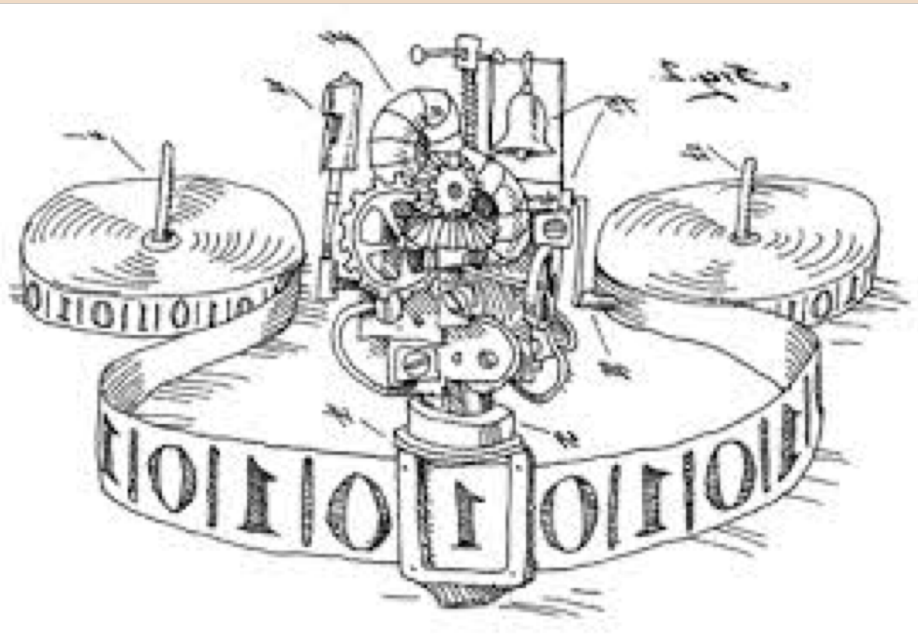


楽しい数学 第三夜

計算理論入門



「やさしい計算」と「むずかしい計算」

丸山不二夫

はじめに

我々は、小学生から計算することを学ぶ。数学者も、いつも計算をしている。小論の副題を「やさしい計算」と「難しい計算」としたのだが、それでは、小学生の計算はやさしくて、数学者の行う計算は難しいのだろうか？

もちろん、それはそうなのだろうが、見方を変えれば、小学生の算数も結構難しいことに気づくし、数学者が行なっているのも「自明な計算」であるようにも思う。

重要なことは、両者の行なっている「計算」は、それらが「チューリング・マシン」で計算可能であるという意味では、数学的には同じタイプのものであるということである。

計算可能性についての「チャーチ＝チューリングの提言」は、そうしたことを主張しているように見える。

はじめに

それでは、「やさしい計算」と「難しい計算」という違いは、原理的には存在しないのであろうか？

実は、そうではない。

歴史的には、まず、数学者が自分たちには計算できない問題があることに気づき(ある意味では、これほど「難しい計算」はないのだが)、そこで「計算できること」の定義をあらためて考えたのが、先の「チャーチ=チューリングの提言」である。

ただ、ここで問題になっているのは、「計算できること」と「計算できないこと」の区別であって、「やさしい計算」と「難しい計算」の区別ではない。

やがて、この「やさしい計算」と「難しい計算」の階層的区別を考える数学の一分野が成立する。それが「計算複雑性理論」である。

はじめに

小論は、こうした「計算理論」の入門編である。

同時に、筆者が意識していたのは、人間の「知能」の本質を「計算」として抽象しようとする人工知能論における「計算主義」の考え方である。

「知能」に対する「計算主義的」なアプローチは、僕が人工知能が将来実現すべき課題として関心を持っている、人間の「言語能力」や日常的な「論理的推論能力」、抽象的な「数学的能力」、さらには「科学する能力」の機械による実現には、そうしたアプローチが不可欠だと考えている。

もちろん、人間の「知能」にまつわる問題を、一つの方法論で対応するのは難しい。それについては、「知能の階層性」で、述べようと思う。

Agenda

- 小学生の算数を考える
 - 理論とモデル -- 「ルール」と「たとえ」の数理
 - 「計算できないもの」の発見 -- ゲーデルの不完全性定理
 - チューリング・マシンで計算する
 - 計算可能だが、とても「計算が難しい」ものがある
 - 複雑さについての多様なアプローチ
 - 「やさしい計算」と「難しい計算」 -- 「計算複雑性理論」
 - 複雑性理論の発展
 - 複雑性理論の現在と人工知能論の課題
-

小学生の算数を考える

小学算数の筆算のルールを考える

思考実験1

二桁の数どうしの計算について、小学校で習った足し算・引き算・掛け算の計算の仕方を、すべて「文章」で記述してみよう。それぞれ、400字以内にまとめられるか？

加減乗除の筆算の例

$$\begin{array}{r} 64 \\ + 37 \\ \hline 101 \end{array}$$

$$\begin{array}{r} 64 \\ - 37 \\ \hline 27 \end{array}$$

$$\begin{array}{r} 64 \\ \times 37 \\ \hline 448 \\ 192 \\ \hline 1368 \end{array}$$

$$\begin{array}{r} 2 \\ \hline 37 \overline{) 64} \\ \underline{64} \\ 0 \end{array}$$

そのルールを書き出してみよう

足し算の筆算のルール

1. 一番下の桁の二つの数を足す。

$$4 + 7 = 11$$

$$\begin{array}{r} 64 \\ + 37 \\ \hline \end{array}$$

2. その答えの一番下の桁の数字を下に書く。

$$4 + 7 = 11$$

$$\begin{array}{r} 64 \\ + 37 \\ \hline 1 \end{array}$$

3. その答えが、10未満だったら、次の桁に進む。
10以上だったら、次の桁の上に、小さく、繰り上がりの1を書く。

$$\begin{array}{r} 1 \\ 64 \\ + 37 \\ \hline 1 \end{array}$$

4. 次の桁の二つの数と、繰り上がりがあれば、それも含めてたす。

$$1 + 6 + 3 = 10$$

$$\begin{array}{r} 1 \\ 64 \\ + 37 \\ \hline 1 \end{array}$$

5. その答えの一番下の桁の数字を
下を書く。 $1+6+3 = 10$

$$\begin{array}{r} 1 \\ 64 \\ + 37 \\ \hline 01 \end{array}$$

6. その答えが、10未満だったら、
次の桁に進む。
10以上だったら、次の桁の上に、
小さく、繰り上がりの1を書く。

$$\begin{array}{r} 1 \quad 1 \\ 64 \\ + 37 \\ \hline 01 \end{array}$$

7. 次の桁の二つの数と、繰り上がり
があれば、それも含めてたす。
 $1+0+0=1$

$$\begin{array}{r} 1 \quad 1 \\ 064 \\ + 037 \\ \hline 01 \end{array}$$

8. その答えの一番下の桁の数字を
下を書く。 $1+0+0 = 1$

$$\begin{array}{r} 1 \quad 1 \\ 64 \\ + 37 \\ \hline 101 \end{array}$$

9. その答えが、10未満だったら、次の桁に進む。
10以上だったら、次の桁の上に、小さく、繰り上がりの1を書く。

$$\begin{array}{r} 1\ 1 \\ 64 \\ +\ 37 \\ \hline 101 \end{array}$$

10. たすものがなければ、計算終了。

$$\begin{array}{r} 64 \\ +\ 37 \\ \hline 101 \end{array}$$

足した結果の、一番下の桁の数字を書き込む
足した結果が、10以上か(繰り上がり)、
そうでないか(繰り上がりなし)で処理が違う。

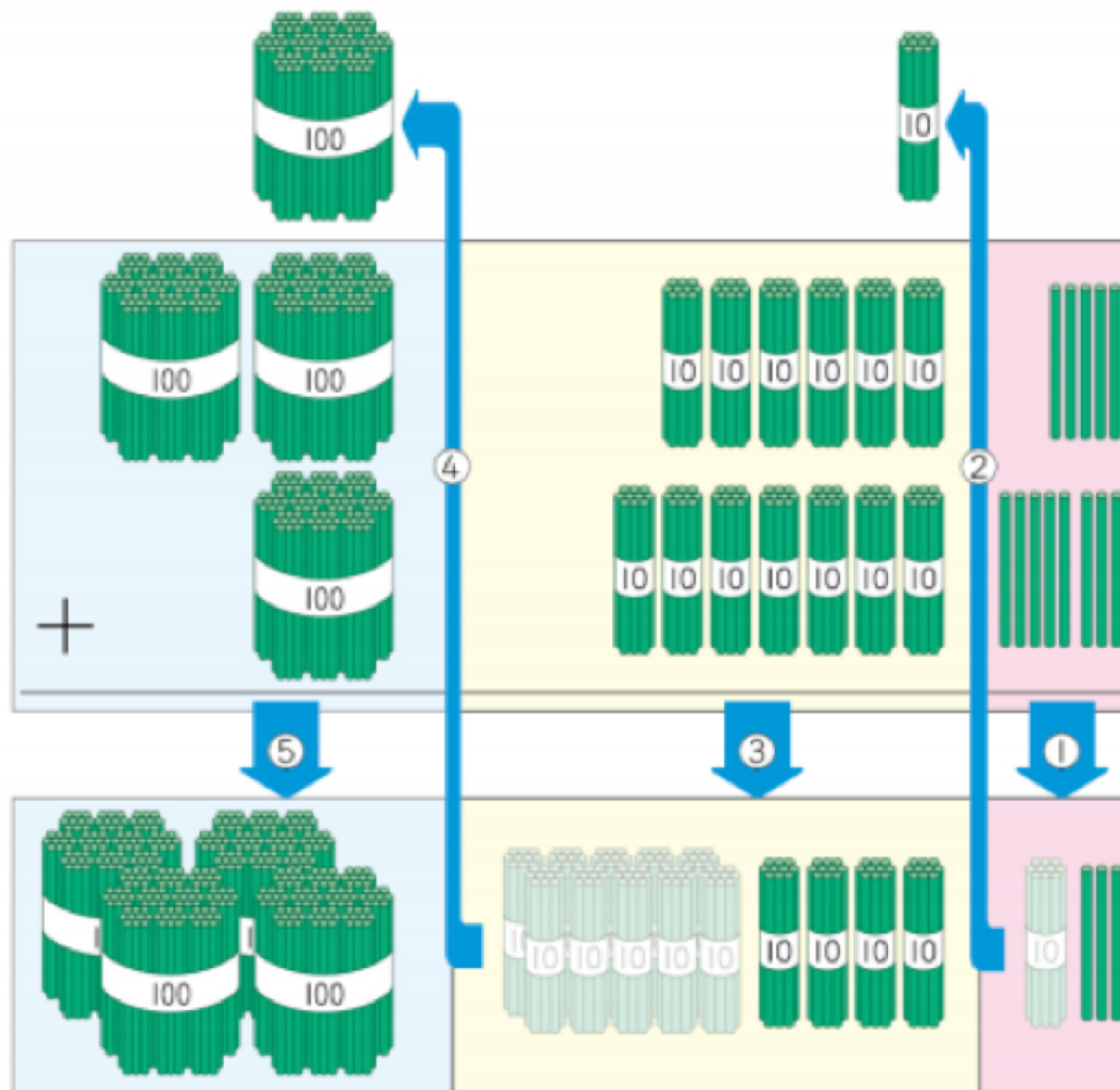
「桁」って、何？
足し算の九九が前提

3

265+178 の筆算のしかたを考えましょう。

啓林館

3年上 p.57



$$\begin{array}{r} 265 \\ + 178 \\ \hline 3 \end{array}$$

$$\begin{array}{r} 265 \\ + 178 \\ \hline 43 \end{array}$$

$$\begin{array}{r} 265 \\ + 178 \\ \hline 443 \end{array}$$

引き算の筆算のルール

1. 一番下の桁の二つの数を引く。
引けない場合は、一つ上の桁を一つ減らして、次の計算を行う。

$$14 + 7 = 7$$

$$4 + (10 - 7) = 7$$

2. その答えの数字を下に書く。

3. 次の桁の二つの数を引く。

$$5 - 3 = 2$$

引けない場合は、一つ上の桁を一つ減らして計算する。

4. その答えの数字を下に書く。

$$\begin{array}{r} 5 \\ 64 \\ - 37 \\ \hline \end{array}$$

$$\begin{array}{r} 5 \\ 64 \\ - 37 \\ \hline 7 \end{array}$$

$$\begin{array}{r} 5 \\ 64 \\ - 37 \\ \hline 7 \end{array}$$

$$\begin{array}{r} 5 \\ 64 \\ - 37 \\ \hline 27 \end{array}$$

掛け算の筆算のルール

1. 7×4 を計算する。

$$7 \times 4 = 28$$

2. この答え28の一番下の数字 **8** を書き込み、上の数字**2**は覚えておく。

3. 7×6 を計算する。

$$7 \times 6 = 42$$

4. この42に、先に覚えた2 を足して $42 + 2 =$ **44** を、上の桁に書き込む。

5. 3×4 を計算する、

$$3 \times 4 = 12$$

6. この答えの一番下の数字 **2** を、**3の下**に書き込み、上の数字**1**は覚えておく。

7. 3×6 を計算する。

$$3 \times 6 = 18$$

8. この18に、先に覚えた1を足して、 $18 + 1 =$ **19**を、上の桁に書き込む。

9. **448**と**1920**を足し算する。.....

