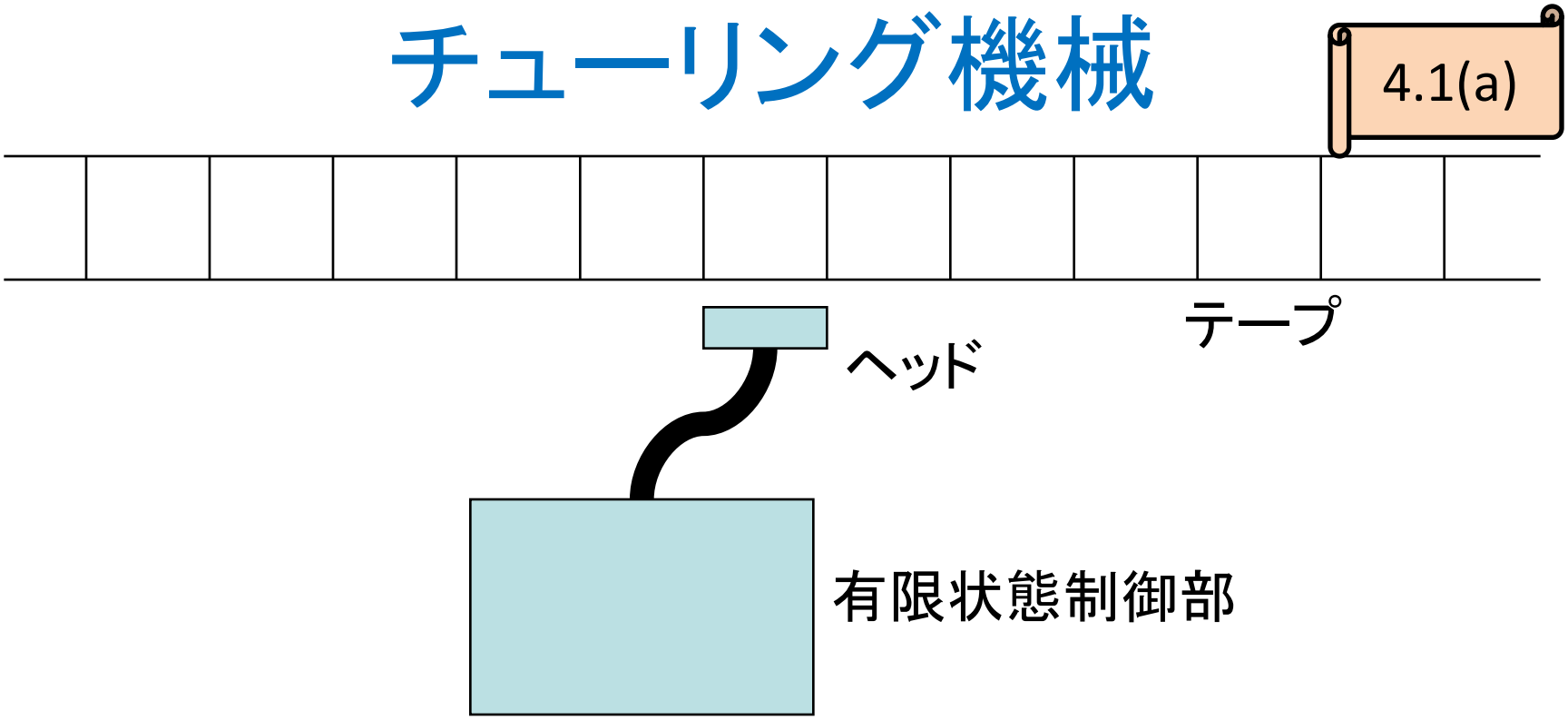
情報論理の壺３

萩谷

計算可能な関数

- チューリング機械によって計算される関数
  - チューリング計算可能
- 原始帰納的関数
  - 0・S( )・射影
  - 合成・原始帰納法
- 部分帰納的関数
  - 最小化
    - μx. ... は ... を満たす最小の x を表す。
    - そのような x が存在しなければ定義されない（止まらない）。
  - チューリング機械によって計算される関数と同じ。
- 帰納的関数
  - 部分帰納的関数のうち、いつも必ず停止するもの。

