+\delta)^2$	0.00000	0.00000	0.00000	···	1.99968	1.99996	2.00024

平方根のアルゴリズム: 二分法の考え方

- アイデア: 1桁ずつ求めてゆく
- 例: 2の平方根(=y)の場合



1. 4 1 4 2 1 3 5 6 …

- 特徴: 解がある範囲を1/10ずつ狭めてゆく
- 単純化: →二分法 (次スライド)

平方根のアルゴリズム: 二分法

- アルゴリズム2 (二分法による平方根の計算)
  x の平方根を精度 δ で求める (ただし $x > 1$):

$a \leftarrow 0$

$b \leftarrow x$

while $b - > \ < \delta$ **do**

$c \leftarrow \frac{a+b}{2}$

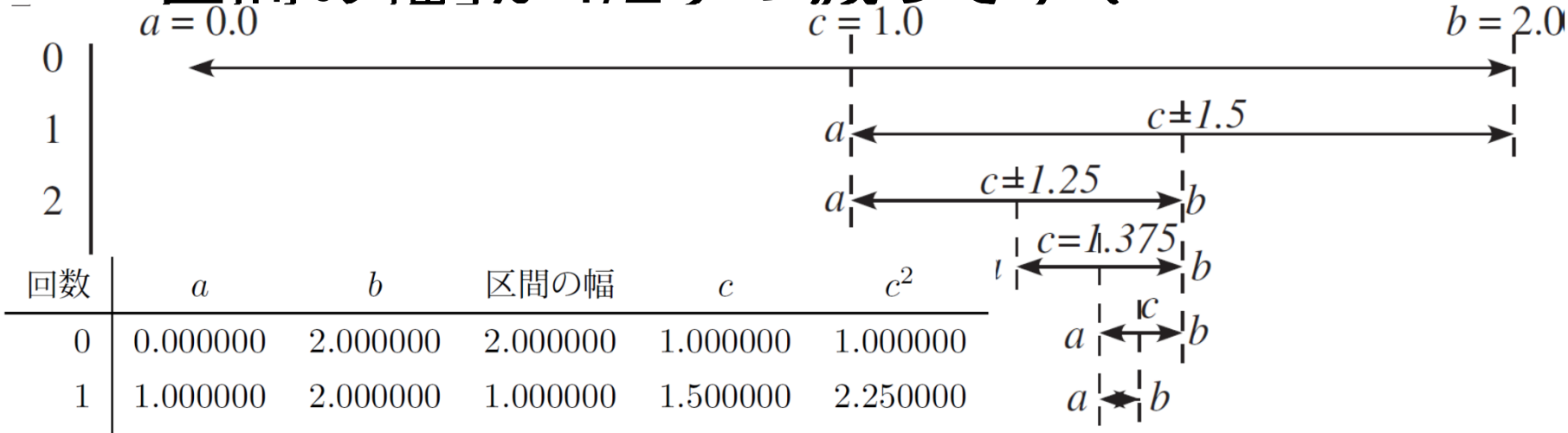
if $c^2 > x$ **then** $b \leftarrow c$ **else** $a \leftarrow c$ **endif**

done

return a

平方根のアルゴリズム: 二分法の実際

- 「区間の幅」が1/2ずつ減ってゆく



回数	a	b	区間の幅	c	c^2
0	0.000000	2.000000	2.000000	1.000000	1.000000
1	1.000000	2.000000	1.000000	1.500000	2.250000
2	1.000000	1.500000	0.500000	1.250000	1.562500
3	1.250000	1.500000	0.250000	1.375000	1.890625
4	1.375000	1.500000	0.125000	1.437500	2.066406
			(中略)		
13	1.414062	1.414307	0.000244	1.414185	1.999918
14	1.414185	1.414307	0.000122	1.414246	2.000091
15	1.414185	1.414246	0.000061		

アルゴリズムの速度

- 反復法: 約 $\sqrt{x}/\delta$ 回
- 二分法:
  - 1回繰り返すごとに区間の幅が1/2になる
  - ***n***回繰り返し後の区間の幅は $x/2^n$
  - これが δ 以下になるのに要する回数
    $\rightarrow$ 約 $\log_2 \frac{x}{\delta}$ 回
- 比較: x=2, δ=0.0000000001 のとき
  - 反復法: 約141億回
  - 二分法: 35回

小数点以下
10桁まで求める

計算量の例: 平方根の計算

- 問題: 精度 δ で x の平方根を求める
- 問題の大きさ: x と δ
- 計算量:
  - 反復法アルゴリズム: $O(\frac{\sqrt{x}}{\delta})$
  - 二分法アルゴリズム $O(\log(\frac{x}{\delta}))$

アルゴリズム３（ニュートン・ラフソン法）

初期値をx_0として$f(x)=0$を解く。

接線の方程式は,$y-f(x_0)=f'(x_0)(x-x_0)$で

$y=0$となるときの　xをx_1とすると

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

ここで$f(x)=x^2-n=0$とすると$x_1=\frac{x^2+n}{2x}$となる

$x_0 \to x_1 \to x_2 \to \cdots$非常に早く$\sqrt{n}$に収束する

6.1 アルゴリズム(p.131最短路問題)

点集合$V=\{v_1,\ldots,v_n\}$, v_1が始点

v_1から各点v_iへの最短距離をr_i

とすると動的計画法の最適律から

$r_1=0,$

$r_j=\min\{r_k+l(v_k,v_j)\,|\,k\neq j\}(j\neq 1)$

一般にこのままでは解けない

ネットワークが有向閉路を持たないとき

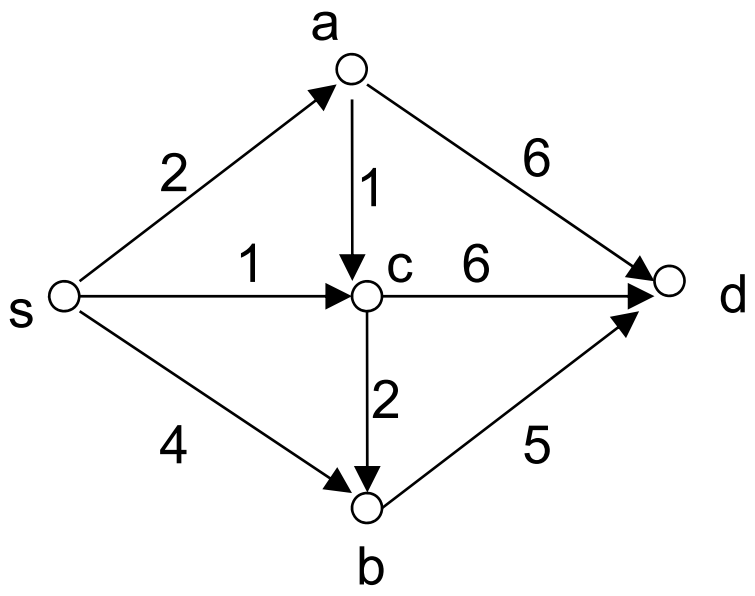
- そのときは、前の式が逐次代入ですぐ解ける。
- ネットワークが有向閉路をもたないとき、点の添え字を適当につけ替えれば、有向辺 (v_k, v_q) が存在すれば必ず $k < q$ となるようにできる

このとき

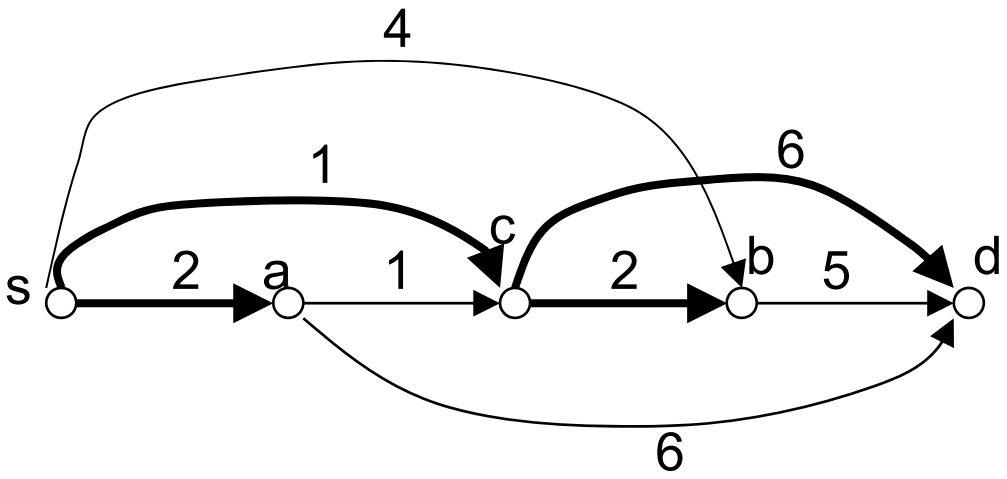
$$r_j = \min\{r_k + l(v_k, v_j) \mid 1 \le k < j\}$$

$$for \quad j = 2, \ldots, n.$$

で、順に求まる。



Acyclic network の例



1. 点を整合的な全順序に並べる

2. 左から順に最短路が求まる

6.1.4 計算量

- 計算量とは、計算時間のこと。ただし、定数倍を無視してオーダーで考える。
- たとえば、入力の問題の大きさが、n であったとして

$$O(n\log n), O(n^2), O(n^3), O(2^n)$$

などと表現する。(p.136表6.1参照)