$$\begin{array}{r} 64 \\ \times 37 \\ \hline 448 \\ 192 \\ \hline 2368 \end{array}$$

もしも、二桁の数字の
かけ算が暗算でできたら

1. 37×64 を計算する。

$$37 \times 64 = 2368$$

2. この答え2368の下の二桁の数字 **68** を
書き込み、上の数字**23**は覚えておく。

3. 37×12 を計算する。

$$37 \times 12 = 444$$

4. この444に、先に覚えた23 を足して
 $444 + 23 = 467$ を、上の桁に書き込む。

5. 34×64 を計算する、

$$34 \times 64 = 2176$$

6. この答えの下の二桁の数字 **76** を、
3桁の下に書き込み、上の数字**21**は覚えておく。

7. 34×12 を計算する。

$$34 \times 12 = 408$$

8. この408に、先に覚えた21を足して、 $408 + 21 = 429$ を、
上の桁に書き込む。

9. **46768**と**42976**を足し算する。.....

$$\begin{array}{r} \boxed{12} \boxed{64} \\ \times \boxed{34} \boxed{37} \\ \hline 46768 \\ 42976 \\ \hline 4344368 \end{array}$$

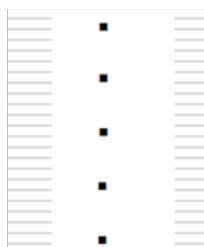
足し算でかけ算をする

1. $64+0=64$

2. $64+64=128$

3. $64+128=192$

4. $64+192=256$



35. $64+2176=2240$

36. $64+2240=2304$

37. $64+2304=2368$

割り算の筆算のルール？

$$\begin{array}{r} 1176 \\ 2 \overline{)2352} \\ \underline{2} \\ 3 \\ \underline{2} \\ 15 \\ \underline{14} \\ 12 \\ \underline{12} \\ 0 \end{array}$$

$$\begin{array}{r} 112 \\ 21 \overline{)2352} \\ \underline{21} \\ 25 \\ \underline{21} \\ 42 \\ \underline{42} \\ 0 \end{array}$$

$$\begin{array}{r} 42 \\ 56 \overline{)2352} \\ \underline{224} \\ 112 \\ \underline{112} \\ 0 \end{array}$$

$$\begin{array}{r} 8 \\ 294 \overline{)2352} \\ \underline{2352} \\ 0 \end{array}$$

かけ算で行う割り算のアルゴリズム (Silly!)

$2 \times 1 = 2$

$2 \times 2 = 4$

$2 \times 3 = 6$

$2 \times 4 = 8$

.

.

.

.

.

.

.

.

.

$2 \times 1173 = 2346$

$2 \times 1174 = 2348$

$2 \times 1175 = 2350$

$2 \times 1176 = 2352$

$2 \times 1177 = 2354$

.

.

.

$21 \times 1 = 21$

$21 \times 2 = 42$

$21 \times 3 = 63$

$21 \times 4 = 84$

.

.

.

.

.

.

$21 \times 109 = 2289$

$21 \times 110 = 2310$

$21 \times 111 = 2331$

$21 \times 112 = 2352$

$21 \times 113 = 2373$

.

.

.

$56 \times 1 = 56$

$56 \times 2 = 112$

$56 \times 3 = 168$

$56 \times 4 = 184$

.

.

.

$56 \times 39 = 2184$

$56 \times 40 = 2240$

$56 \times 41 = 2396$

$56 \times 42 = 2352$

$56 \times 43 = 2408$

.

.

.

$294 \times 1 = 294$

$294 \times 2 = 588$

$294 \times 3 = 882$

$294 \times 4 = 1176$

$294 \times 5 = 1470$

$294 \times 6 = 1764$

$294 \times 7 = 2058$

$294 \times 8 = 2352$

$294 \times 9 = 2646$

.

.

.

計算をたとえばで説明する

思考実験

「二分の一を三分の二で割る」を、「例え話」で説明せよ。

次の分数の計算を「たとえ」で説明せよ

● $\frac{1}{2} + \frac{2}{3} = \frac{7}{6}$

● $\frac{2}{3} - \frac{1}{2} = \frac{1}{6}$

● $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

● $\frac{1}{2} \div \frac{2}{3} = \frac{3}{4}$

分数計算の「ルール」

- $\frac{1}{2} + \frac{2}{3} = \frac{7}{6}$
- $\frac{2}{3} - \frac{1}{2} = \frac{1}{6}$
- $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$
- $\frac{1}{2} \div \frac{2}{3} = \frac{3}{4}$

二つの分母を「通分」して、分母を同じにしてから、分子の足す・引くを行う。

分子は分子、分母は分母でかける。

割る数の分子と分母を逆にして割られる数にかける。

$$\frac{1}{2} + \frac{2}{3} = \frac{7}{6} \text{ の場合}$$

12本の鉛筆が一つのケースに入っています。
ケースを二等分すれば、 $\frac{1}{2}$ ケースには、鉛筆6本が入っています。ケースを三等分すれば、 $\frac{1}{3}$ ケースには、鉛筆4本が入っています。 $\frac{2}{3}$ ケースは、 $\frac{1}{3}$ ケースの二倍ですので、 $\frac{2}{3}$ ケースには、鉛筆が8本入ります。

ですので、 $(\frac{1}{2} + \frac{2}{3})$ ケースには、鉛筆が、 $6 + 8 = 14$ 本入ることになります。

これは、ケース一つ(12本)とケース6分の1(2本)分になります。

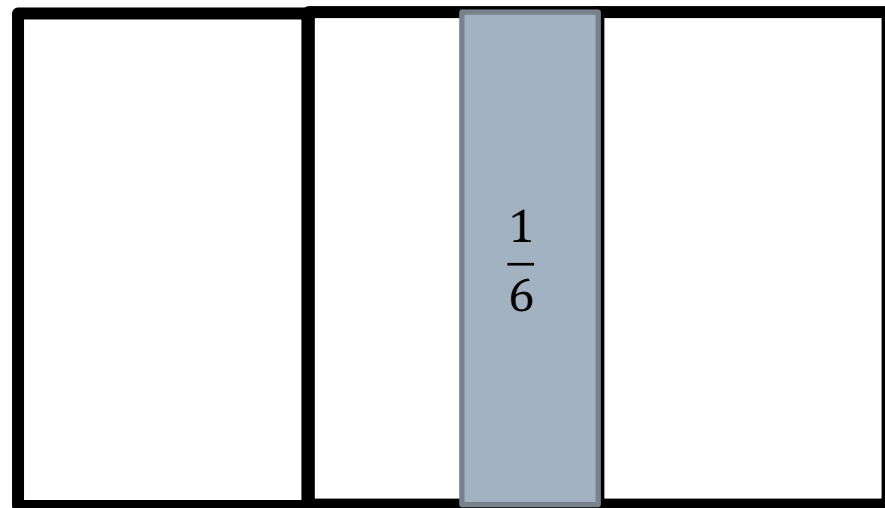
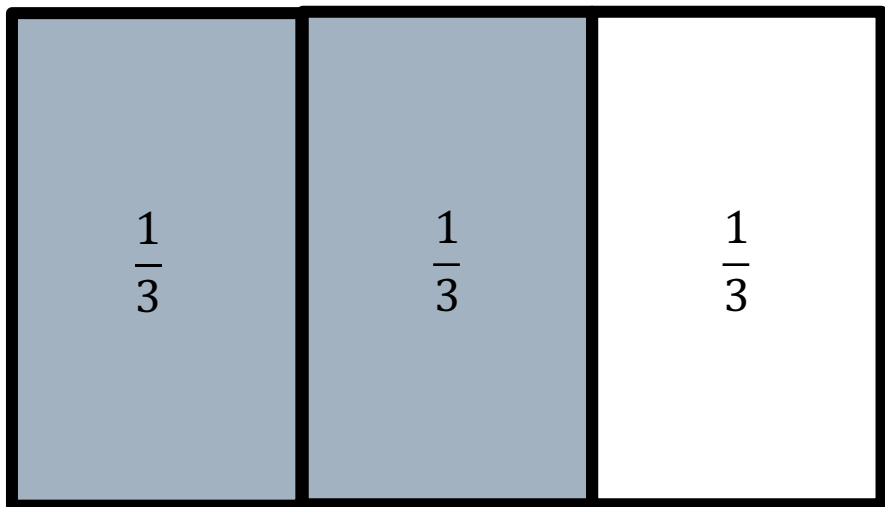
$$\frac{2}{3} - \frac{1}{2} = \frac{1}{6} \text{ の場合}$$

12本の鉛筆が一つのケースに入っています。
ケースを二等分すれば、 $\frac{1}{2}$ ケースには、鉛筆6本が入っています。ケースを三等分すれば、 $\frac{1}{3}$ ケースには、鉛筆4本が入っています。 $\frac{2}{3}$ ケースは、 $\frac{1}{3}$ ケースの二倍ですので、 $\frac{2}{3}$ ケースには、鉛筆が8本入ります。