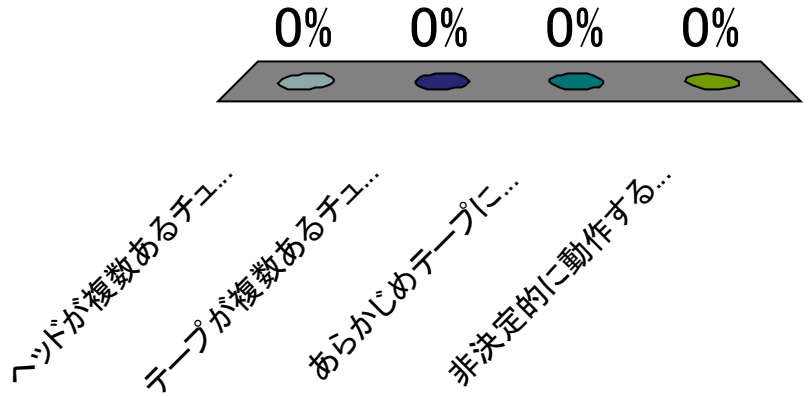
チューリング機械

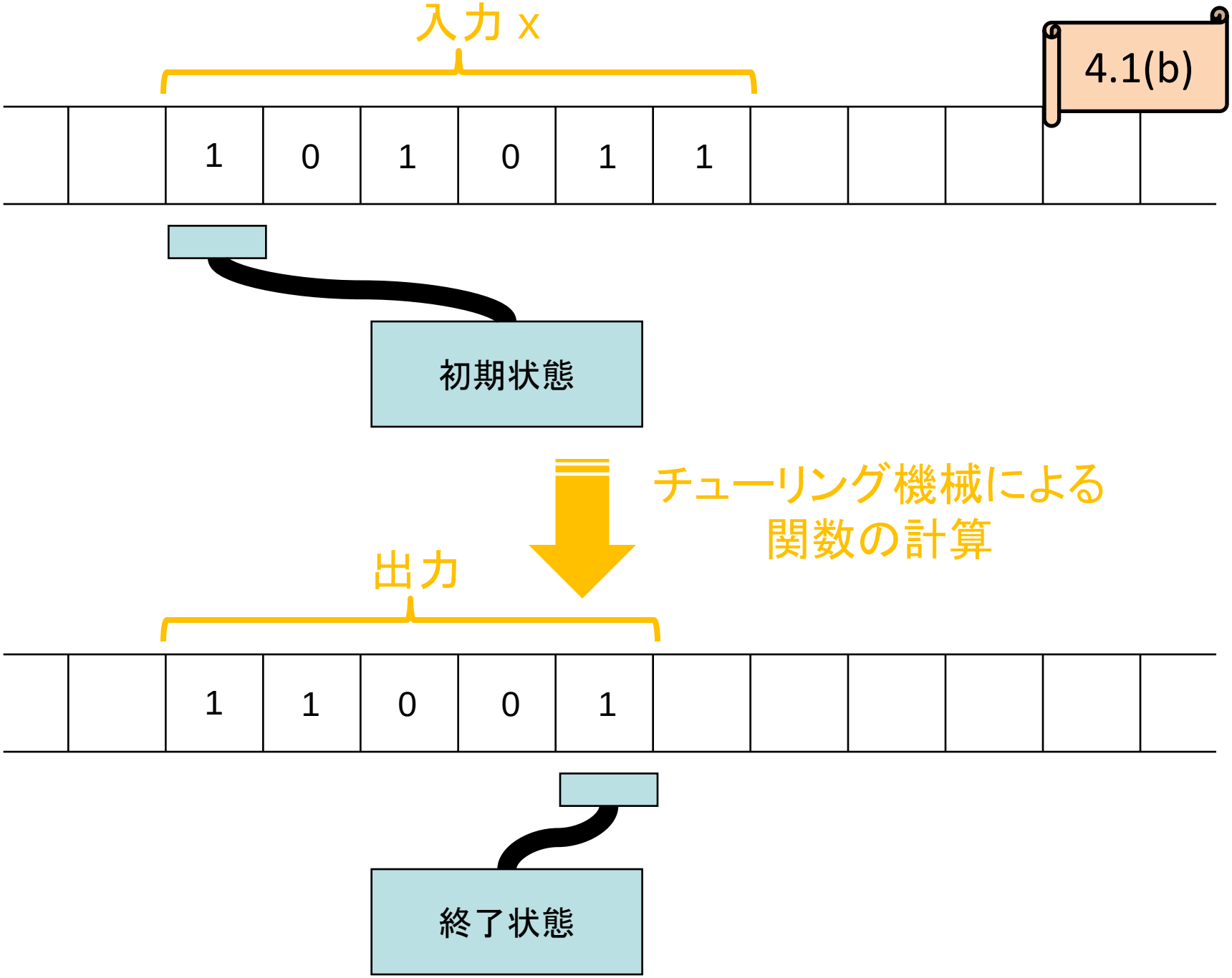


- チューリング機械の遷移規則:
  (現在の状態, ヘッドが読む文字) →
  (次の状態, ヘッドが書く文字, ヘッドの移動方向)
- このような規則を集めたものが遷移テーブル。
  すなわち、チューリング機械のプログラム。

計算能力がより高いのは

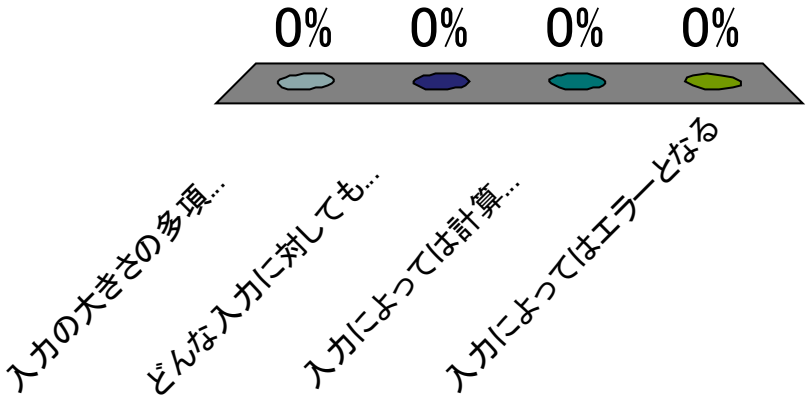
1. ヘッドが複数あるチューリング機械
2. テープが複数あるチューリング機械
3. あらかじめテープに無限個の文字が書かれているチューリング機械
4. 非決定的に動作するチューリング機械




入力 x
4.1(b)
1
0
1
0
1
1
初期状態
チューリング機械による
関数の計算
出力
1
1
0
0
1
終了状態

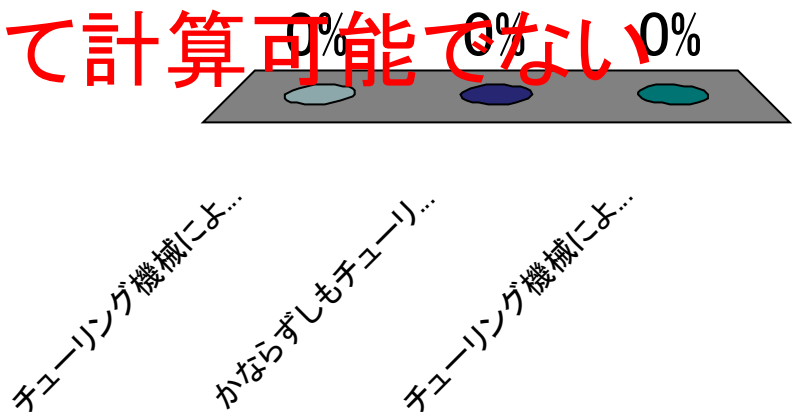
チューリング機械は

1. 入力の大きさの多項式時間で答えを出す
2. どんな入力に対しても止まって答えを出す
3. 入力によっては計算が止まらない
4. 入力によってはエラーとなる



f(x) と g(x) がチューリング機械によって計算可能ならば f(g(x)) は

1. チューリング機械によって計算可能
2. かならずしもチューリング機械によって計算可能とは限らない
3. チューリング機械によって計算可能でない



原始帰納的関数

4.2(a)