定数倍を無視する。各変数について一番変化の大きい項だけを残す。定数倍の差は、問題のサイズが大きくなって計算時間が増えていくとき無視できる。

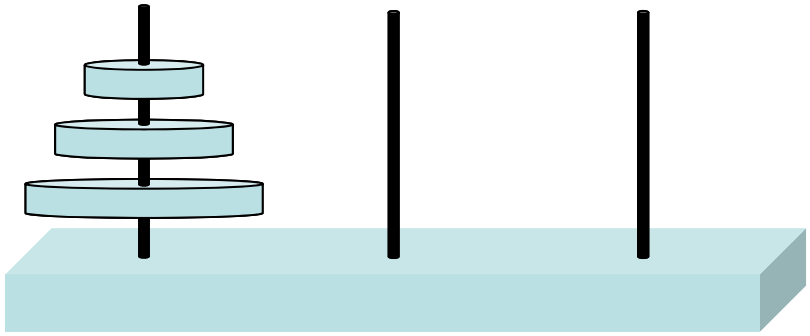
ハノイの塔問題

64 枚の円盤を1枚ずつ別の棒に通して置くことをくり返して円盤の山を左から右端に移動する。このとき常に各棒で下にある円盤は上のものより大きくなければならない。1枚移動させるのに1秒かかったとして、移し終えるのにどのぐらいの時間がかかるか。移し終えると世界の終わりが来る。

円盤の枚数が n のときにかかる時間は　$2^n - 1$　秒である。

n= 64 のとき、これは約5,845億年になる。

(なお、ビッグバンは今から約137億年前の発生とされている)



証明：f(n) を n 枚の円盤の場合の秒数とする。

$$f(1)=1,$$

$$f(n)=f(n-1)+1+f(n-1)=2f(n-1)+1,$$

を満たすから、ゆえに

$$f(n)+1=2(f(n-1)+1)$$

$$f(n)+1=2^{n-1}(f(1)+1)=2^{n}$$

$$f(n)=2^{n}-1$$

計算量の違いによる実行時間の差

1回の計算に 1 マイクロ秒 (1.0×10^{-6}) かかるときの計算時間

サイズ n	10	20	40	100
$10^6 n^2$	100秒	400秒	26.7分	2.78時間
n^6	1秒	1.06分	1.14時間	277.8時間
e^n	0.02秒	485秒	7464年	8.52×10^{21} 億年
2^n	0.01秒	1.05秒	305時間	4.02×10^{8} 億年
$n!$	3.63秒	4060年	2.59×10^{26}億年	2.99×10^{134} 億年

たとえ頭の定数がどんなに大きくても次数が高くても多項式時間の増加より、指数時間的増加のほうがすぐに圧倒的に大きくなる。
(計算機の能力向上でもカバーしきれない。)

ハミルトニアン閉路問題

グラフG=(V,E) (Vは頂点集合、Eは辺集合　|V|=n) に対して

「ひとつの点から出発して他の点をちょうど一度ずつ通って元に戻る閉路が存在するか？」

場合の全てを調べる方法：場合の数が (n-1)! で、小さな n で計算がすぐに破綻する。

ナップザック問題

品物の集合 S={1,2,…,n}，各品物 i の重さ g_i , 品物の価値 v_i

ナップザックに詰められる重量の限界 K

「品物を選んで総重量が K 以下で、価値の和が最大になるようにするには、どの品物を選べばよいか？」

場合の全てを調べる方法：S の部分集合全部を調べることにすると、部分集合は 2^n 個ある。→これも計算時間がすぐ破綻する。

ナップザック問題：簡単そうで実際的には解けない問題

- N 個の品物がある。その集合を S とする。
- 各々 a_1,…, a_n 円の価値がある。
- 各々重さが w_1,…, w_n kg あるとする。
- S から選んだ品物の集合で主さの和が W 以下になるもののうちで、価値の総和が最大になるものを見つけたい。

$$\max \sum_{i \in A} a_i \ : \quad A \subseteq S, \ \sum_{i \in A} w_i \leq W$$

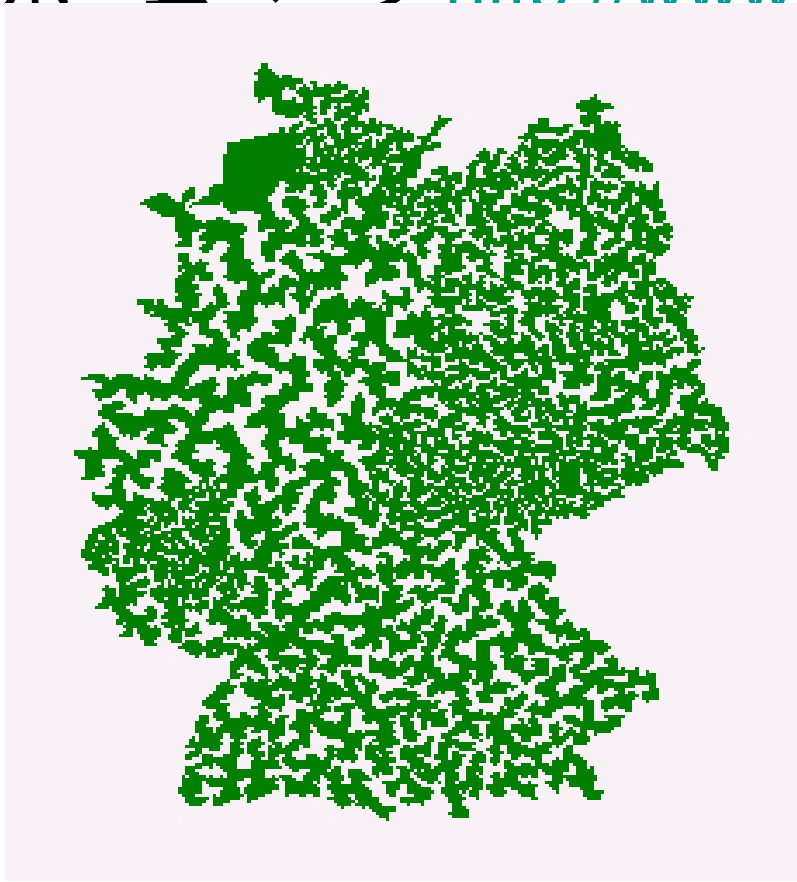
S の部分集合を全部列挙して、最適なものを見つければよい。しかし

S の部分集合の数は 2^n 個ある。n が大きくなると全列挙は計算時間が爆発して非実際的である。しかもこの問題は　NP-完全で、効率的なアルゴリズムは存在しないと考えられる。

巡回セールスマン問題 (Traveling Salesman Problem)

与えられた n 個の都市をどの都市も少なくとも1度訪れて始点に戻るパスの中の最短距離のものを探す

(TSP のホームページ http://www.tsp.gatech.edu/)



(ドイツ、15112地点)

巡回セールスマン問題の最適解

スウェーデン
24,978 都市

良いアルゴリズム (効率的なアルゴリズム)

「良い」は自然言語なのでそのままでは数学的な定義ではない。
以下で数学的な定義とする。

(A) 計算時間のオーダーが以下のように入力サイズ n の多項式でおさえられるとき、多項式時間のアルゴリズムと呼び、これをもって良いアルゴリズム (nice, efficient algorithm) とする。

$$O(n\log n), O(n^2), O(n^3), O(n^4), \ldots\ldots\ldots$$

(B)指数時間的アルゴリズム

$$O(e^n), O(2^n), O(n!), \ldots\ldots\ldots$$

n の値が少しでも大きくなると計算時間が爆発的に増えて、事実上計算できなくなる。実際上は「百年後に答が出る」は解けないのと同じことである。

ここで述べる問題という言葉の意味:

「与えられた自然数 p が素数であるか」という判定問題は、その中に「19 は素数であるか」「35 は素数であるか」という個々の問題を無限個含んでいる。このとき、「与えられた自然数 p が素数であるか」を問題と呼び、「19 は素数であるか」といった個々の問題を例問 (instance) と呼ぶ。つまり、ここで言う問題とは、例問の (可算無限個の) 集まりである。