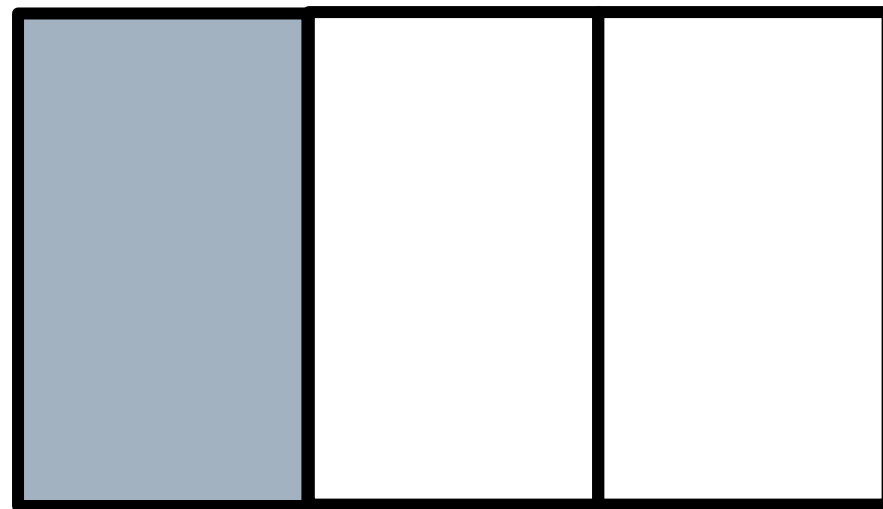
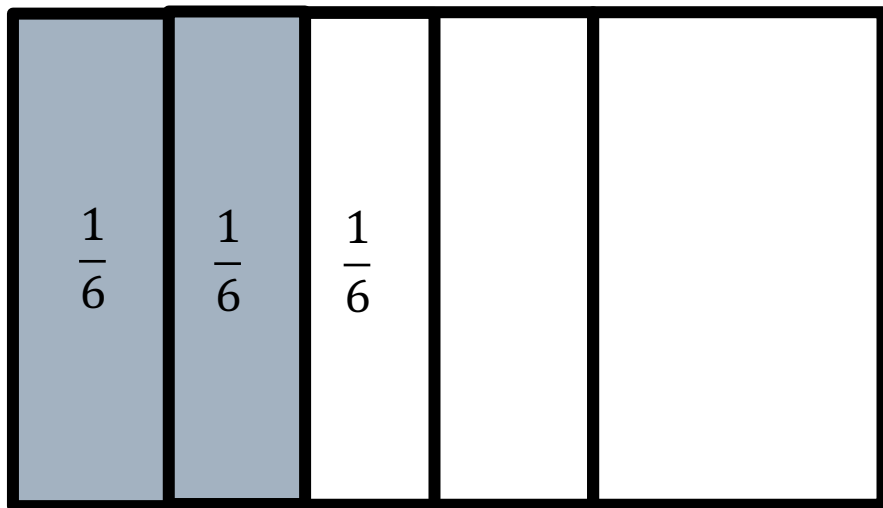
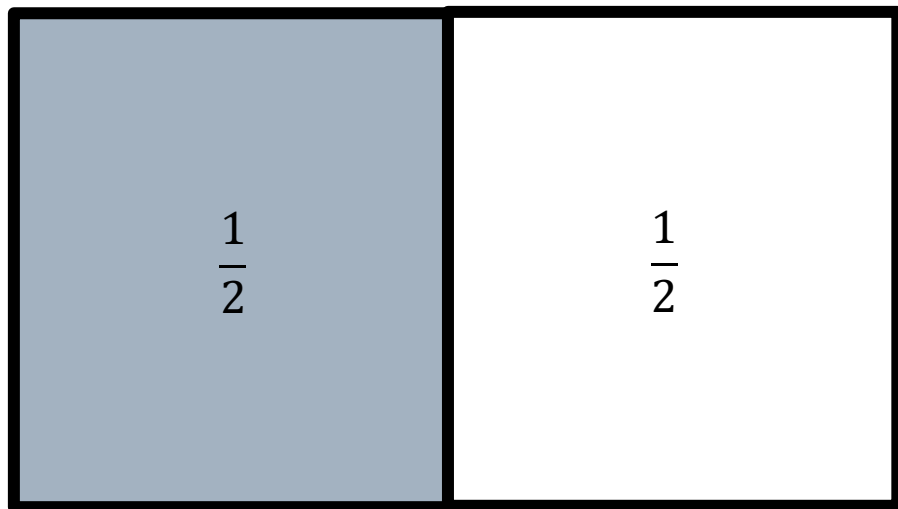
ですので、 $(\frac{2}{3} - \frac{1}{2})$ ケースには、鉛筆が、 $8 - 6 = 2$ 本入ることになります。

これは、ケース一つ(12本)の $\frac{1}{6}$ になります。

$$\frac{2}{3} - \frac{1}{2} = \frac{1}{6} \text{ の場合}$$



$$\frac{1}{2} \times \frac{2}{3} = \frac{2}{6} = \frac{1}{3} \text{ の場合}$$



$$\frac{1}{2} \div \frac{2}{3} = \frac{3}{4} \text{ の場合}$$

鉛筆が12本入ったケースと、3人で1組のグループがあったとする。この時、ケースの1/2をグループの2/3で分けたとしよう。

これは、鉛筆6本を二人で分けたことになるので、グループの一人は3本(3/12)の鉛筆を取ったことになる。この取り方で、一つのグループあたり何本の鉛筆が分配されたかを考えれば、グループは、3人で1組なので、9本の鉛筆をえたことになる。これは、ケースで考えれば、 $9/12=3/4$ ケースになる。

1グループあたりのケースを求めると、

$$\frac{1}{2} \div \frac{2}{3} = \frac{1}{2 \times 2} \times 3 = \frac{1 \times 3}{2 \times 2} = \frac{3}{4}$$

4

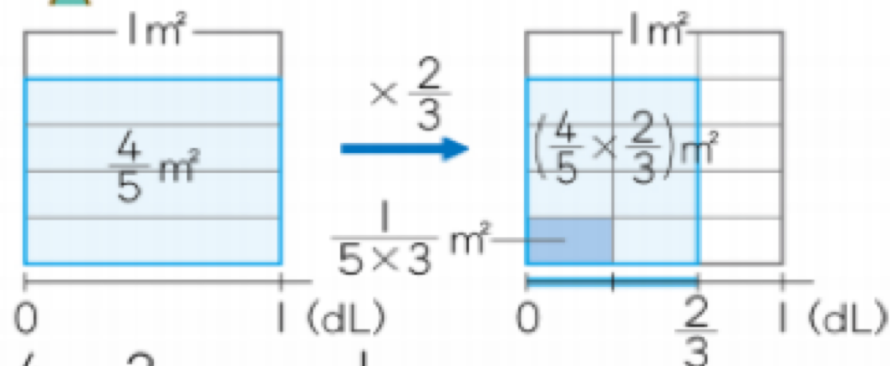
1 dLで $\frac{4}{5}\text{m}^2$ ぬれるペンキがあります。
 $\frac{2}{3}\text{dL}$ のペンキでは何 m^2 ぬれますか。



6年 p.280



ひなた



$\frac{4}{5} \times \frac{2}{3}$ は、 $\frac{1}{5 \times 3}$ の が

(4×2)個分だから、

$$\frac{4}{5} \times \frac{2}{3} = \frac{\boxed{} \times \boxed{}}{\boxed{} \times \boxed{}}$$

$$= \boxed{} \text{ m}^2$$



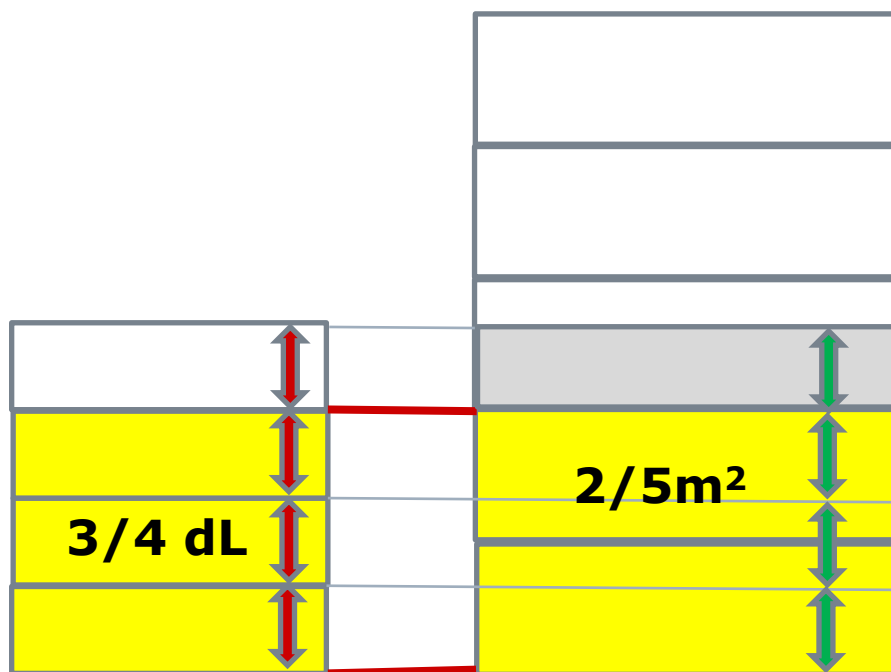
だいち

$$\begin{aligned} \frac{4}{5} \times \frac{2}{3} &= \frac{4}{5} \times \left(\frac{1}{3} \times 2 \right) \\ &= \left(\frac{4}{5} \times \frac{1}{3} \right) \times 2 \\ &= \frac{4}{5 \times 3} \times 2 \end{aligned}$$

$$= \frac{\boxed{} \times \boxed{}}{\boxed{} \times \boxed{}}$$

$$= \boxed{} \text{ m}^2$$

3/4dLのペンキで2/5m² ぬるとき, 1dLでぬれる面積を求め
る式を考える



もしも、2dLのペンキで、3m²
ぬれるのなら、1dLでぬれる面積
は、3 ÷ 2の1.5m² である。

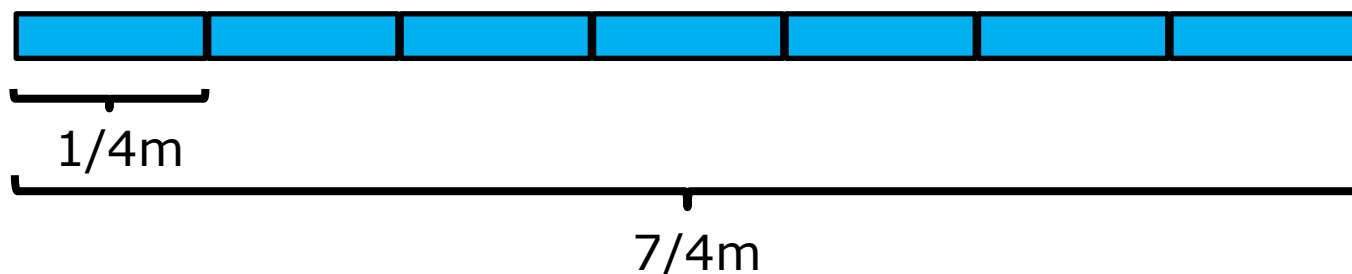
だから、この場合の式は、

$$\frac{2}{5} \times \frac{3}{4} \text{となる。}$$

1/4dLでぬれる面積は、2/5m²の
1/3なので、2/5×1/3=2/15m²
よって3/4dLでぬれる面積は、
2/15×4=8/15

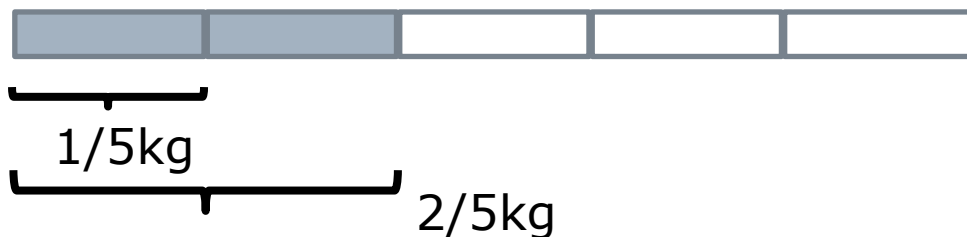
$$\begin{aligned} \frac{2}{5} \times \frac{3}{4} &= \frac{2}{5} \times \frac{1}{3} \times 4 \\ &= \frac{2}{5} \times \frac{4}{3} \\ &= \frac{8}{15} \end{aligned}$$

7/4mの重さが2/5kgのホースについて、
ホース1mの重さ、及びホース1kgの長さを求めよ



7/4mの重さが2/5kg だから、1/4mの重さは、 $2/5 \div 7 = 2/35kg$
 よって、1m=4/4mの重さは、 $2/35 \times 4 = 8/35kg$ 。

$$\begin{aligned} & \frac{2}{5} \div \frac{7}{4} \\ &= \frac{2}{5} \times \frac{4}{7} \\ &= \frac{8}{35} \end{aligned}$$



2/5kgの長さが7/4m だから、1/5kgの長さは、 $7/4 \div 2 = 7/8$
 よって、1kg=5/5kgの長さは、 $7/8 \times 5 = 35/8m$ 。

$$\begin{aligned} & \frac{7}{4} \div \frac{2}{5} \\ &= \frac{7}{4} \times \frac{5}{2} \\ &= \frac{35}{8} \end{aligned}$$

「ルール」の世界と「たとえ」の世界

「ルール」で理解する vs. 「たとえ」で理解する

2+3が5になるのを、「りんご二つと、みかん三つで、全部で五つ」というように、具体的なものをあげて、「たとえ」で説明することができる。

数学のルールは、基本的には現実の具体的な関係から抽象されたものであるので、理解しやすい具体的関係から、ルールを導入・説明するのは有効である。

「ルール」は、抽象的なものだが、「たとえ」は、ルールの「意味」を理解させる。

「ルール」で理解する vs. 「たとえ」で理解する

先のりんごとみかんの例では、実は、「全部で」という言葉が、足し算の説明になっている。「たとえ」では、具体的なものにたいする理解とともに、「言葉」の理解が、前提になることがある。