- 0 はアリティ0の関数（定数）として原始帰納的。
- 関数 $S(x)=x+1$ はアリティ1の原始帰納的関数。
- 射影 $f(x_1,\ldots,x_n)=x_i$ は、各 n と i に対して、原始帰納的。
- f と g_i が原始帰納的ならば、合成

  $h(x_1,\ldots,x_n)=f(g_1(x_1,\ldots,x_n),\ldots,g_m(x_1,\ldots,x_n))$

  も原始帰納的。

原始帰納的関数

4.2(a)

- f と g が原始帰納的ならば、

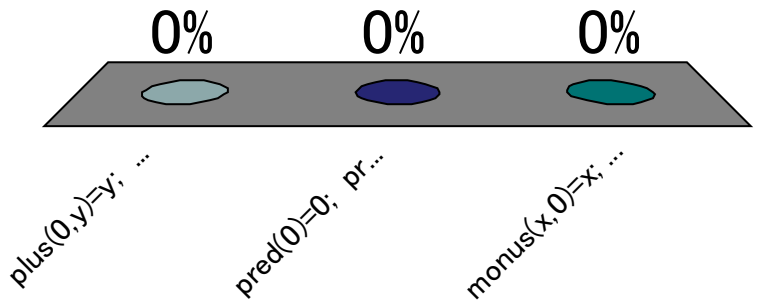
  $h(0,x_1,\ldots,x_n)=f(x_1,\ldots,x_n)$

  $h(S(x),x_1,\ldots,x_n)=g(x,h(x,x_1,\ldots,x_n),x_1,\ldots,x_n)$

  と再帰的に定義される関数 h も原始帰納的。（原始帰納法）
- 以上の規則を有限回適用して得られる関数のみが、原始帰納的。

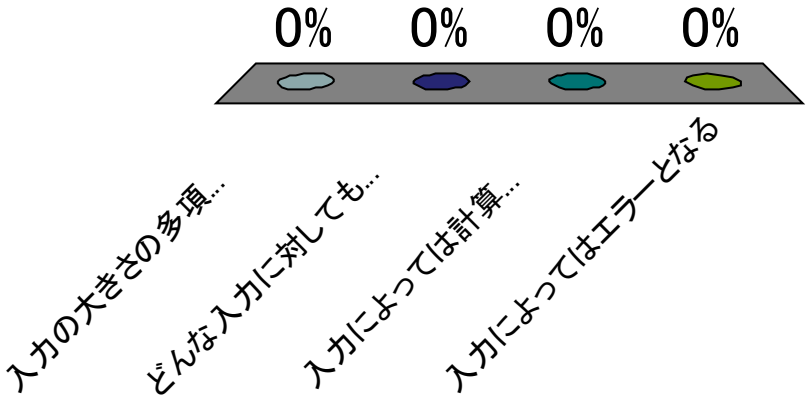
正しい原始帰納法は

1. plus(0,y)=y; plus(S(x),y)=plus(x,S(y))
2. pred(0)=0; pred(S(x))=x
3. monus(x,0)=x;
   monus(S(x),S(y))=monus(x,y)



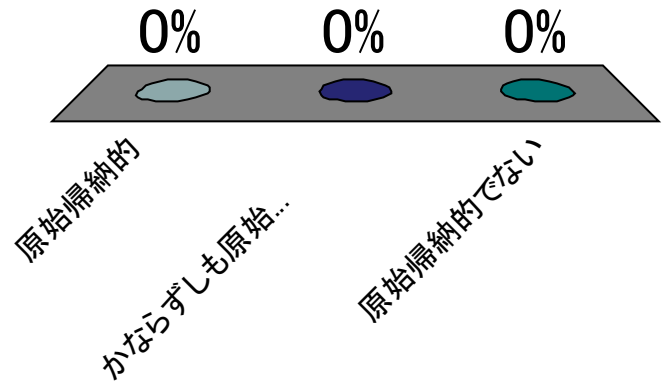
原始帰納的関数は

1. 入力の大きさの多項式時間で答えを出す
2. どんな入力に対しても止まって答えを出す
3. 入力によっては計算が止まらない
4. 入力によってはエラーとなる



f(x) と g(x) が原始帰納的ならば f(g(x)) は

1. 原始帰納的
2. かならずしも原始帰納的とは限らない
3. 原始帰納的でない



原始帰納的関数の例

4.2(a)

- plus(x,y) = x+y
- pred(x) = if x>0 then x-1 else 0
- monus(x,y) = if x≧y then x-y else 0
- times(x,y) = x*y

上は原始帰納的関数としての定義ではない。

階乗関数を
原始帰納的関数として定義せよ。

フィボナッチ関数を
原始帰納的関数として定義せよ。

二つの自然数を一つの自然数に
符号化する関数 p(x,y) および
z=p(x,y) から x と y を取り出す関数
$p_1(z)$ と $p_2(z)$ は、原始帰納的関数として
使ってもよい。
足し算も原始帰納的関数として使ってよい。

有界最小化

4.2(a)

- f(x,y) は原始帰納的関数とする。

```
int g(int x, int z) {
    int y;
    for (y = 0;  y < z;  y++)
        if (f(x,y) == 0) return y;
    return 0;
}
```