クラスP: 良いアルゴリズムつまり多項式時間のアルゴリズムで解ける問題の全体のなすクラスを P と名付ける。

クラス P に属する問題の例

1. グラフ上の最短路問題
2. グラフ上の最大マッチング問題
3. グラフの平面性判定問題
4. 枝容量つきネットワークの最大流問題
5. 線形計画法（線形計画問題）
6. 有向グラフ上での r-branching の packing 問題
7. グラフの最小重み spanning tree 問題
8. 素数の判定問題

問題のクラスNP

問題の解の候補をひとつ挙げてそれで問題の答えがYes と分かる可能性のある問題全体のクラス。（厳密な定義で言えば、ある非決定性 (Non-deterministic) Turing 機械で多項式時間 (Polynomial-time) で解ける問題のこと。）

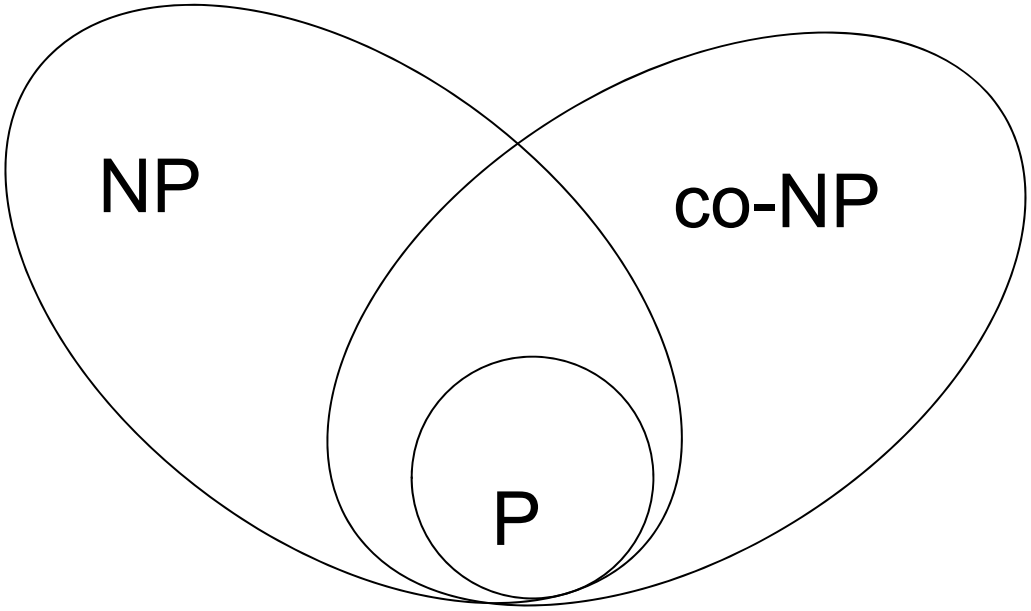
例　ハミルトニアン閉路問題では、実際にハミルトニアン閉路が存在するとき、それを解の候補としてチェックすれば、ただちにYes と答えられる。が、No 存在しないという答えを得るのには、証拠をひとつ挙げただけでは No とは言えない。

問題のクラス co-NP

問題の解の候補をひとつ挙げてそれで問題の答えがNo と分かる可能性のある問題全体のクラス

例 「与えられた自然数 p が素数であるか」は、p を割り切る数の候補をひとつ挙げてそれで実際に割り切れれば No と答えられる。逆に、証拠をひとつ挙げただけでは Yes と答えられない。

計算量のクラスの関係



(a) 実は、最近、素数判定問題がクラスPに属することが証明された.
(b) 教科書 6.3.1「計算量の階層」p.149
「…一筆書きのような「NPであるがPだとはわかっていない」問題である」という記述は誤り。一筆書き問題(Euler 閉路問題)はPに属する。ハミルトニアン閉路問題 は NP-完全。

NP-完全問題

NPに属する問題で、もしその問題に多項式時間の解法 (アルゴリズム)が存在すればNPに属する他の問題がすべて多項式時間のアルゴリズムで解ける、という性質を持つ問題を NP-完全問題と呼ぶ。

単純に見えるものから複雑なものまでこれまでに極めて多数の問題が NP-完全であることが示されている。もし、ひとつのNP-完全問題に多項式時間のアルゴリズムが存在することが分かれば、そこから NP に属する他の全ての問題に対する多項式時間のアルゴリズムが構成できて、クラス P と NP が一致することが分かる。が、逆に言うと、それら全ての問題が良いアルゴリズム、つまり多項式時間のアルゴリズムで解けるとはとても思われないので、問題がNP-完全と分かれば多項式時間のアルゴリズムは存在しないと考える、というのが現在のこの分野の常識である。つまり　　と考えるのが常識であるが、これは証明されていない。

P=NP かどうかの問題は、非常に大きな未解決問題であるが、現在の数学の論理の枠内では証明できないのではないかと唱える学者もいて、実際のところ証明ができる見通しがまったく立っていない。

NP-完全な問題の例

- グラフの最小彩色数（点の彩色）
- Traveling Salesman Problem ←→Chinese Postman Problem is in P
- ハミルトン閉路問題、ハミルトン閉路問題
- 最長路問題　←→ 最短路問題はクラスPに属する
- 集合のパッキング
- 集合の Covering
- 集合分割問題
- ナップザック問題
- Quadratic Diophantine Equations: Instance [正整数 a, b, c]

  Question　$\exists x, y : \text{integer} \quad ax^2 + by = c\,?$
- 充足可能性問題：Instance [論理命題　$P(x_1,\ldots,x_n)$]
- Generalized Hex(パズル): Instance [グラフ G とその2点 s,t]

←→辺で考えると Shannon switching game になってクラスPに属する。

- 有限オートマトンの非同値性

NP-完全, NP-困難 (多項式オーダーの時間のアルゴリズムの存在しない問題) なそれぞれの特定な問題に対して、多項式時間ではないものの最適解を与える有効なアルゴリズムが開発されてきています。

その良い例がハミルトン閉路問題の一般化にあたる**巡回セールス問題**で、Webページ http://www.tsp.gatech.edu/ に様々な例が集められています。24,978 cities in Sweden を廻る最適解が載っています。ということは、ハミルトン閉路問題もかなり大きなサイズまで実際に最適解が求まるということです。

・p.149 12行目から15行目

「先の一筆書きをするような…..計算量のオーダーが n^k となるようなアルゴリズムは見つかっていない。」

→これは間違いです。一筆書き（オイラー閉路問題）はきわめて簡単に解けます。これはハミルトン閉路問題と取り違えたのでしょう。たぶん。

ハミルトン閉路問題の一般化である巡回セールスマン問題では、かなり多きなサイズの問題の最適解が求めることができるようになってきました。

P.147 本文下から5行目

「たとえば 6.1.3 項の最短経路を探す問題のように、たくさんの経路の中から最もよい解を探すような問題は、現在のコンピュータではすべての組み合わせを1つ1つ順に調べている」とあるのは、**まったく誤りです**。

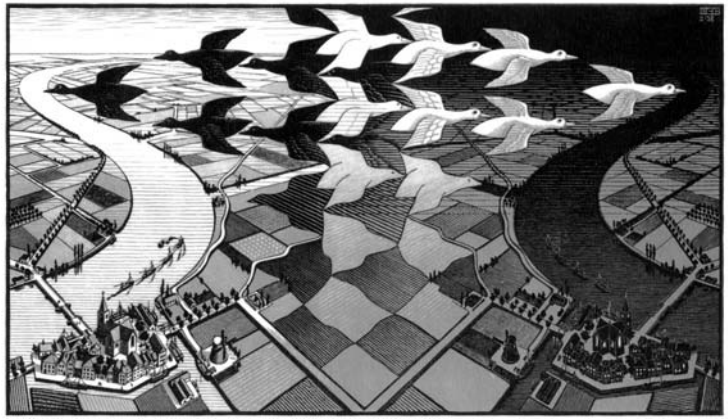
→最短路問題は、効率的アルゴリズム（多項式時間アルゴリズム）で解ける代表的な問題の例です。辺の長さが非負の場合には、Dijkstra 法が実用的には一番よく、計算のオーダーも O(n^2) ですみます。データ構造に工夫を加えるとオーダーが O(n+m) まで下がります。ここで、n は点の数、m は辺の数です。平面グラフならばO(n) ですみます。

第6章－2　計算のモデル

オートマトン

Turing 機械

計算可能性



6.2 計算のモデル色々

1. 有限状態機械の例

有限オートマトン、ペトリネット、順序機械（出力を持つ有限オートマトン）

→これらは計算の能力を持つとは言い難い。足し算はできるが自然数のかけ算ができない。

2. 無限状態をもつ計算の数学的モデルと関数のクラス

Turing 機械、Random Access 機械 (RAM Machine)