「たとえ」の世界は、具体的で現実的である。その世界は、我々の直観と「言葉」によって理解可能な「意味」の世界によって支えられている。

ただ、それは、複数の意味を持つかもしれないし、ひとつの「たとえ」で全ての「ルール」を説明できないという意味では個別的である。

また「たとえ」にはいろいろのタイプがある。言葉だけでなく、図解すること、また「たとえ」による説明の一種である。言葉による「たとえ」と図形による「たとえ」では、抽象度が異なる。

「ルール」の世界と「たとえ」の世界

数学の計算では「ルール」を「アルゴリズム」という。その「ルール」に従って、きちんと計算すると、余計なことは考えなくても正しい計算結果が保証されるという限りでは、「ルール」の世界は、他の世界とは独立しているように見える。

ただ、「ルール」は、計算をするという目的のためには万能かもしれない。ただ、計算「ルール」というものは、具体的なようで我々の頭の中にしかない抽象的なものであって、それ自体では意味を持たないようにも見える。

数学は、基本的には、それ自身の原理に基づいた自律的な「ルール」の世界である。(実際には、それは単純化しすぎである。)

小学生は何を学んでいるのか？



なかよし

1. 10までのかず
2. なんばんめ
3. いくつと いくつ
4. あわせていくつ ふえるといくつ
5. のこりはいくつ ちがいはいくつ
6. しらべよう
7. 10より おおきいかず
8. なんじ なんじはん
9. 3つのかずの けいさん

10. ながさくらべと ひろさくらべ
11. たしざん
12. かさくらべ
13. ひきざん
14. いろいろな かたち
15. 30より 大きい かず
16. なんじ なんぷん
17. ずをつかって かんがえよう
18. かたち づくり



1. せいのりのかた
2. 時刻と時間
3. 2けたのたし算
4. 2けたのひき算
5. 長さのたんい
6. 100より大きい数
7. かさのたんい
8. たし算のひっ算
9. ひき算のひっ算

10. 三角形と 四角形
11. かけ算
12. かけ算九九づくり
13. 長いものの 長さの たんい
14. たし算と ひき算の かんけい
15. 1000より 大きい 数
16. はこの 形
17. かけ算の きまり
18. 分数



1. 3けたや4けたのたし算とひき算
2. かけ算
3. 時こくと時間
4. かけ算の筆算
5. ぼうグラフと表
6. わり算
7. 分数
8. 円と球
9. いろいろなわり算

10. 長さ
11. 小数
12. 重さの単位
13. 10000より大きい数
14. 2けたの数をかける計算
15. 二等辺三角形と正三角形
16. □を使った式
そろばん



1. 大きな数
2. 折れ線グラフ
3. 角度
4. わり算の筆算
5. がい数
6. 垂直と平行と四角形
7. 小数
8. 2けたでわる計算

9. 式と計算
10. 面積
11. 整理のしかた
12. 変わり方
そろばん
13. 小数と整数のかけ算・わり算
14. 分数
15. 直方体と立方体



1. 整数と小数

2. 2つの量の変わり方

3. 小数のかけ算

4. 体積

5. 小数のわり算

6. 図形の角の大きさ

7. 合同な図形

8. 整数の性質

9. 分数のたし算とひき算

10. 平均

11. 単位量あたりの大きさ

12. 正多角形と円

13. 割合

14. 帯グラフと円グラフ

15. 分数と小数, 整数

16. 四角形と三角形の面積

17. 分数と整数のかけ算・わり算

18. 角柱と円柱

19. 数量の関係の調べ方



1. 対称な図形

2. 文字を使った式

3. 分数のかけ算

4. 分数のわり算

5. 円の面積

6. 速さ

7. 角柱と円柱の体積

8. 場合の数

9. 比

10. 拡大図と縮図

11. 比例と反比例

12. およその形と面積

13. 資料の調べ方

14. 量の単位

6年間のまとめ

わくわく算数ミュージアム

数学の世界へ

なぜ算数が重要か？
言語能力と数学的能力の違いから考える

母語の習得と算数の習得

我々は、中学校で英語を習うように、小学校で日本語を習うわけではない。日本語に関して小学校で学ぶのは、主要には、「文字」の読み書きである。

母語の習得は、生まれた時から始まる、その子を取り巻く言語環境との相互作用の複雑なプロセスであるが、明らかなのは、制度的な「学校」や意識的な「教育」が必要なわけではないということ。

母語は、ある意味「自然」に身につく。それは、言語習得のプログラムが、人間に生物学的・遺伝子的に組み込まれているからだと考えられている。

文字については、そうではない。

算数についても、それが、「自然」に身につくことはない。それは長い時間をかけて、子供時代から「教育・学習」されねばならない。

言語能力と数学的能力

母語を聞き話すという能力は、ある発話が文法的に正しいかを瞬時に判断し、また、すぐに文法的に正しい文を発話できるということなのだが、それは、その母語の使い手が、その言語の文法を意識的に理解しているということを意味するわけではない。

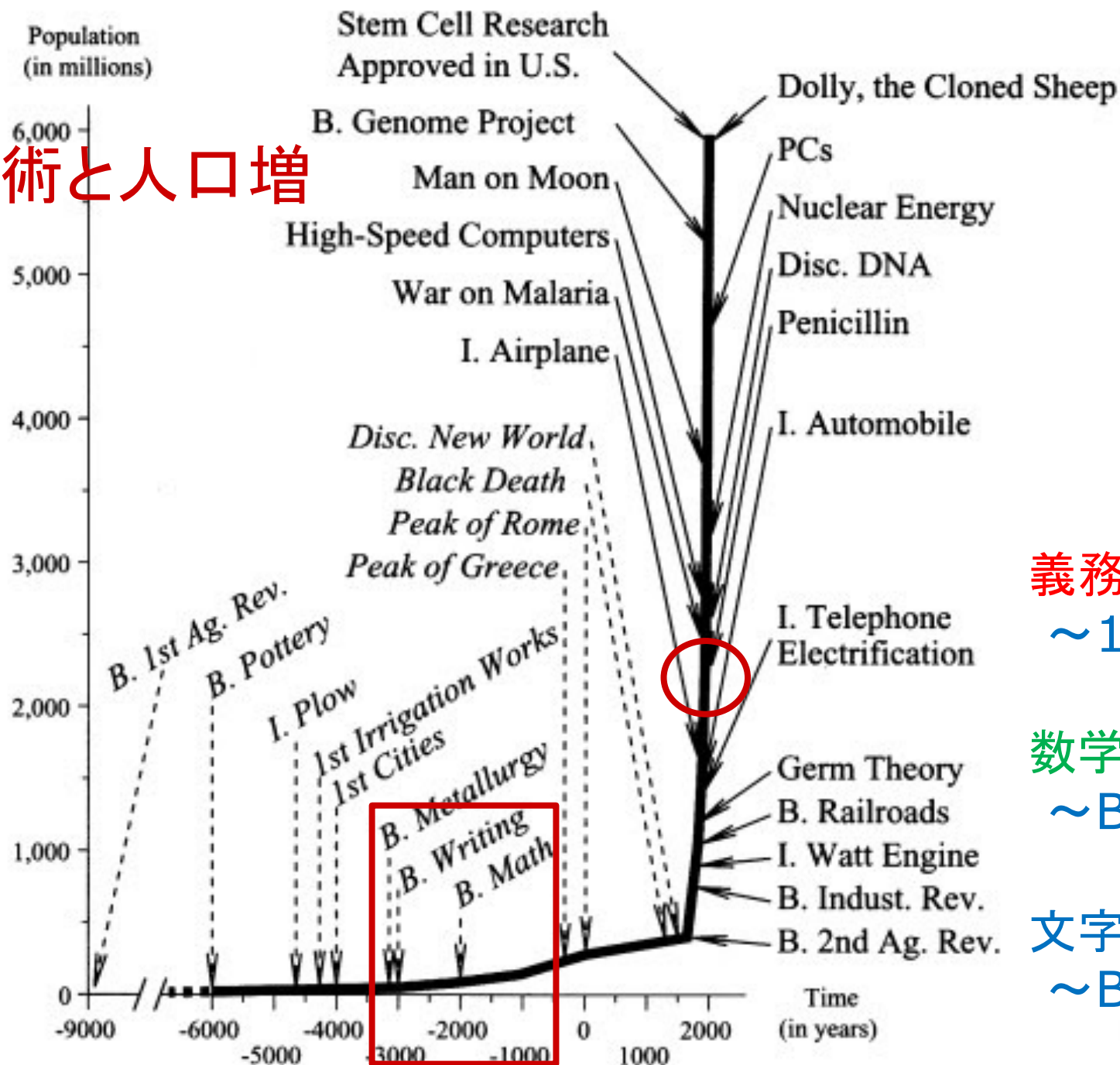
にも関わらず、「言い間違い」を除けば、我々は誰でも、常に、「正しい文」を、意識的な努力なしでも、生み出すことができる。

数学については、そうではない。

ある計算が正しいか否か、ある数学的命題が正しいか否か、反射的に答えるのは、一般的には、難しい。

言語能力での「文法的正しさ」に対する直感は、数学的能力における「数学的正しさ」に対する直感とは、そのあり方が、大きく異なっている。

技術と人口増



義務教育制度
～19世紀

数学の誕生
～BC2000年

文字の誕生
～BC3000年

(20)

No. 1	$56,606 \div 62 =$
2	$40,492 \div 53 =$
3	$276,345 \div 801 =$
4	$114,250 \div 457 =$
5	$2,838,156 \div 326 =$
6	$1,799,736 \div 984 =$

平成29年度
全日本珠算選手権大会

後編

<https://goo.gl/Wp7Cny>

Dyslexia(識字障害)にみる 話しコトバと文字の違い

NHKスペシャル 病の起源 4
読字障害 ～文字が生んだ病～

<https://goo.gl/bjLy8R>

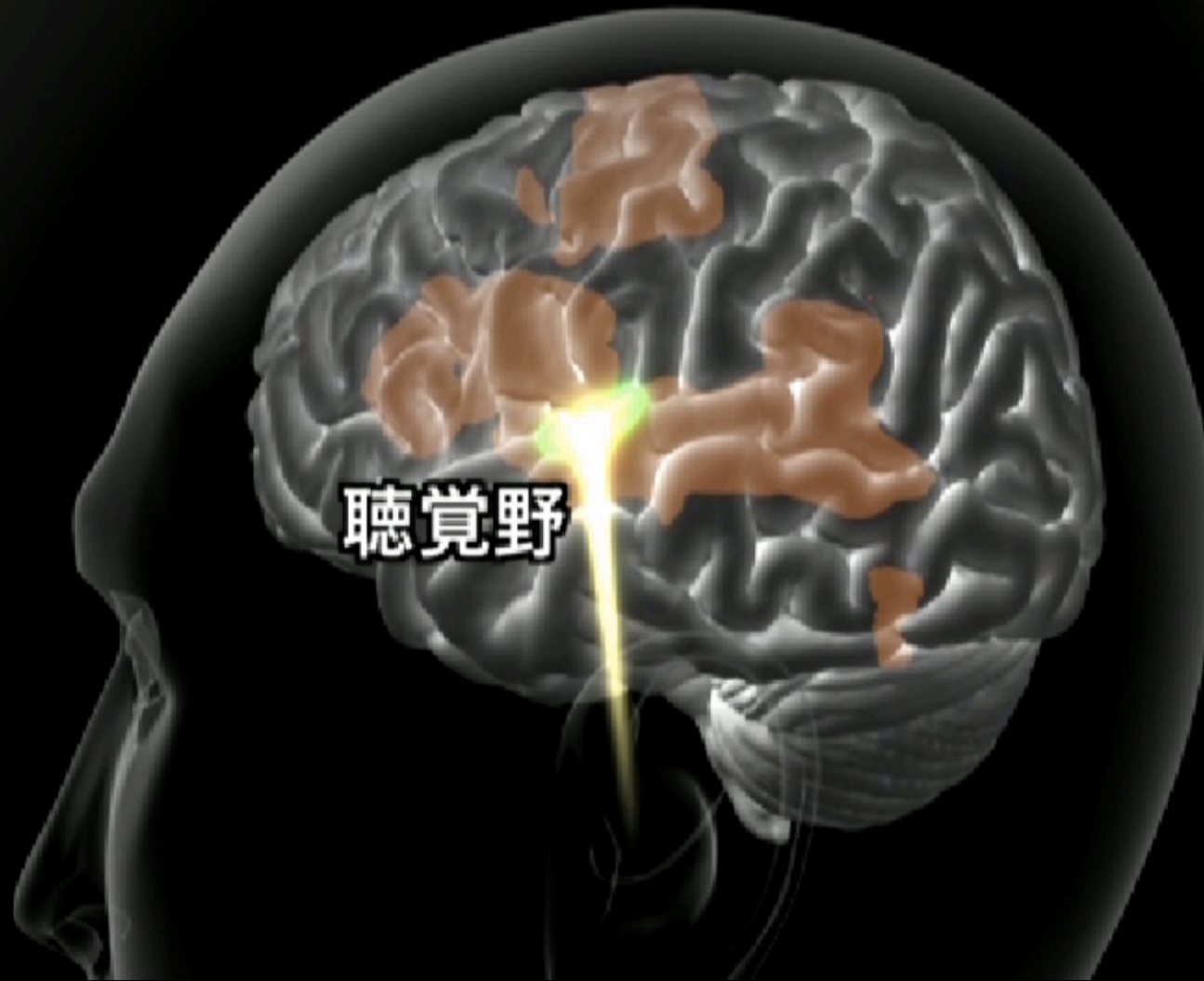
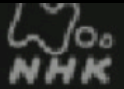
Dyslexia (識字障害)