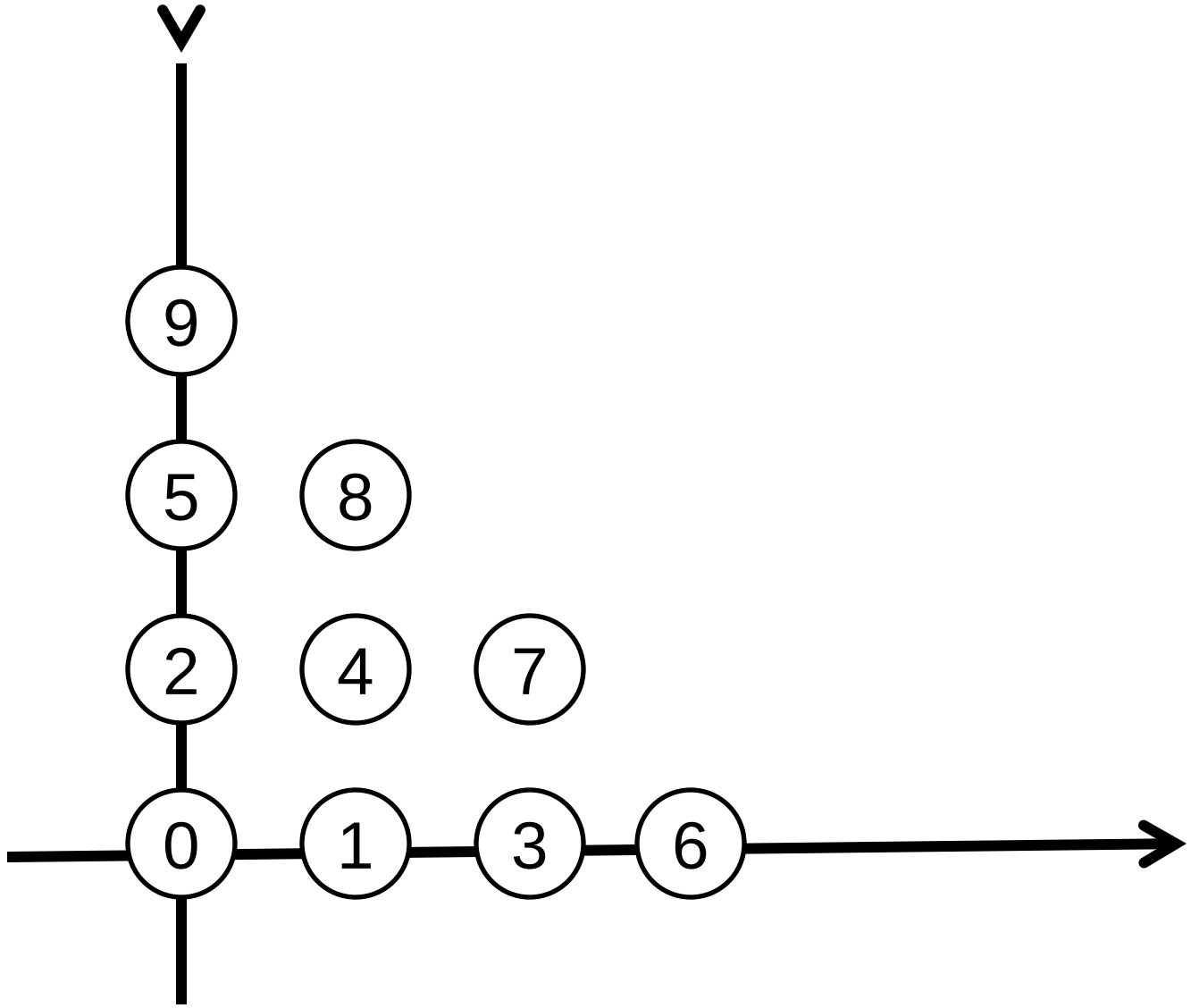
- g(x,z) も原始帰納的。

原始帰納的関数の例

4.2(a)

- div2(x)=x/2
- p(x,y)=div2((x+y+1)*(x+y))+y
- z=p(x,y) から x と y を取り出す関数 $p_1(z)$ と $p_2(z)$
- $y=p(x_0,p(x_1,\ldots,p(x_{n-1},0)\ldots))$ から x_i を取りだす関数 b(y,i)

上は原始帰納的関数としての定義ではない。


y
9
5
8
2
4
7
0
1
3
6

拡張された帰納法

4.2(a)

- k と g が原始帰納的であり、
  f(x)≠0 ならば必ず g(x)<x が成り立つとき、
  h(x) = if f(x)=0 then c else k(x,h(g(x)))
  と定義される h も原始帰納的になる。

原始帰納的でない関数

- 次のように定義される関数ack(m,n)の計算は、任意の自然数m,nに対して止まるが、

  ack(0,n) = n+1
  ack(m+1,0) = ack(m,1)
  ack(m+1,n+1) = ack(m,ack(m+1,n))

  この関数は原始帰納的ではない。

部分帰納的関数

4.2(b)

- f(x,y) と g(y) は原始帰納的関数とする。

```
int h(int x) {
    int y = 0;
    while (f(x,y) != 0)
        y++;
    return g(y);
}
```