帰納的関数、ラムダ計算、Post システム

→ 2.に挙げた計算モデルやシステムで計算可能な関数は、関数のクラスとしてすべて同じクラスを定義することが知られている。このクラスに属する関数を計算可能と呼ぶ。

有限オートマトン

Σ：記号の非空有限集合

Σ^*: 記号の有限列の全体

1)Q：内部状態の集合

2)$s \in Q$：初期状態

3)$F \subseteq Q$：受理状態

4)γ： $Q \times \Sigma \to Q$, 遷移関数

→オートマトンシミュレーターの実演

入力記号列　　$w = x_1 x_2 x_3 \ldots x_k$　　に対して

初期状態sからwで遷移した最終状態を$\gamma(s, w)$とする。

$L = \{w \in \Sigma^* : \gamma(s, w) \in F\}$

最終状態がLに含まれる記号列を、Lで受理される受理される記号列(語)と呼び、オートマトンから受理される記号列の全体のなす集合がひとつ定まる。

有限オートマトンの応用

(1) デジタル回路の動作の設計とチェック

(2) コンパイラーの字句解析のソフトウェア

(3) web ページのような大量のテキストを走査して単語などの有無を探索するソフトウェア

(4) 通信プロトコルなど、いくつかの有限個の状態を取り得るあらゆる種類のシステムの検査をするソフトウェア

オートマトンと数理言語

言語	正則言語	文脈自由言語
システム	オートマトン	プッシュダウン付きオートマトン
文法、表現	正規表現	文脈自由文法

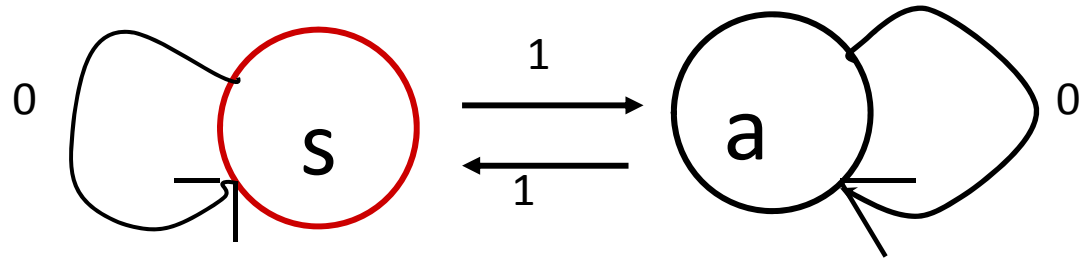
コンパイラー　プログラム言語の文法の大部分は自由文脈文法 (context-free grammar) のクラスに属する。このクラスは、プッシュダウン・オートマトンの受理言語のクラスに一致する。プログラミング言語に対応する構文解析プログラム (コンパイラー) は、あるプッシュダウン・オートマトンに同等である。

プッシュダウン・オートマトンとは、スタックというタイプのメモリを持ったオートマトンのことである。

有限オートマトンの例

記号集合 S={0,1}, 状態集合 Q={s,a}, 初期状態: s,
受理集合: F={a}

状態遷移図



1 受理

01 受理

10101 受理

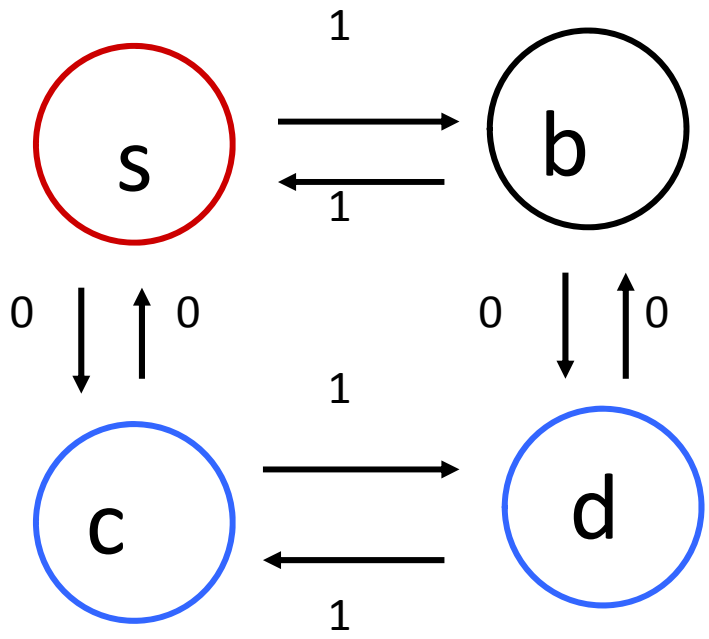
011 受理されない

受理される記号列の全体＝
1を奇数個含む列 の全体

38

例

S：初期状態　　記号集合: $\Sigma = \{0,1\}$

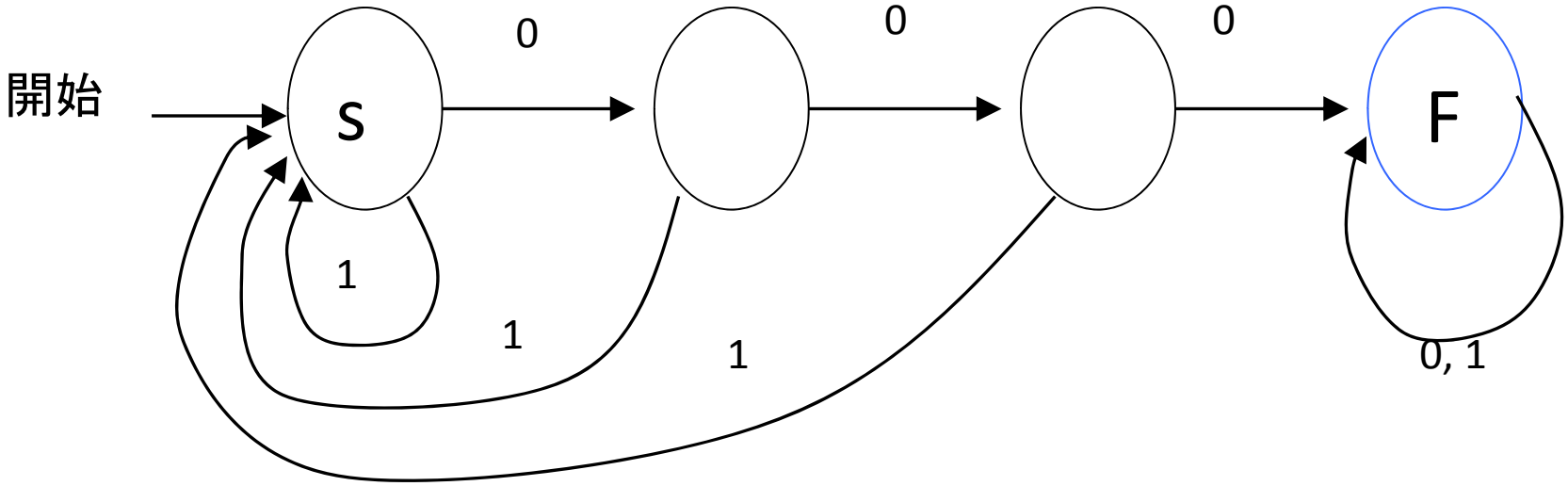


受理状態：　$F = \{c, d\}$ ⟶ 受理される語(word)は、0 が奇数個含まれるもの

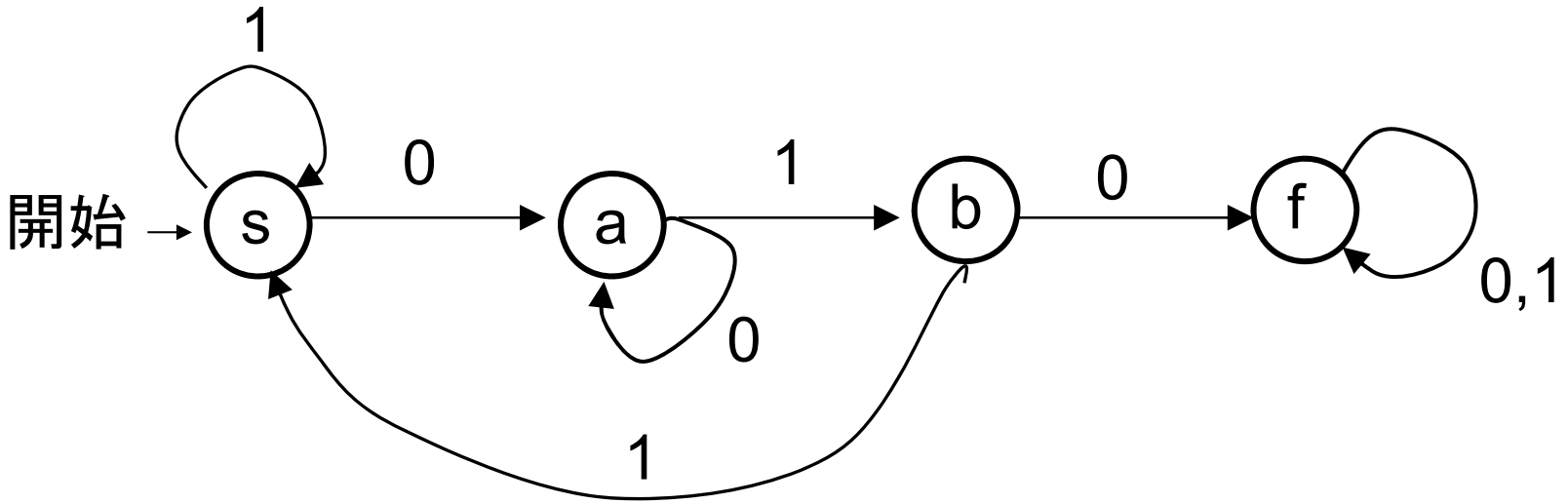
受理状態：　$F = \{d\}$ ⟶ 受理される語(word)は、0,1 をそれぞれ奇数個含むもの