話しコトバと文字の違い

NHKスペシャル 病の起源 4 読字障害 ～文字が生んだ病～

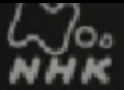
<https://goo.gl/bjLy8R>

コトバの理解



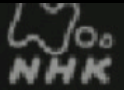
耳から入ったコトバは、まず、聴覚野に送られる。

コトバの理解



その後、ウェルニッケ野に送られ、

コトバの理解



さらにブローカ野に送られて、前頭葉で理解される。

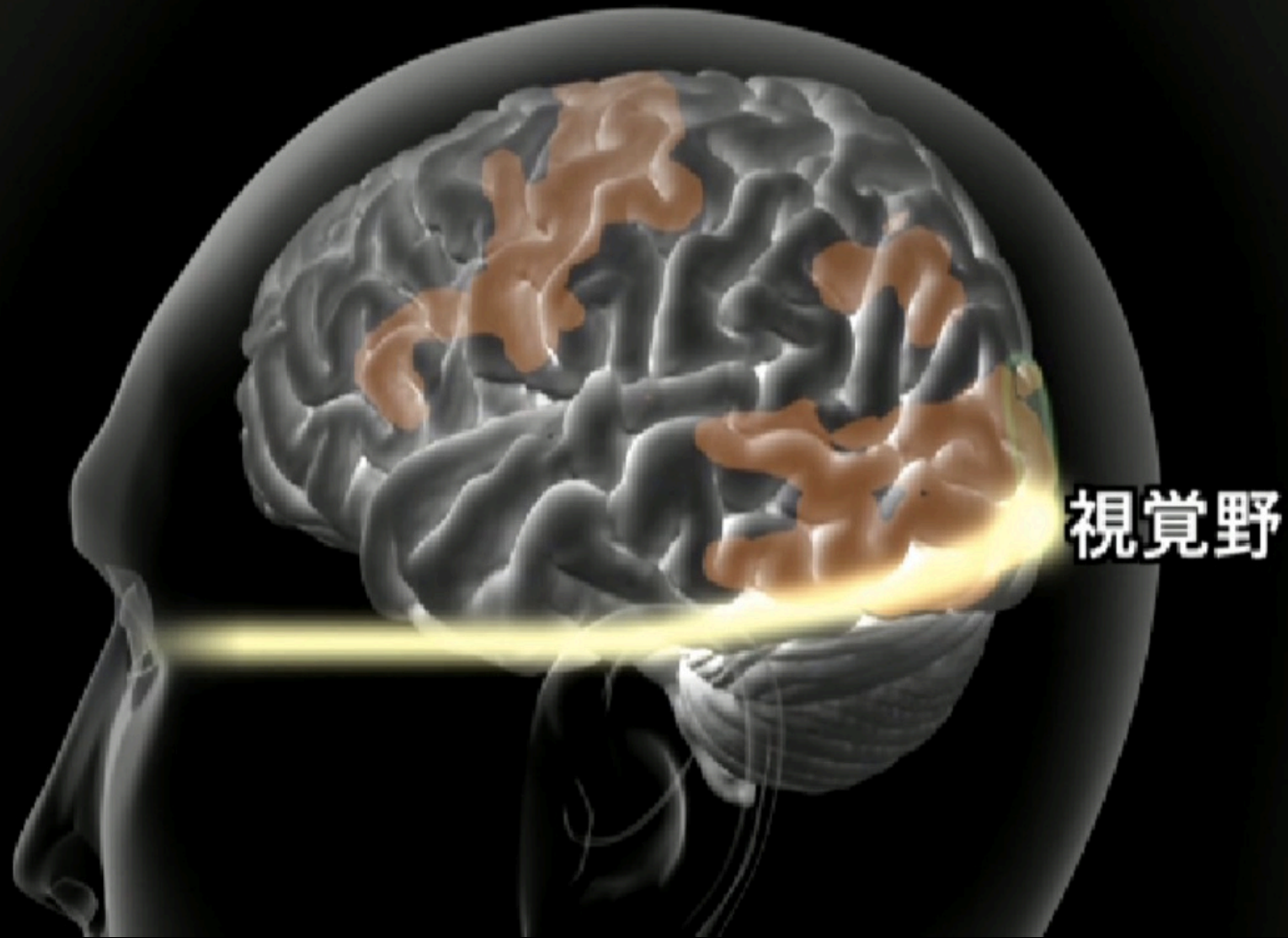
文字の理解



いるか

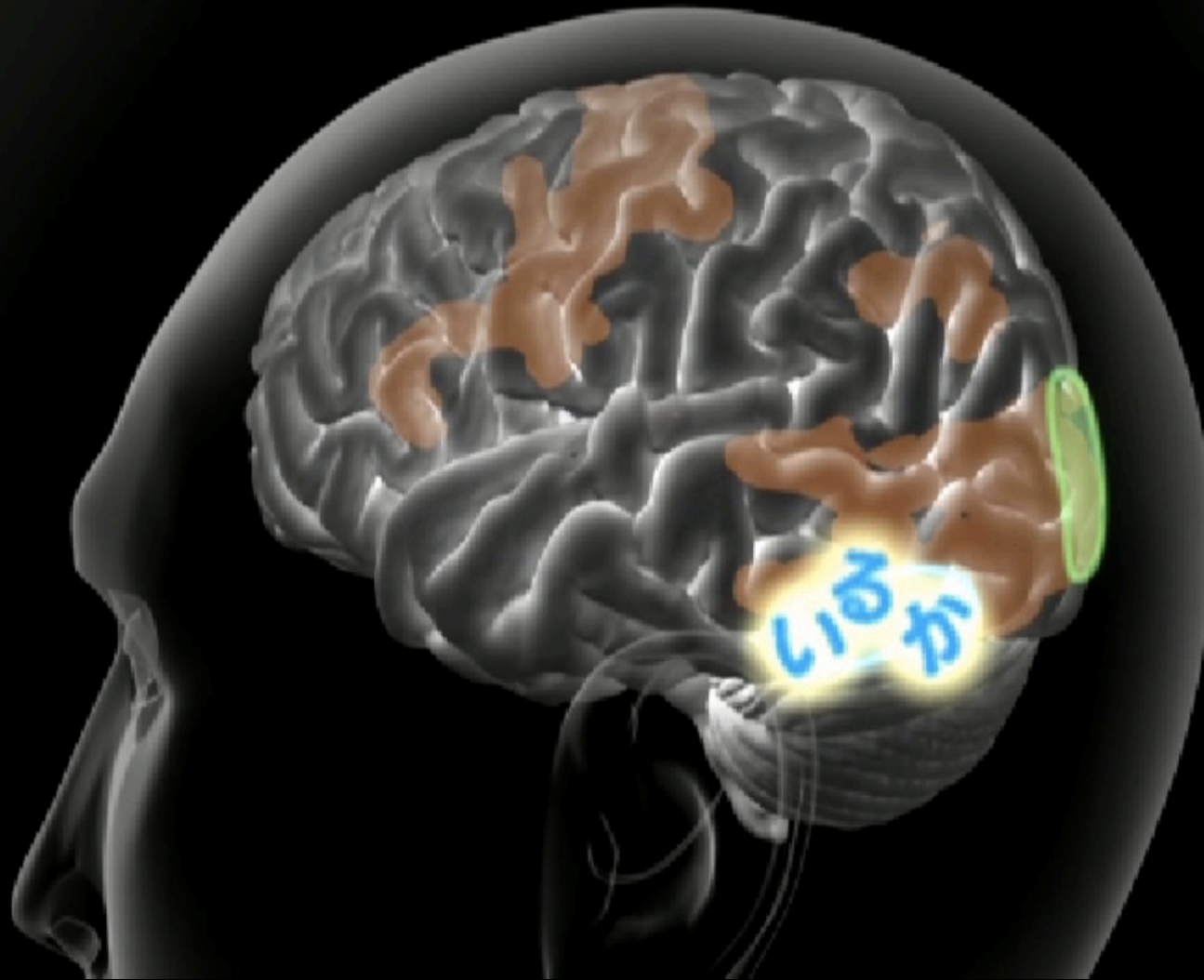
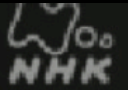
文字の情報は、目から入り、