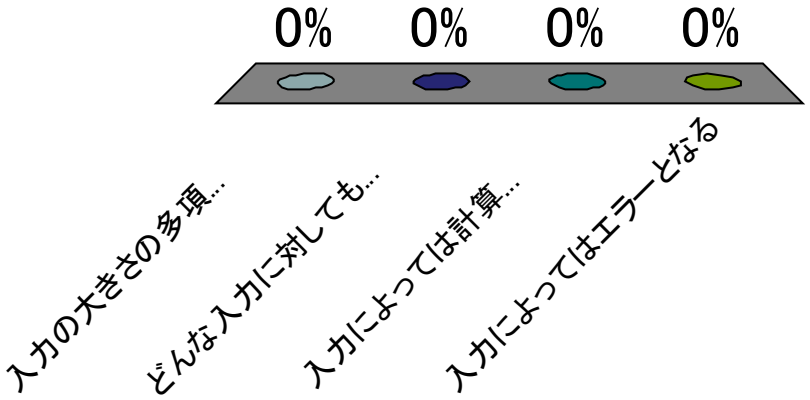
μy. f(x,y)=0

自然数から自然数への計算可能な関数は、
すべてこのような形で書くことができる。
（⇒チューリング機械の符号化）

- h(x) = g(μy. f(x,y)=0) は部分帰納的。

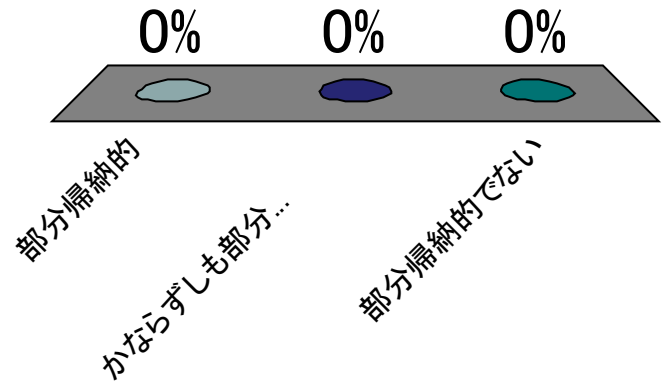
部分帰納的関数は

1. 入力の大きさの多項式時間で答えを出す
2. どんな入力に対しても止まって答えを出す
3. 入力によっては計算が止まらない
4. 入力によってはエラーとなる



f(x) と g(x) が部分帰納的ならば f(g(x)) は

1. 部分帰納的
2. かならずしも部分帰納的とは限らない
3. 部分帰納的でない



h1(x) と h2(x) が部分帰納的ならば h2(h1(x)) は部分帰納的を示せ。

h1(x) = g1(μy. f1(x,y)=0)

h2(x) = g2(μy. f2(x,y)=0)

であるとき、

$f(x,y) = \mathbf{if}\ (\forall z.\ z<p_1(y) \supset f1(x,z)>0) \wedge$
$f1(x,p_1(y))=0 \wedge$
$(\forall z.\ z<p_2(y) \supset f2(g1(p_1(y)),z)>0) \wedge$
$f2(g1(p_1(y)),p_2(y))=0\ \mathbf{then}\ 0\ \mathbf{else}\ 1$

$g(y) = g2(p_2(y))$

帰納的関数

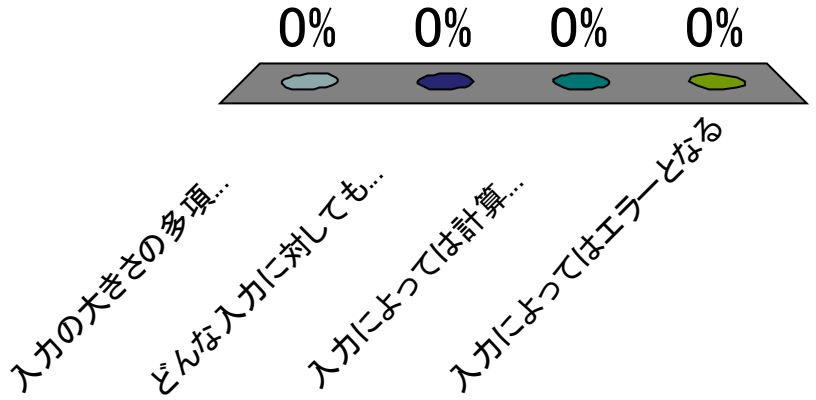
4.2(b)

- 部分帰納的関数が全関数であるとき、帰納的関数という。

関数 h(x) が帰納的になる条件は？

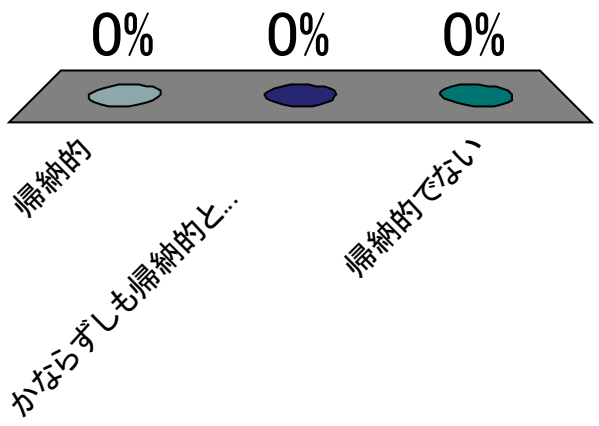
帰納的関数は

1. 入力の大きさの多項式時間で答えを出す
2. どんな入力に対しても止まって答えを出す
3. 入力によっては計算が止まらない
4. 入力によってはエラーとなる



f(x) と g(x) が帰納的ならば f(g(x)) は

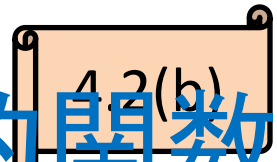
1. 帰納的
2. かならずしも帰納的とは限らない
3. 帰納的でない



原始帰納的関数は
帰納的であることを説明せよ。

帰納的だが、
原始帰納的でない関数をあげよ。

チューリング機械と部分帰納的関数



- 部分帰納的ならばチューリング計算可能
  - チューリング機械で原始帰納的関数をシミュレート
    - テープは無限のスタックとして使う
    - まぁ、できるでしょう…
  - 部分帰納的関数をシミュレート
    - ループを回せばよい
- 逆は？

チューリング機械の符号化

4.2(b)