39

例題： 記号集合={0,1}, 受理集合={F}

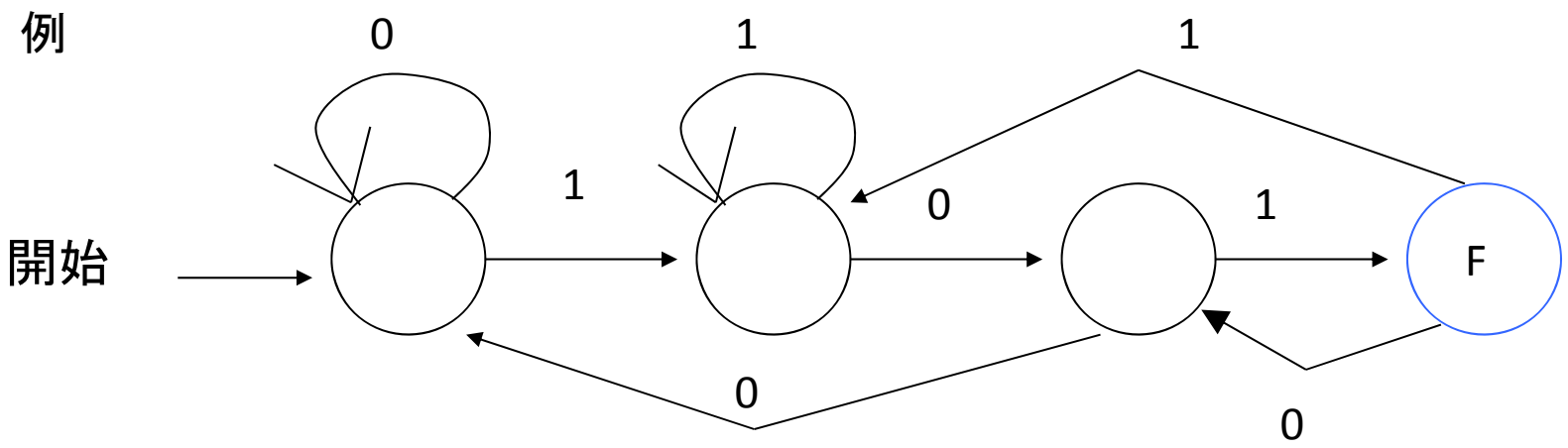


受理言語＝記号列内に列 000 を少なくともひとつ含むものの全体



入力記号：{0,1}
受理集合 F={ f }

受理記号列：010 を部分列として含む記号列

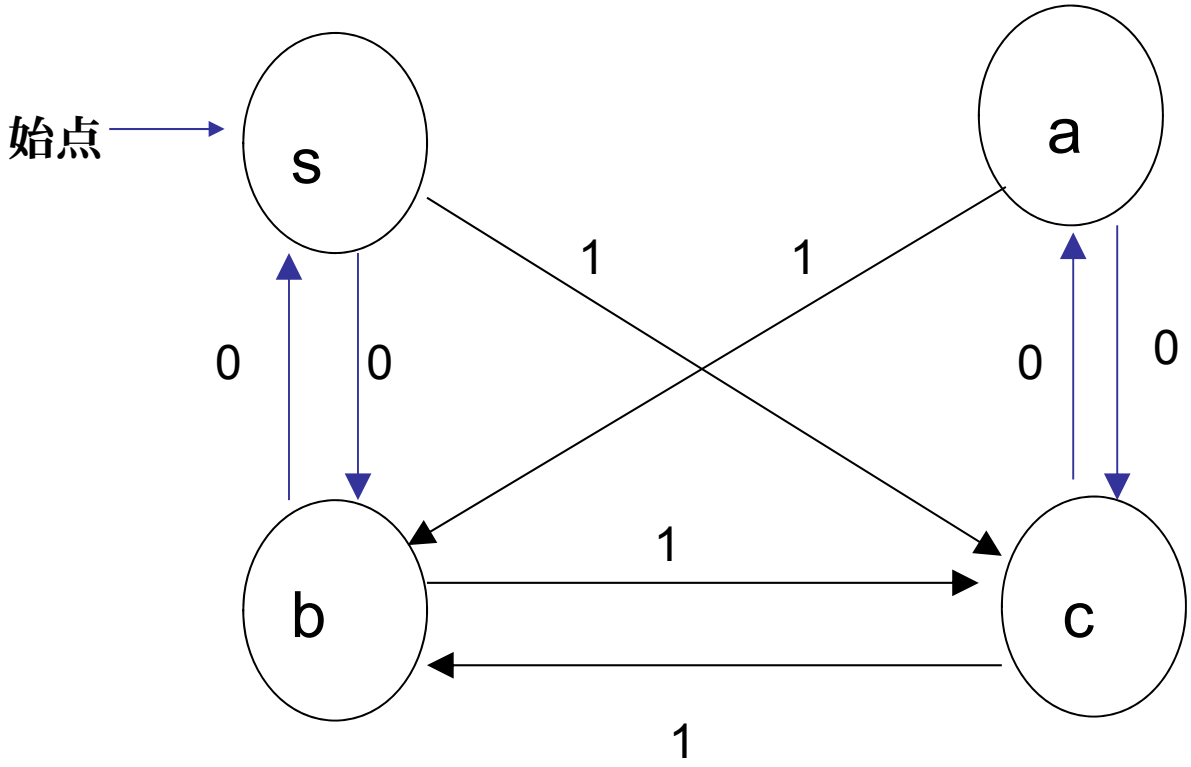


対応する言語 L = 末尾が 101 で終わる語の全体

オートマトン・シミュレーター（中村講義用ページから）
http://lecture.ecc.u-tokyo.ac.jp/johzu/joho/automaton/
にアクセスして、AutoSim.jar をダウンロードして、実行する。
ファイルをクリックすると展開して実行が始まる。

演習問題

受理集合 F={s,c} のとき、受理される言語 L (受理される語の全体) はどのようなものか?



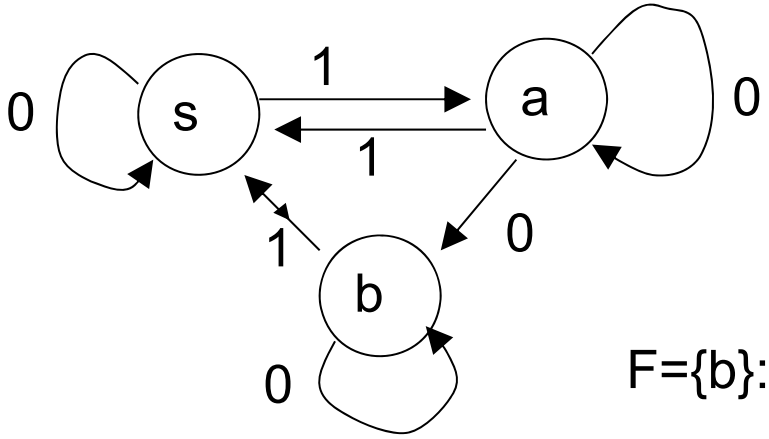
· 語(記号列)に従ってオートマトンが内部状態の遷移を行い、最後の状態が受理状態であれば、このオートマトンは、この語を受理するという。