文字の理解



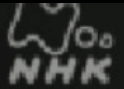
視覚野に入り、

文字の理解



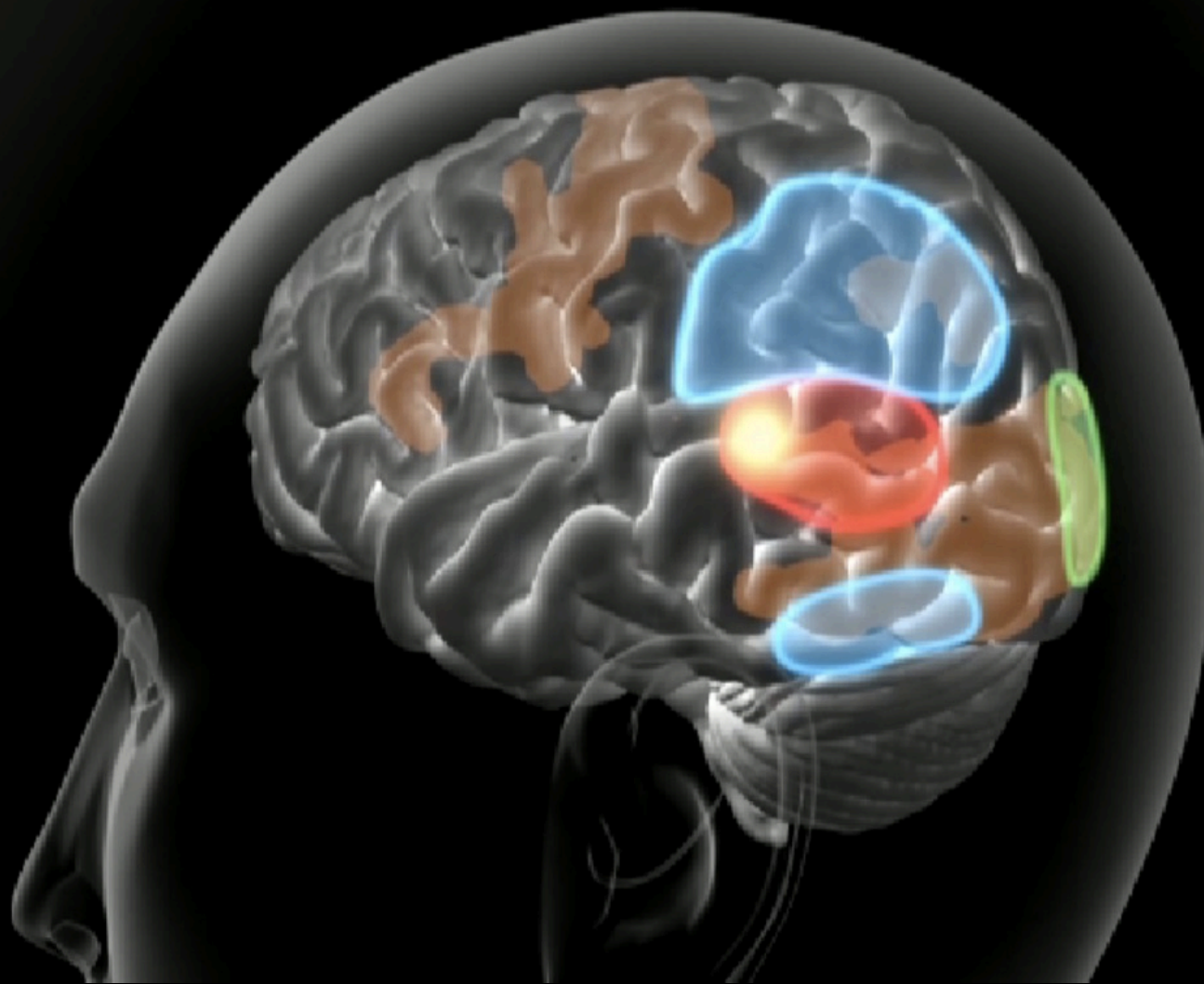
その形が識別される。

文字の理解



識別された文字情報は、39野・40野と呼ばれる領域に入り

文字の理解



そこで音声信号に変換され、ウェルニッケ野に送られる。

文字の理解



さらに、ブローカ野に送られ、

文字の理解



ブローカ野

前頭葉で理解される。

認識能力の発展と人間の知能の階層

認識能力の発展と人間の知能の階層

数学的・科学的な認識



文字による知識の集積

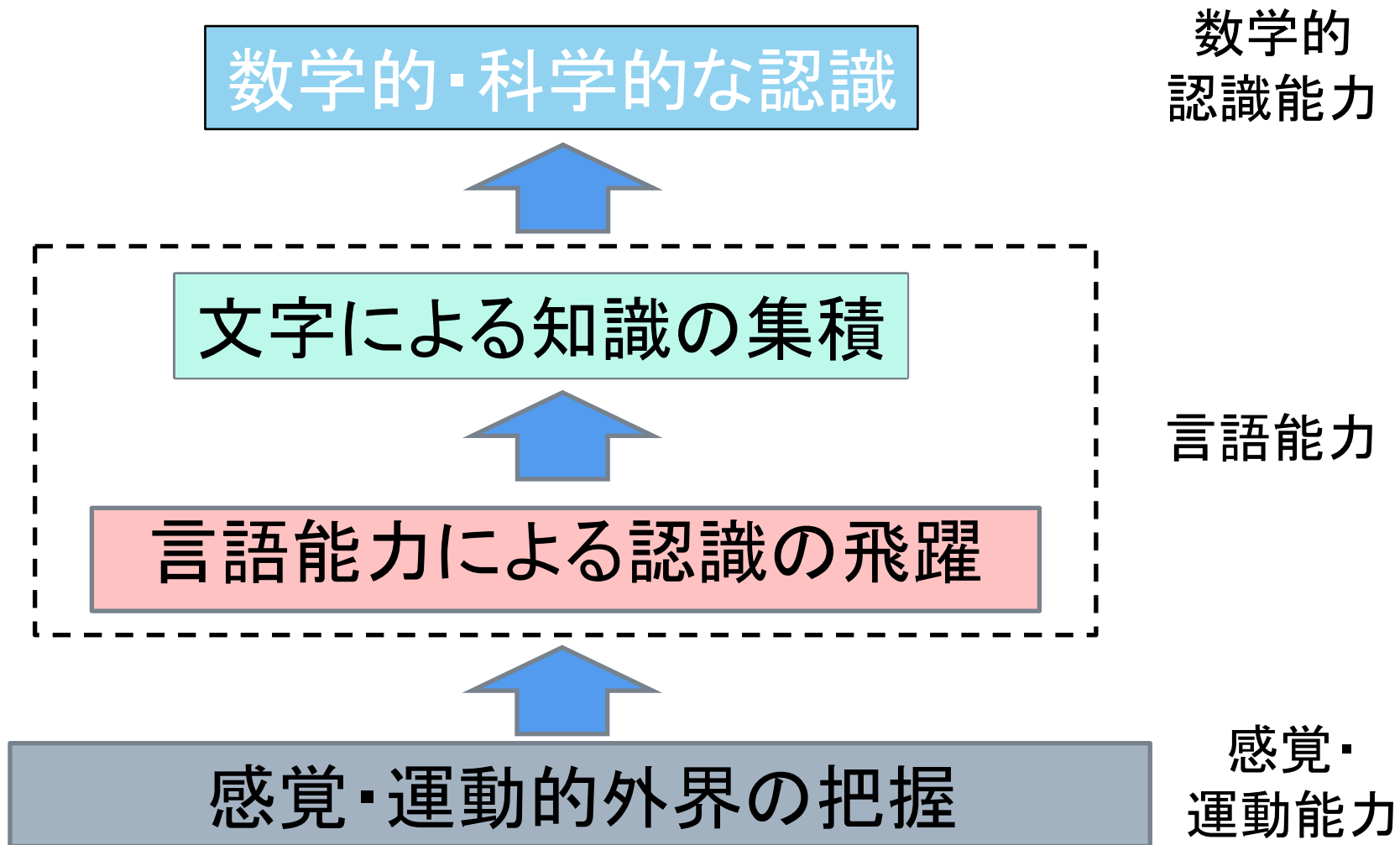


言語能力による認識の飛躍

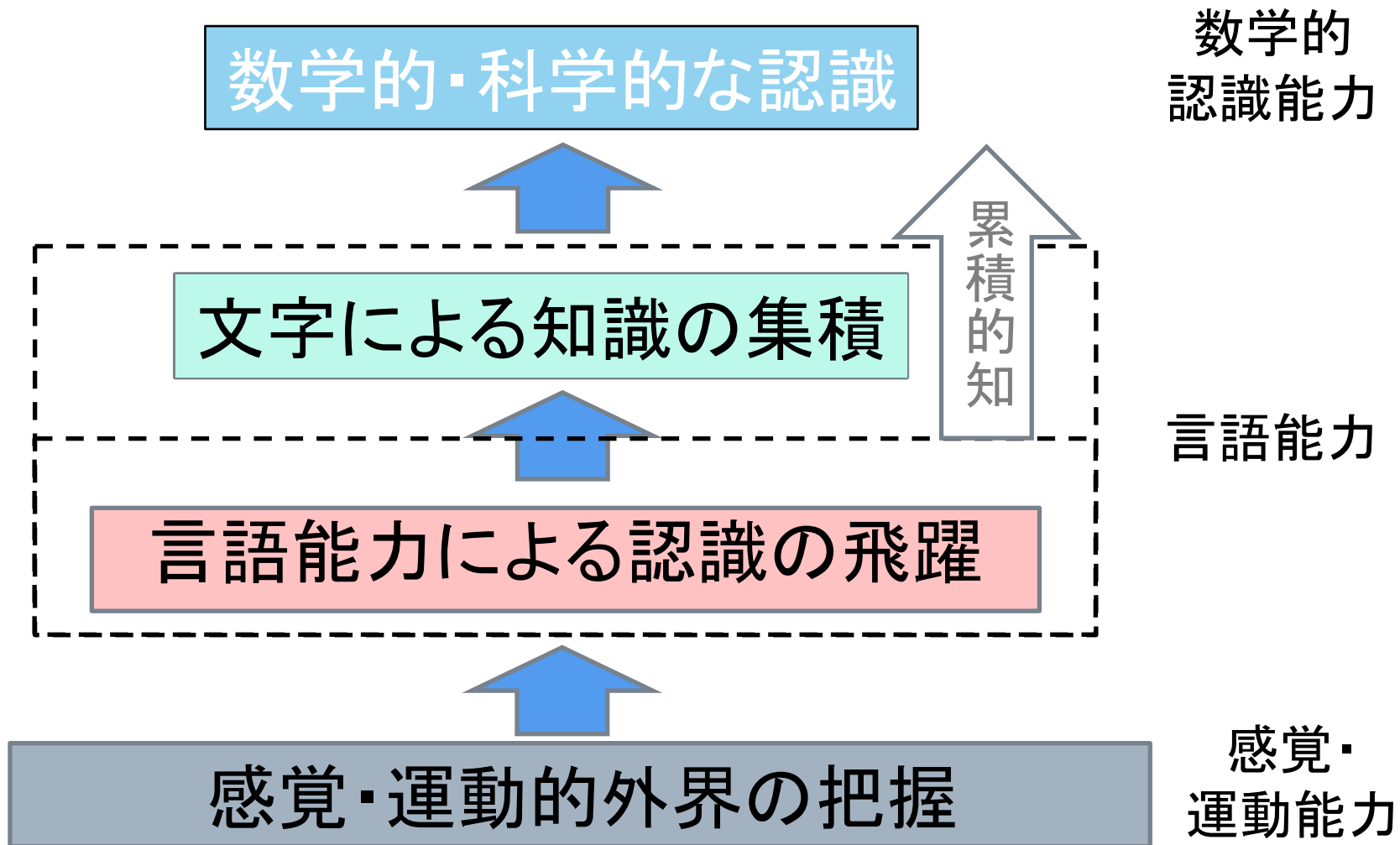


感覚・運動的外界の把握