- チューリング機械の遷移テーブルを自然数 e に符号化。
- チューリング機械の状態とヘッドの位置とテープの空白でない有限部分を一つの自然数に符号化。 チューリング機械の状況
- さらに、チューリング機械による実行過程を一つの自然数 y に符号化。
  - 上のような自然数の有限列。
  - 停止する計算過程のみを考える。
- 原始帰納的関数 T(e,x,y)（Kleeneの述語という）

  T(e, x, y) = 0 if y はチューリング機械 e が入力 x で実行されたときの実行過程

  T(e, x, y) = 1 otherwise
- 原始帰納的関数 U(y) は実行過程 y から出力を取り出す。
- すると、チューリング機械 e の計算する関数 f(x) は、

  $f(x) = U(\mu y.\ T(e, x, y)=0)$

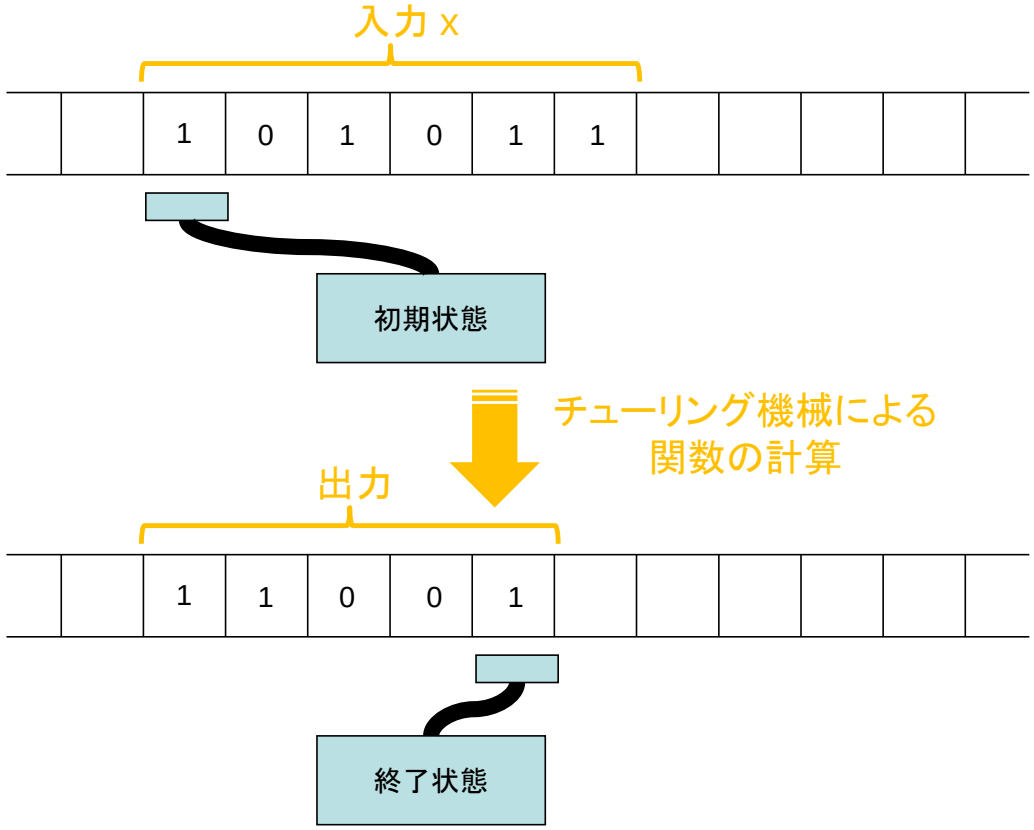
チューリング機械の状況

4.2(b)

- チューリング機械の状態とヘッドの位置とテープの空白でない有限部分

- 初期状況 c_0
  c_1
  c_2
  :
- 終了状況 c_f

実行過程



4.2(b)

T(e,x,y)

- 遷移テーブルの符号が e である
  チューリング機械が入力 x で実行されたときの終了状況に至る実行過程の符号が y ならば、0 を返す。
- そうでなければ、1 を返す。
- チューリング機械 e が入力 x に対して停止するとは、∃y. T(e,x,y)=0
- その出力は、U(μy. T(e,x,y)=0)
  - 部分帰納的！

遷移テーブルの符号 e と
自然数 y が与えられたとき、
y が e の実行過程の符号か
どうかを判定するには、
どうすればよいか。

チューリング機械の符号化

4.2(b)

- チューリング機械の遷移テーブルを自然数 e に符号化。
- チューリング機械の状態とヘッドの位置とテープの空白でない有限部分を一つの自然数に符号化。 チューリング機械の状況
- さらに、チューリング機械による実行過程を一つの自然数 y に符号化。
  - 上のような自然数の有限列。
  - 停止する計算過程のみを考える。
- 原始帰納的関数 T(e,x,y)（Kleeneの述語という）

  T(e, x, y) = 0 if y はチューリング機械 e が入力 x で実行されたときの実行過程

  T(e, x, y) = 1 otherwise
- 原始帰納的関数 U(y) は実行過程 y から出力を取り出す。
- すると、チューリング機械 e の計算する関数 f(x) は、

  $f(x) = U(\mu y.\ T(e, x, y)=0)$

帰納的に可算な集合・帰納的集合

- 自然数の集合 X が帰納的に可算であるとは、
  ある部分帰納的な関数 f(x) が存在して、
  $x \in X \Leftrightarrow$ f(x):定義されている（止まる）
- X が帰納的であるとは、
  ある帰納的な関数 f(x) が存在して、
  $x \in X \Leftrightarrow f(x) = 0$
- もちろん、帰納的ならば帰納的に可算。
- 以下は同値
  - X は帰納的。
  - X と X の補集合の両方が帰納的に可算。

部分帰納的関数

4.2(b)

- f(x,y) と g(y) は原始帰納的関数とする。

```
int h(int x) {
    int y = 0;
    while (f(x,y) != 0)
        y++;
    return g(y);
}
```

μy. f(x,y)=0

自然数から自然数への計算可能な関数は、
すべてこのような形で書くことができる。
（⇒チューリング機械の符号化）

- h(x) = g(μy. f(x,y)=0) は部分帰納的。

部分帰納的関数
$h(x) = g(\mu y.\ f(x,y)=0)$ が
定義されるような x の集合とは?