決定性有限オートマトンによって受理される言語の全体の集合 = 正規表現で表せる言語 = 正則文法で決定される言語

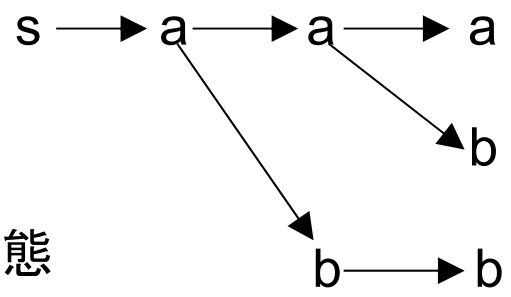
非決定性オートマトンで受理される言語の全体と決定性オートマトンで受理される全体は一致する。

非決定性オートマトン

（cf.非決定性Turing機械　p.149）



入力 100 に対する状態遷移



F={b}:受理状態

$\gamma(s,w)$ は、入力列 w に対して到達しうる内部状態全体の集合を表すとする。

このとき以下で受理列の全体のなす言語がひとつ定まる。

$$L = \{w \in \Sigma^* : \gamma(s,w) \cap F \neq \phi\}$$

上の場合 L は、1 を奇数個含み 0 で終わる列の全体になる。

演習問題　以下を受理するオートマトンを作れ。

1)00 で終わる列の全体

2)3 個連続した 0 を含む列の全体

3)1 で始まる列で、2 進数としてみたとき 、5 で割り切れるものの全体。

4)どの相続いた5個の記号の中にも2 個以上の 0 が含まれるもの。

オートマトンで識別できない言語の例

・ L を 1 で始まる 0,1 の列で 2 進数とみたとき素数になるものの全体の集合。

Exercise 1: lock

暗証番号「ababc」が入力されたときに解錠する(終了状態になる)電子錠．ただし，途中間違った暗証番号を押しても，後から「ababc」と入力すれば解錠するものとする．また「ababc」と入力された後，何か入力されたら施錠する(終了状態でなくなる)ものとする．入力は「a」「b」「c」の3文字だけと仮定してよい．

Exercise 2: three a before b

入力は「a」が何回か続き，最後に「b」が1回だけ現われるものとする．「b」が現われたときに，それまでに「a」の現われた回数が3の倍数だったときだけ終了状態になる．なお，3の倍数には0を含むものとする．つまりaが1回も現われていないときも終了状態になる

Exercise 3: three a before c

入力が先頭から「a」または「b」が何回か続き，最後に「c」が1回だけ現われるものとする．「c」が現われたとき，それまでに「a」がちょうど3回現われていた場合のみ，終了状態になるようなオートマトン．c以降のことは考えなくてよい．

Exercise 6: abac of baca

2つの暗証番号「abac」または「baca」のどちらが押されても解錠するような電子錠．ただし，途中間違った暗証番号を押しても，後から正しい暗証番号を入力すれば解錠するものとする．入力は「a」「b」「c」の3文字だけと仮定してよい．一度解錠されたら，どんな文字が来ても解錠されたままだとする．
注:「abaca」と入力すると4文字目と5文字目の両方で解錠する．

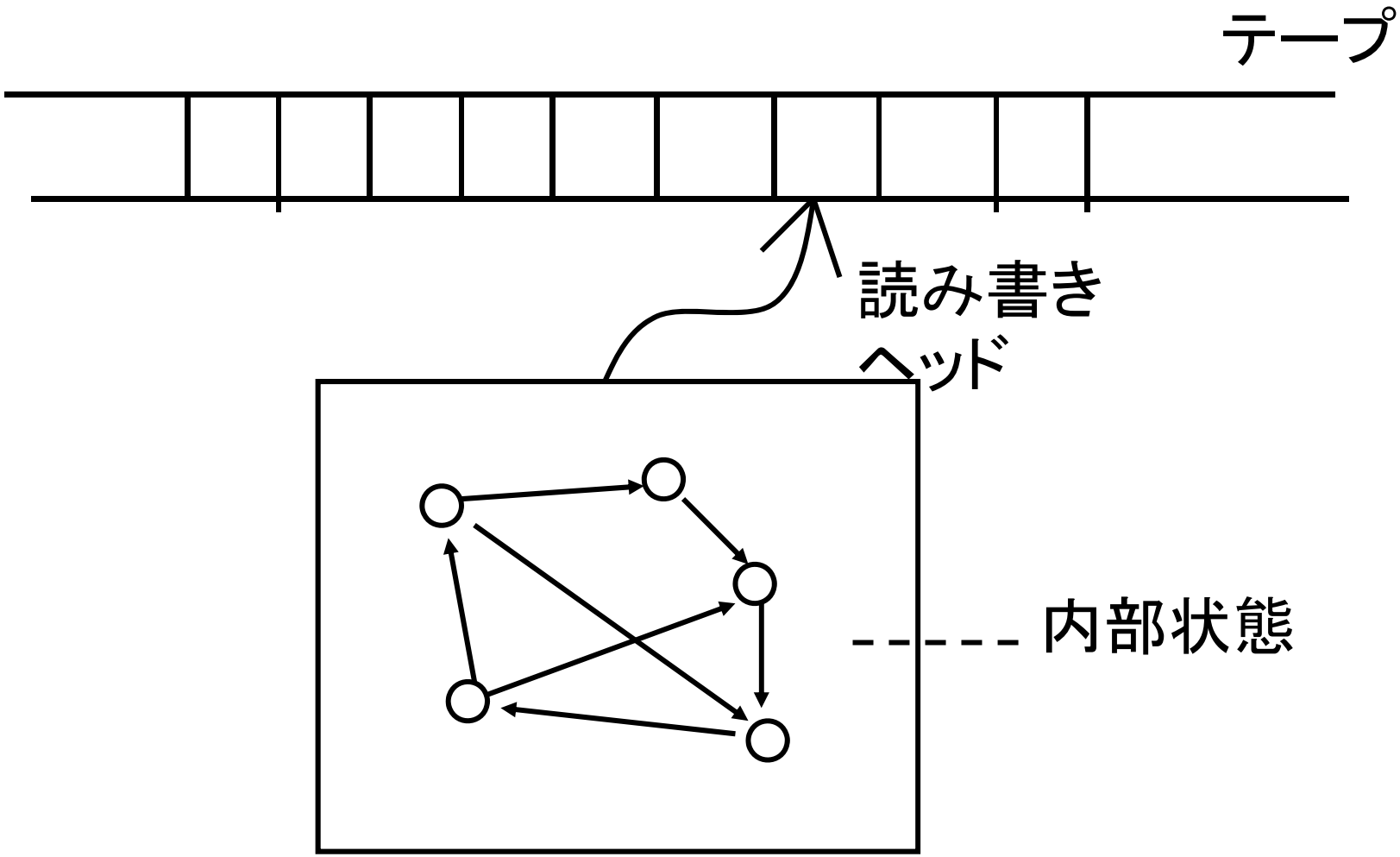
Exercise 8: electric lock

「ab」を文字キー，「c」をロックキー，非終了状態をロックされた状態だとしたときに，2文字のパスワードを覚えてロックし，パスワードを打つと解除される電子錠．最初は解除状態(=終了状態)だとする．
解除状態のときは，文字キーが2回以上押された後ロックキーが押されるものとする．
ロックキーが押された後はロック状態(=非終了状態)になる．
ロック状態のときは，文字キーしか押されないものとする．
ロック状態のときに，ロックキーを押す直前に押された2つの文字キーを同じ順序で押されると，解除状態になり，最初に戻る．
たとえば「abc」と押すとロックされ，続けて「ab」と押すとロックが解除される．
また「abbc」と押すとロックされ，続けて「aabb」と押すとロックが解除される．

Exercise 10: multiple of three

入力記号が 0,1 で、入力列を2進数としたとき、入力が3の倍数のときにだけ終了状態となるオートマトン．

6.2.1(b) Turing機械 (p.139)