認識能力の発展と人間の知能の階層

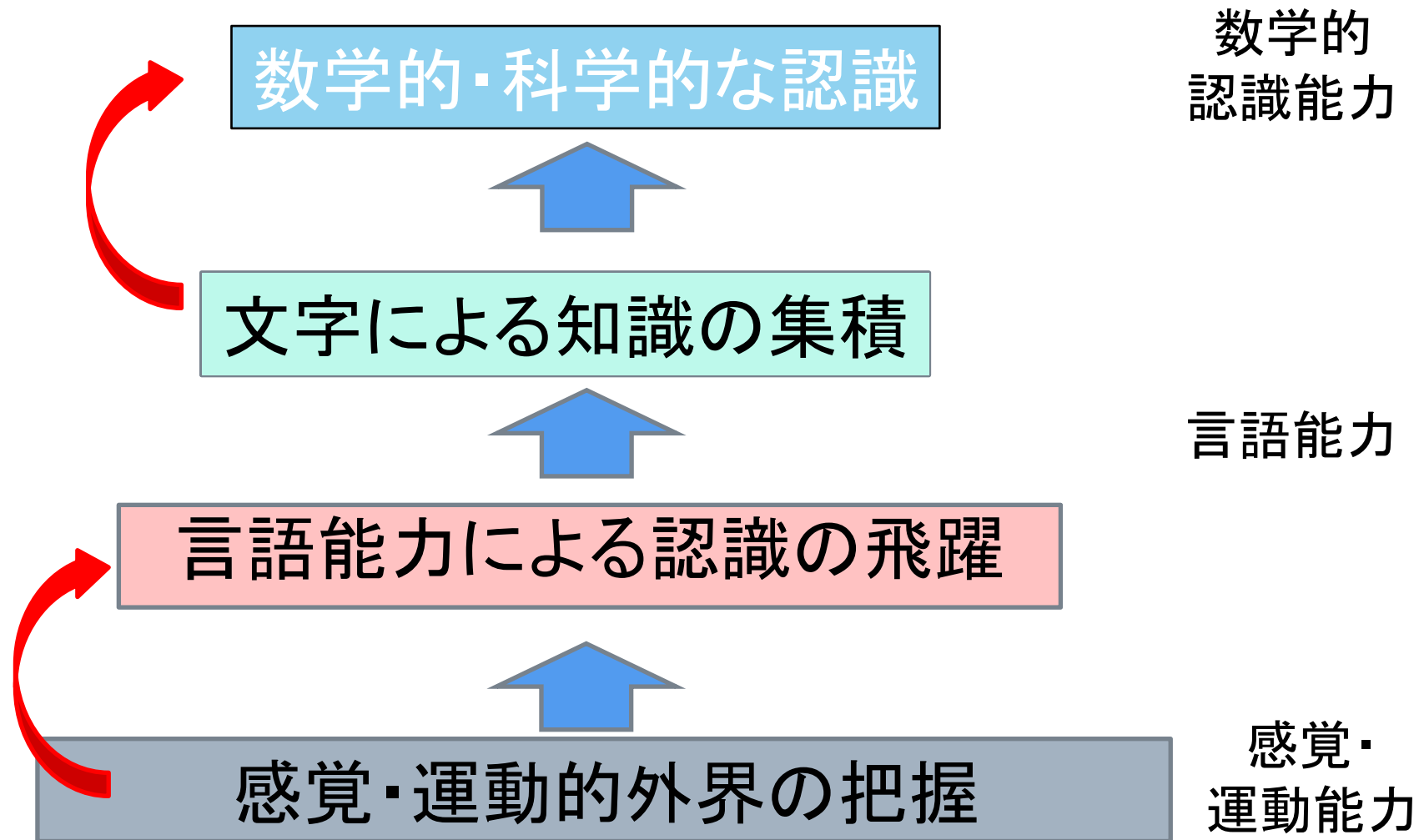


認識能力の発展と人間の知能の階層

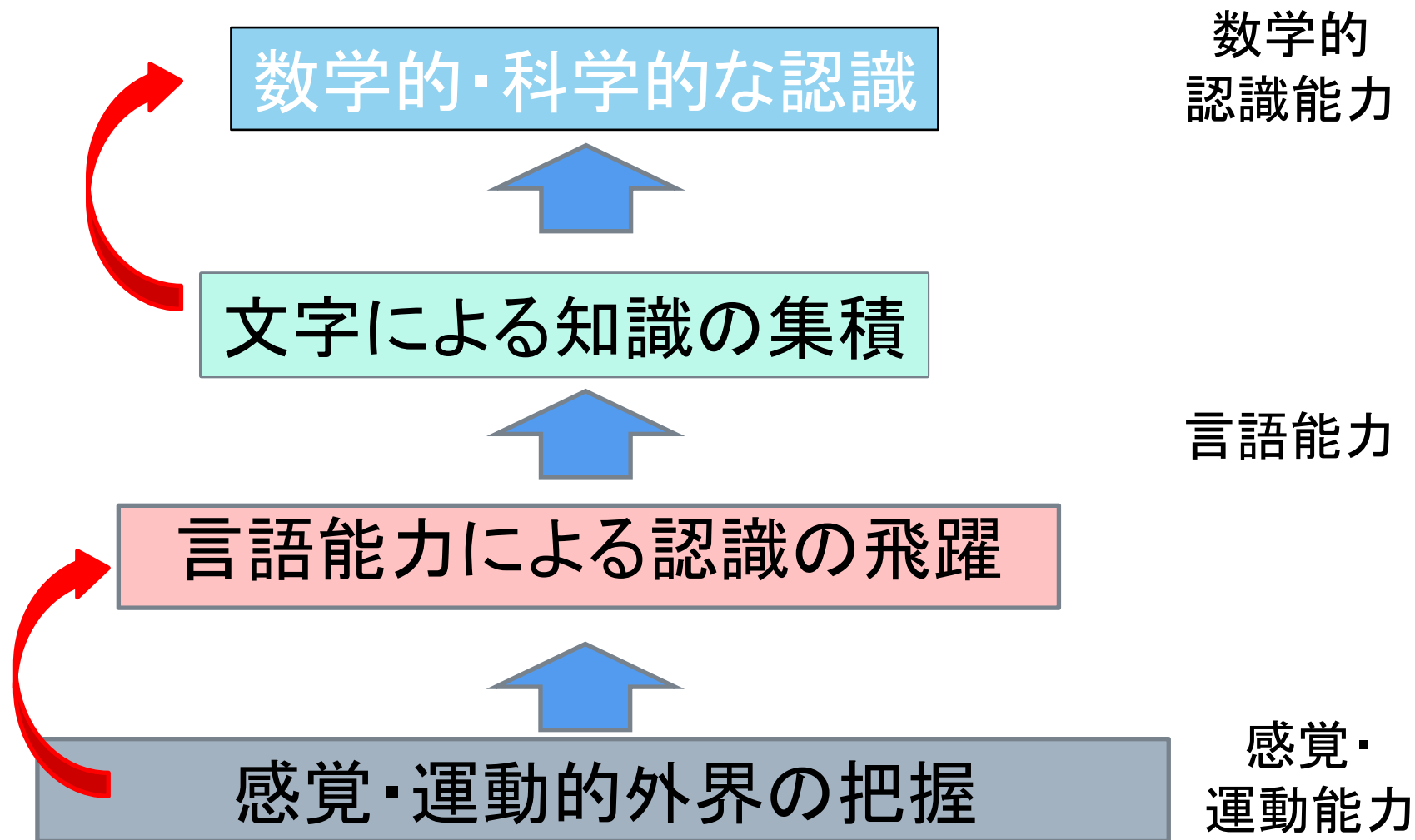


認識能力の発展と人間の知能の階層

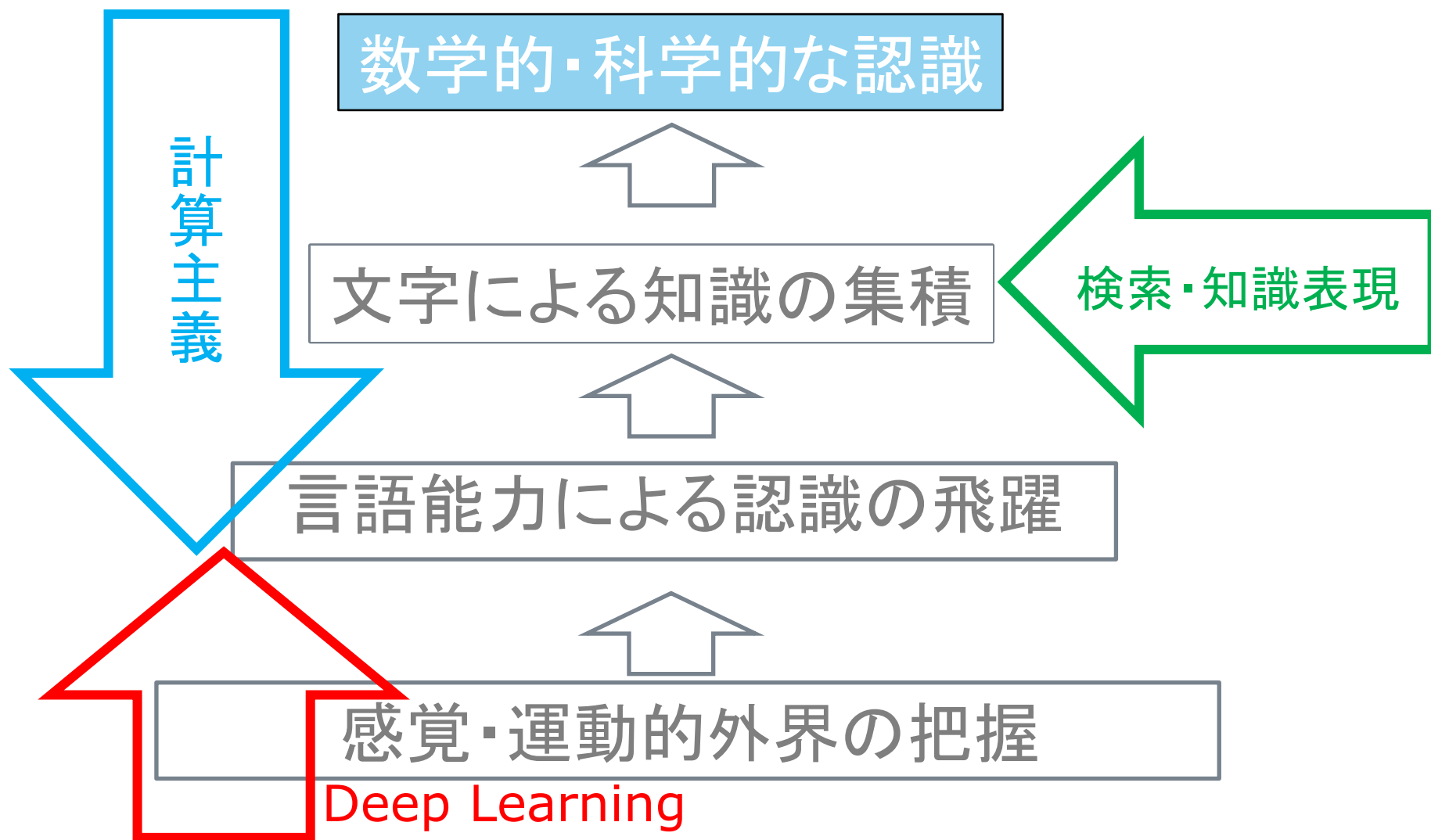
大きな飛躍は二つある



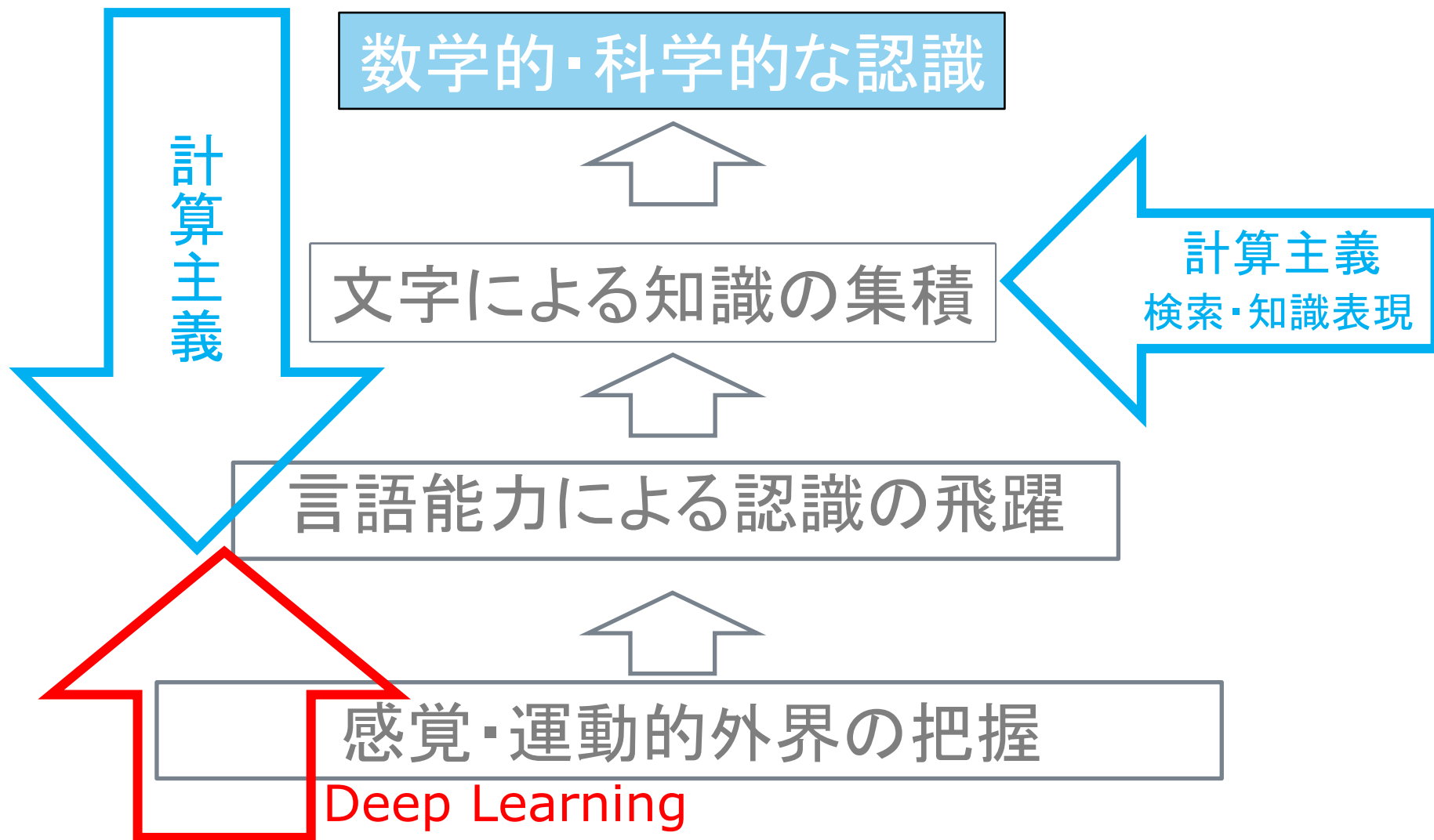
そして、それは現在の人工知能技術が
攻めあぐねている課題である



複数のアプローチが必要



計算主義の重要性は高まる



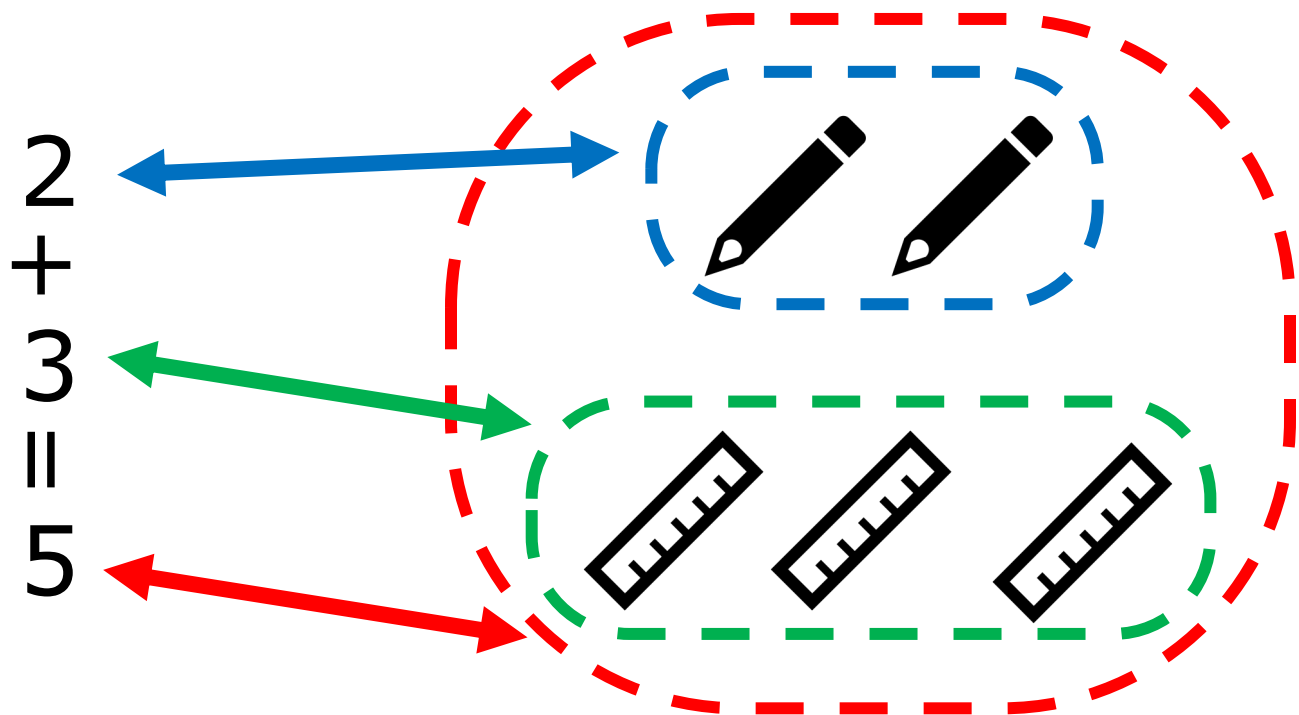
理論とモデル

「ルール」と「たとえ」の数理

素朴な対応理論

「ルール」と「たとえ」の対応関係