なぜ、帰納的ならば帰納的に可算か？

$h(x) = g(\mu y.\ f(x,y)=0)$ が帰納的であるとき、

$h(x)=0$ となる条件は？

$(\forall z.\ z<y \supset f(x,z)>0) \wedge f(x,y)=0 \wedge g(y)=0$

という条件を考えてみよ。

Ｘと Ｘ の補集合の両方が帰納的に可算ならば、なぜ Ｘ は帰納的か

Ｘ = { x | ある y が存在して f(x,y)=0 }

Ｘの補集合 = { x | ある y が存在して g(x,y)=0 }

としたとき（f と g は原始帰納的）、

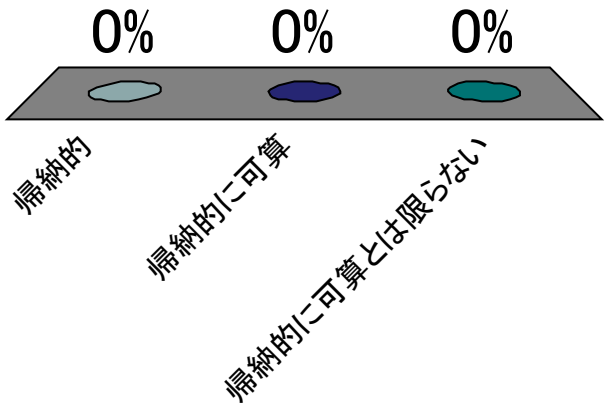
$h(x,y) = \text{if } x \neq p_1(y) \text{ then } 1$

$\text{else } f(p_1(y),p_2(y))*g(p_1(y),p_2(y))$

として $k(y) = f(p_1(y),p_2(y))$ とすると…

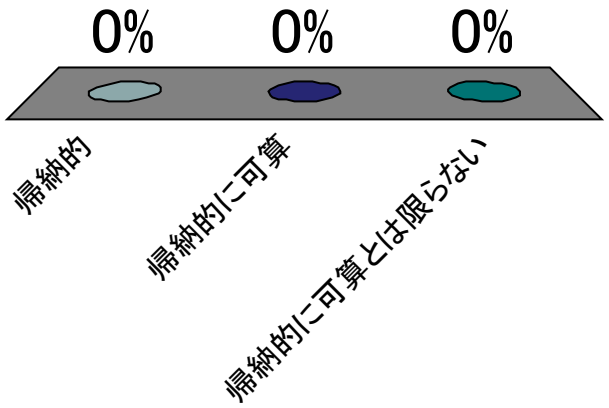
帰納的な集合 X と Y に対して X∩Y は

1. 帰納的
2. 帰納的に可算
3. 帰納的に可算とは限らない



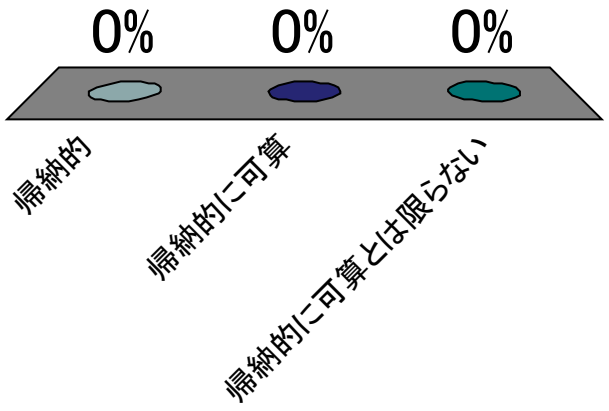
帰納的な集合 X と Y に対して X∪Y は

1. 帰納的
2. 帰納的に可算
3. 帰納的に可算とは限らない



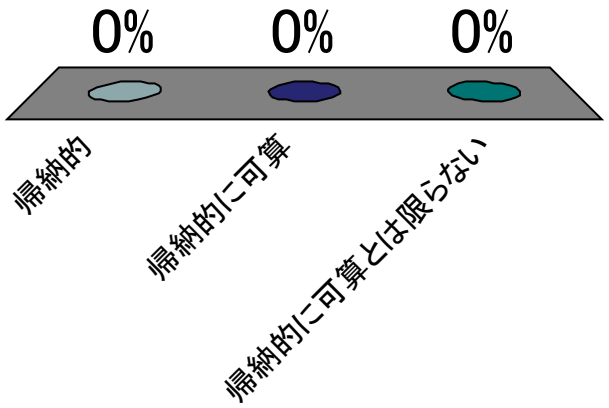
帰納的な集合 X に対して
X の補集合は

1. 帰納的
2. 帰納的に可算
3. 帰納的に可算とは限らない



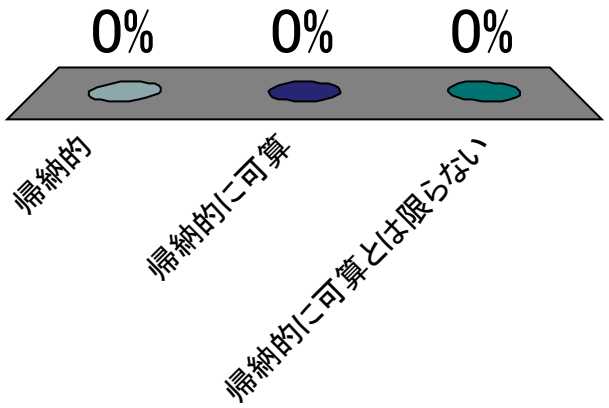
帰納的な集合 X に対して
$\{x \mid \exists y.\ p(x,y) \in X\}$ は