Turing 機械

Σ:テープ上の記号の非空有限集合

1)Q:内部状態の集合

2)$q_0 \in Q$:初期状態

3)$F \subseteq Q$:停止状態

4)遷移関数

次の時点での内部状態

現在のマス目に記入する記号

$$\Gamma : Q \times \Sigma \to Q \times \Sigma \times \{\text{左}、\text{右}\}$$

現在見ているテープ
のマス目上の記号

現在の内部状態

次の時点で読み取りヘッドを
左または右に1マス動かす

（現在の内部状態、ヘッドの見ている記号）

↓

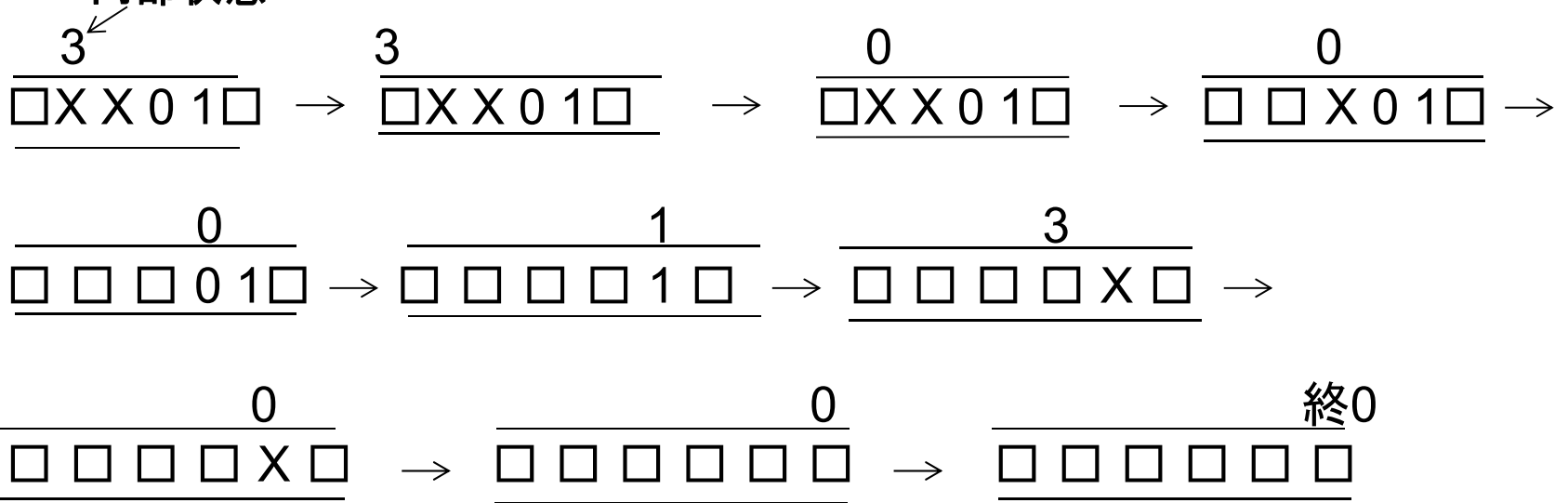
（次の内部状態へ移り、ヘッドのあるマス目に記号を書き込んで、ヘッドを右か左に1マス動かす）

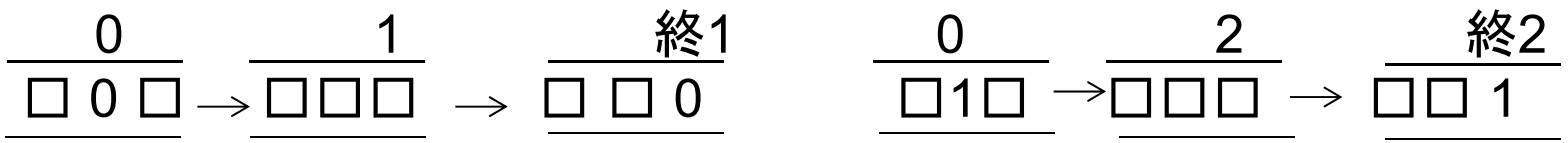
↓

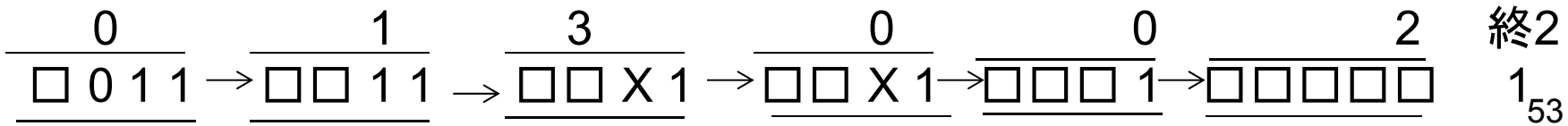
内部状態が停止状態になったら、停止する。一般に任意のTuring 機械に任意の入力を与えたとき、有限時間内に停止するか否かの問題は決定不能。

6.2 p.141　チューリング機械の動作の続き

内部状態







ランダムアクセス機械

- Turing 機械で、テープの代わりに番地のついたメモリが利用できるもの。
- たとえば「１０１番地の中身を読み込む」「２０５番地に書き入れる」というようなことができる
- チューリング機械のように1マスごとにヘッドを動かさずにすむ

部分帰納的関数、帰納的関数

関数の合成

$f(y_1,...,y_m)$と$g_1(x_1,...,x_n),...,g_m(x_1,...,x_n)$
から$f(g_1(x_1,...,x_n),...,g_m(x_1,...,x_n))$を作る

原始帰納法

与えられた関数　$f(x_1,...,x_n)$,　$g(x_1,...,x_{n+2})$　から以下
新しく関数 $h(z,x_1...,x_n)$ を定義できる。
これを原始帰納法と呼ぶ。

$$h(0,x_1,...,x_n) = f(x_1,...,x_n),$$

$$h(z+1,x_1,...,x_n) = g(z,h(z,x_1,...,x_n),x_1,...,x_n)$$

原始帰納的関数

以下の関数リストPを出発点にして、合成と原子帰納法を有限回くりかえして得られる関数を

原始帰納的関数であるという。

リストP

$$(1) S(x) = x + 1,$$

$$(2) N(x) = 0,$$

$$(3) U_i^n(x_1, \ldots, x_n) = x_i$$

$$(1 \leq i \leq n).$$

原始帰納法の適用

２数の足し算

plus(0,y)=y

plus(x+1,y)=S(plus(x,y))

２数のかけ算

multi(0,y)=0

multi(x+1,y)=plus(multi(x,y),y)

原始帰納法と原始帰納的関数の例

$(a) x + y$

$$x + 0 = U_1^1(x),$$

$$x + (y+1) = S(x+y).$$

$(b) x \cdot y$

$$x \cdot 0 = N(x),$$

$$x \cdot (y+1) = (x \cdot y) + U_1^2(x, y).$$

$(b) P(x)$

$$P(0) = N(x),$$

$$P(x+1) = U_1^1(x).$$

P(x)=0 if x=0,
P(x)=x-1 if x>0

$(c) x \dot{-} y$

$$x \dot{-} 0 = U_1^1(x),$$

$$x \dot{-} (y+1) = P(x \dot{-} y)$$

最小化

$$\min_y[f(y,x)=0]=$$

f(y,x)= を満たすy が存在すれば、そのような y のうち、最小の値を取る。f(y,x)=0 を満たす y が無ければ、定義されないとする。

常にy の値が定義されるとは限らない。常に定義されるとき、f は正則であるという。

y の値が定義されない←→Turing 機械が止まらない

帰納的関数（定義１）

リストP　の関数に合成と原始帰納法と、関数の最小化を有限回繰り返して得られる関数を、**部分帰納的関数**と呼ぶ。

全域で定義されている部分帰納的関数を

帰納的関数という。

例　帰納的だが原始帰納的でない関数：アッカーマン関数(Ackermann function)

$$A(0, y) = y + 1, \quad A(x+1, 0) = A(x, 1),$$

$$A(x+1, y+1) = A(x, A(x+1, y)).$$

Turing 機械と帰納的関数の対応

Turing 機械 $\longleftrightarrow$ 部分帰納的関数

任意の入力で有限ステップで停止する Turing 機械 $\longleftrightarrow$ 帰納的関数

帰納的関数（定義２）

- つぎの４種の関数から出発して

$$(1) S(x) = x + 1,$$

$$(2) U_i^n(x_1, \ldots, x_n) = x_i (1 \le i \le n)$$

$$(3) x + y$$

$$(4) x \dot{-} y$$

$$(5) xy$$

合成と最小化を有限回繰り返して得られる関数を部分帰納的関数と呼び、全域的に定義されるときは、帰納的関数という。

Hilbert の第10問題

- 整数係数の多項式 f(x1,x2,…,xn) に対して f(a1,a2,…,an)=0 を満たす整数a1,a2,…,an が存在するか否か判定するアルゴリズムは存在するか。[Hilbert の第10問題]
- Yu Matiyasevich (1970) が、そのようなアルゴリズムが存在しないことを証明した。

計算可能性とアルゴリズムの定義
(Church-Turing の提唱)

関数のクラスとして、以下のシステムは全て同じクラスを与える。

- 常に有限時間で停止するTuring機械（A. チューリング）
- 帰納的関数(Kleene、Church）
- λ-calculus
- Post のシステム

この同値なクラスに属する問題を「計算可能」と呼ぶ。

計算できる或いは解法（アルゴリズム）が存在するという人間の言葉を、上記の関数のクラスに属することとして定義しようという提唱（数学的言明ではなく、人間の言葉の代わりとしてふさわしい定義であろうということ）。

Turing 機械の停止問題は計算不能である

- 「Turing機械とその入力データが与えられたとき、そのTuring機械が有限時間内に停止するか」という問題。
- その問題を解けるTuring機械がMが存在したとする。そこから、あるTuring機械とそれへの入力が与えられたとき、Mを利用して、Mが有限時間内で止まると分かれば永遠に動き続け、Mが有限時間内に止まらないと分かれば停止する、そのようなTuring機械 M* が構成できる。
- M* に入力として M*自身を与えたとする (入力データは何？) 。それが有限時間内に停止すれば、それは M* が有限時間内に止まらないことを意味するので、矛盾。それが永遠に動き続ければ、定義から M* は有限時間で停止しなければならないはずで、矛盾。ゆえに、そのようなTuring機械 M は存在しない。

ゲーデル文とゲーデルの不完全性定理

ある無矛盾な論理体系が完全であるとは、その体系内の論理式がすべて証明可能か反証可能であることを言う。

命題論理(術語計算) は、完全であることがゲーデルによって示されています。そのあとにゲーデルは第一不完全性定理を証明しています。及び1階述語論理(術語計算)も完全であることが示されています。

ゲーデル文(G)

(G)　「この命題 G の証明は存在しない。」

矛盾を含まない論理体系で上のような述語論理式(G)が論理体系内に存在する時、(G) が証明できれば、矛盾。ゆえに、体系 が無矛盾であれば、(G) は証明不能である。
つまり、体系T は不完全である。

ゲーデル数の割り当て

定数項　1 から 10 の整数を割り当てる

$$\cup \rightarrow 2, \qquad \exists \rightarrow 4$$

数詞変項　11 より大きい素数を割り当てる

$$x \rightarrow 11, \qquad y \rightarrow 13$$

文変項　10 よりも大きな素数の平方を割り当てる

$$0 = 0 \rightarrow 11^{2}, \qquad p \supset q \rightarrow 13^{2}$$

述語変項　10 よりも大きな素数の立方を割り当てる

$11^{3} \rightarrow$　素数である

$13^{3} \rightarrow$　よりも大きい

式のゲーデル数

$$(\quad \exists \quad x \quad) \quad (\quad x \quad = \quad s \quad y \quad)$$

8 4 11 9 8 11 5 7 13 9

$$2^8 \times 3^4 \times 5^{11} \times 7^9 \times 11^{13} \times 13^{11} \times 17^5 \times 19^7 \times 23^{13} \times 29^9$$

すべての式は、そのゲーデル数から一意に決まる。

$(p \cup p) \Rightarrow p$ $\quad 2^8 \times 3^{11^2} \times 5^2 \times 7^{11^2} \times 11^9 \times 13^3 \times 17^{11^2} = a$

$(p \cup p)$ $\quad 2^8 \times 3^{11^2} \times 5^2 \times 7^{11^2} \times 11^9 = b$

第 2 式は第 1 式の前提部分である ⟺ b は a の約数である

左の言明は、右の算術的関係に置きかえることができる

ゲーデル数 x をもつ式系列は、ゲーデル数z を持つ式の証明になっている。
→ この関係を式で表したものを $Dem(x,z)$ とする。

$(\exists x)(x = sy)$ この式のゲーデル数を m とする

この y に m を代入したものを考える $(\exists x)(x = sm)$

y のゲーデル数は 13 なので、このゲーデル数は、m と 13 のある算術的関数としてひとつの数を定める。これを以下のように書くことにする。

$$sub(m,13,m)$$

ゲーデル式を実際に具体的に書くと、次である。(ナーゲル、ニューマン「ゲーデルは何を証明したか」白揚社 p.121)

(G) $(x)\sim Dem(x, sub(n,13,n))$

この式 (G) は「ゲーデル数 sub(n,13,n) をもつ式は証明不可能であるという数学的言明でかつそれを算術計算の中に写像したものである。この算術式は G が証明可能でないという超数学的言明を計算の内部で表している。

ゲーデルの第一不完全定理

論理体系が無矛盾で算術(初等的な自然数論)を含むとすると、その体系内で証明も反証もできない命題が存在する。つまり、体系は不完全である。

ゲーデルの第一不完全性定理は、要するに「基本算術（初等的数論）をどのようにして公理化してもそれが無矛盾であれば、その体系内で証明することも反証することもできない論理式（命題）がある」ということにつきる。

それを無視した誤用が多く、

「数学外での不完全性定理の応用もどきは、基本算術を含むという本質的な条件をしばしば無視して、この定理が形式体系一般についての定理であるかのような間違った定式化を行っている。」
（フランセーン「ゲーデルの定理」みすず書房, p.39.）

人文社会学系の人がしばしば

「ゲーデルの不完全性定理によって、人間の知の限界が露わになった」

という意味の文章をしばしば書くけれど、これは意味がない。

- P.147下から5行目

  たとえば6.1.3項の最短経路を探す問題のように、たくさんの経路の中から最も良い回を探すような問題では、現在のコンピュータではすべての組合せを一つ一つ順に調べている

  →これは間違いです

- p.149 12行から14行

  先の一筆書きをするような…..計算量のオーダーが

  n^k となるようなアルゴリズムは見つかっていない

  →これは間違いです

p.150下から3行

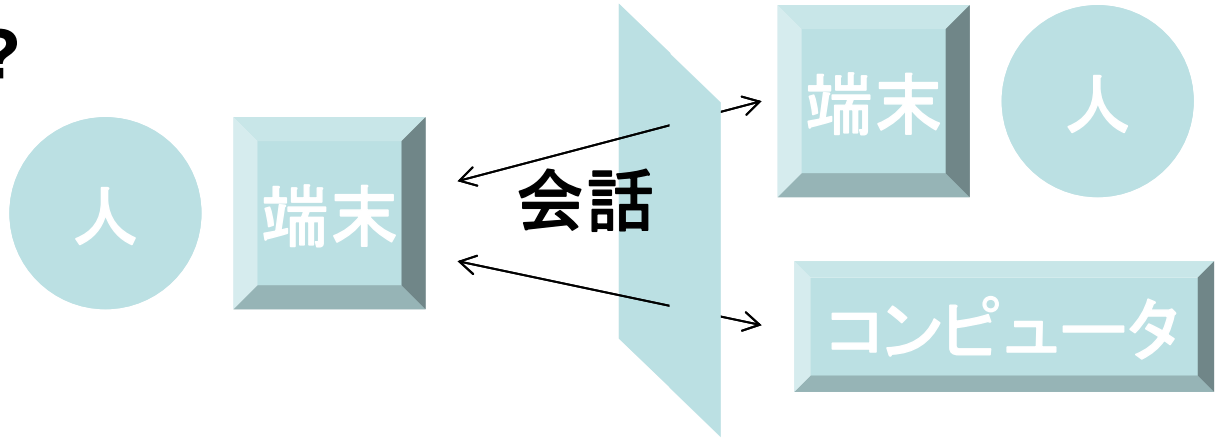
「矛盾のない論理学の体系には必ず証明できない命題がある」というゲーデルの不完全定理

→これは間違いです。命題論理、1階述語論理は完全です。

世間に流布しているチューリングテストの説明

- 「コンピュータは人間の知能を模倣できるか?」
  → チューリングテスト:

**話し相手は人か
コンピュータか?**



イミテーション・ゲーム

チューリングが最初に提示した本当のチューリングテスト

原文：・Turing, “Computing Machinery and Intelligence,” Mind, Vol. LIX, No.236, 1950.　　翻訳：・”計算機械と知能” 「マインズ・アイ(上)」ホフタッター、ベネット著、NTT コミュニケーションズ, 1992, pp.70—93.

男性（A）、女性（B）、と質問者（男性でも女性でもよい）の３人で行われる。質問者、男性、女性は別の部屋にいる。このゲームでの質問者の目的は、この二人のうち、どちらが男性であり、どちらが女性であるかを確定することである。

質問者は、２つの部屋と通信回線（メイルと思えばよい）で会話することができる。

彼はこのふたりを X 及びY という呼び名で知っており、ゲームの終わりに、彼は「 X が A であり、Y が B である」もしくは「 X が B であり、Y が A である」と述べることになる。
プレイヤー B(女性) は質問者を正解のほうに援助することを目的とし、プレイヤー A(男性) はその逆を目的とする。
「このゲームにおける男性の役割を機械が演じるとしたらどういうことになるだろうか」。質問者は人間相手のときと同じくらいの頻度で誤った決定を下すだろうか。
これが「機械が考えることができるか」に取って代わる問題である。

（注：本当だろうか？）

私の信ずるところによれば、平均的な質問者が質問を5分したあとで正しい同定を行う機会が70パーセントを超えないように上手にイミテーション・ゲームをすることができて、およそ 10^9 の記憶容量を持つ機械のプログラムを作成することは、およそ50年もたてば可能である。「機械は考えることができるか」という最初の問題設定は余りにも無意味で、議論するに値しない。
今世紀の終わりには、機械が考えるということについてもはや反論される心配なしに語ることができるようになると思う。(マインズ・アイ(上) p.75)