1. 帰納的
2. 帰納的に可算
3. 帰納的に可算とは限らない



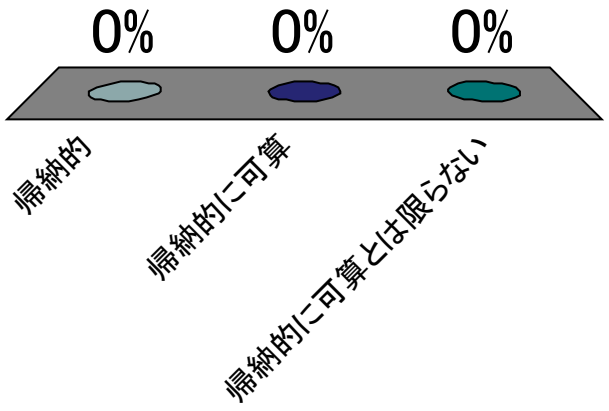
帰納的に可算な集合 X と Y に対して X∩Y は

1. 帰納的
2. 帰納的に可算
3. 帰納的に可算とは限らない



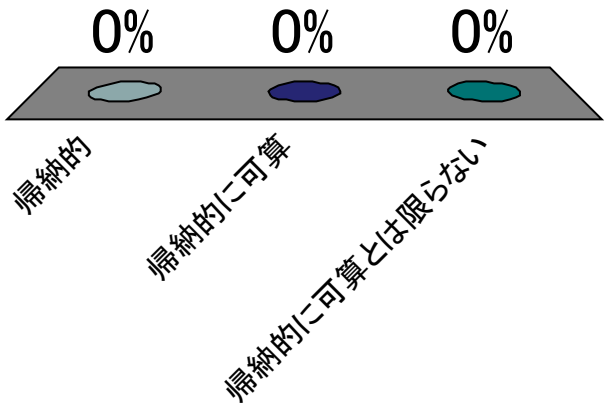
帰納的に可算な集合ＸとＹに対してＸ∪Ｙは

1. 帰納的
2. 帰納的に可算
3. 帰納的に可算とは限らない



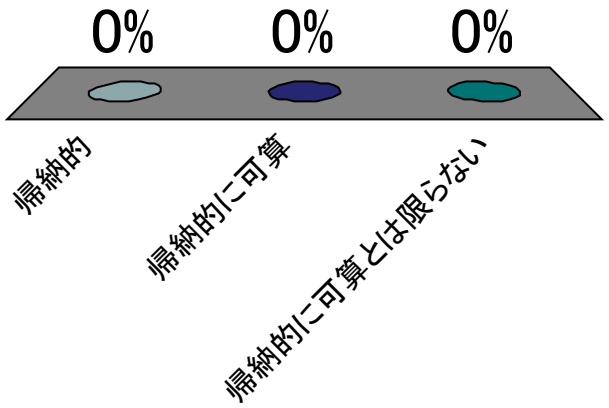
帰納的に可算な集合 X に対して X の補集合は

1. 帰納的
2. 帰納的に可算
3. 帰納的に可算とは限らない



帰納的に可算な集合 X に対して $\{x \mid \exists y.\ p(x,y) \in X\}$ は

1. 帰納的
2. 帰納的に可算
3. 帰納的に可算とは限らない



チューリング機械の停止問題

2.2(c)

- チューリング機械 e が入力 x に対して
  停止するかどうかは判定できない。
  - 次のような計算可能関数 h(e,x) は存在しない。

    h(e,x) = 0 if e が x に対して停止する。

    h(e,x) = 1 otherwise
- これに関連して、

  K = { x | T(x,x,y)=0 を満たす y が存在する }

  とおくと、
  - K は帰納的に可算だが帰納的でない。
  - K の補集合は帰納的に可算でない。

K が帰納的でないことを示せ。

K が帰納的であるとすると、帰納的な関数 f を用いて $K = \{x \mid f(x)=0\}$ と書ける。
$f(x)=0$ ならば無限ループ、$f(x)\neq 0$ ならば停止するチューリング機械の符号を x_0 として、
このチューリング機械に x_0 を入力すると…

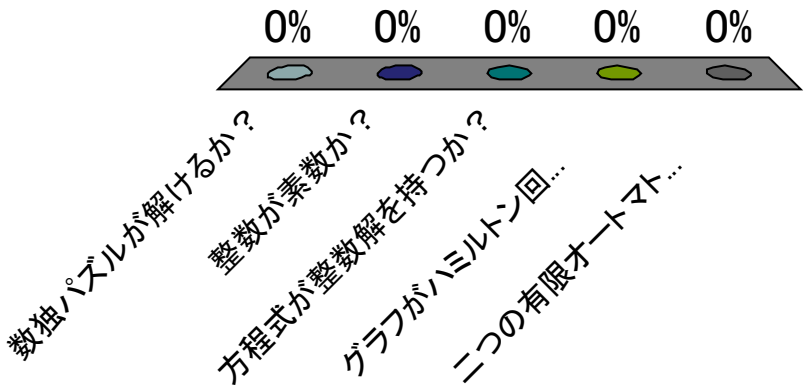
決定可能

4.2(c)

- 判定問題の事例を自然数に符号化したとき、条件を満たす事例の集合が帰納的ならば、その問題は決定可能（可解）であるという。
- そうでないとき、決定不能（非可解）
- チューリング機械の停止性は決定不能。

決定不能な問題は？

1. 数独パズルが解けるか？
2. 整数が素数か？
3. 方程式が整数解を持つか？
4. グラフがハミルトン回路を持つか？
5. 二つの有限オートマトンが認識する言語が同じか？



決定不能な問題の例

- 連立高次方程式が整数解を持つか？
- 二つの文脈自由文法が生成する言語が同じか？