先に見た「ルールの世界」と「たとえの世界」の二つの世界は、お互いに関連している。「ルールの世界」は「たとえの世界」に、具体的対応物を持ち、「たとえの世界」は「ルールの世界」に抽象的な対応物を持つ。

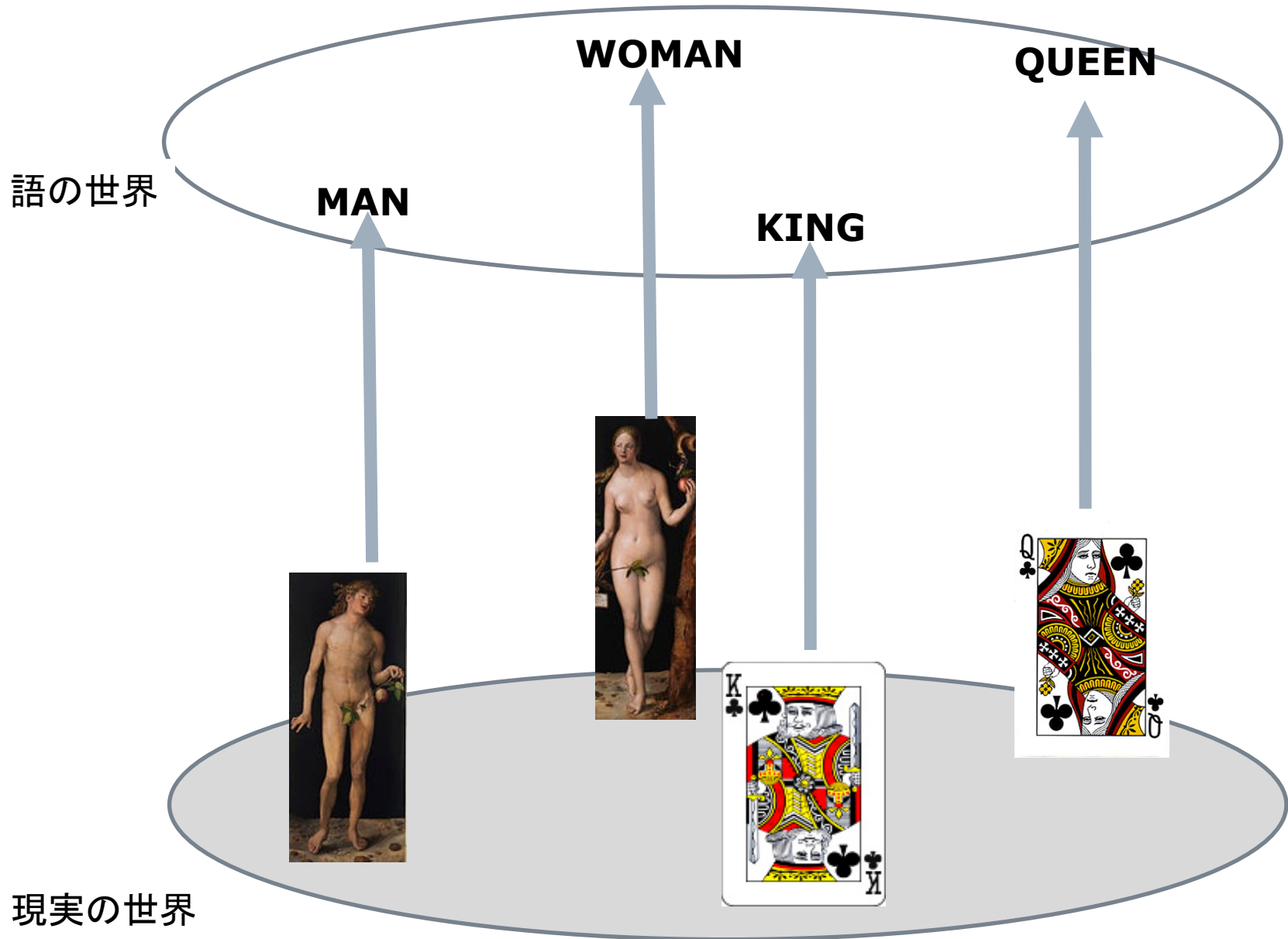


二つの「世界」の対応関係が、「意味」を与える

二つの世界に対応関係があり、一方の世界がより具体的で、もう一方の世界がより抽象的な場合、この対応関係は、抽象的な世界に、具体的な「意味」を与える。あるいは、この対応関係は、抽象的なものに具体的な「解釈」を与える。

以下では、「語の意味」の素朴なモデルとして、現実に存在する「事物」と、その「名前」の「対応関係」を、考えてみよう。

「語の意味」の素朴なモデル



意味：ものに「名前」をつけること

語の世界

MAN

WOMAN

QUEEN

KING

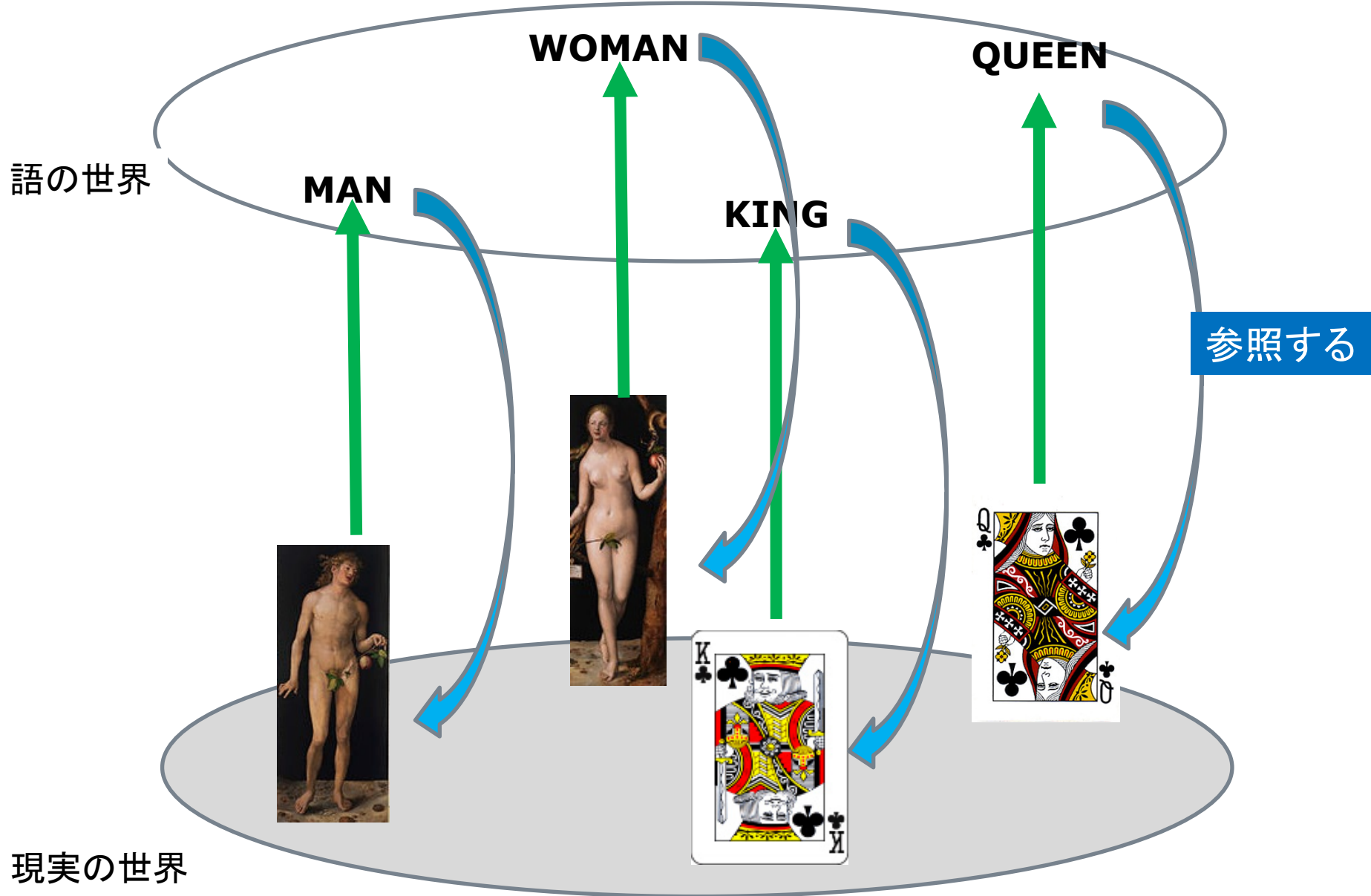
対応づける
名付ける

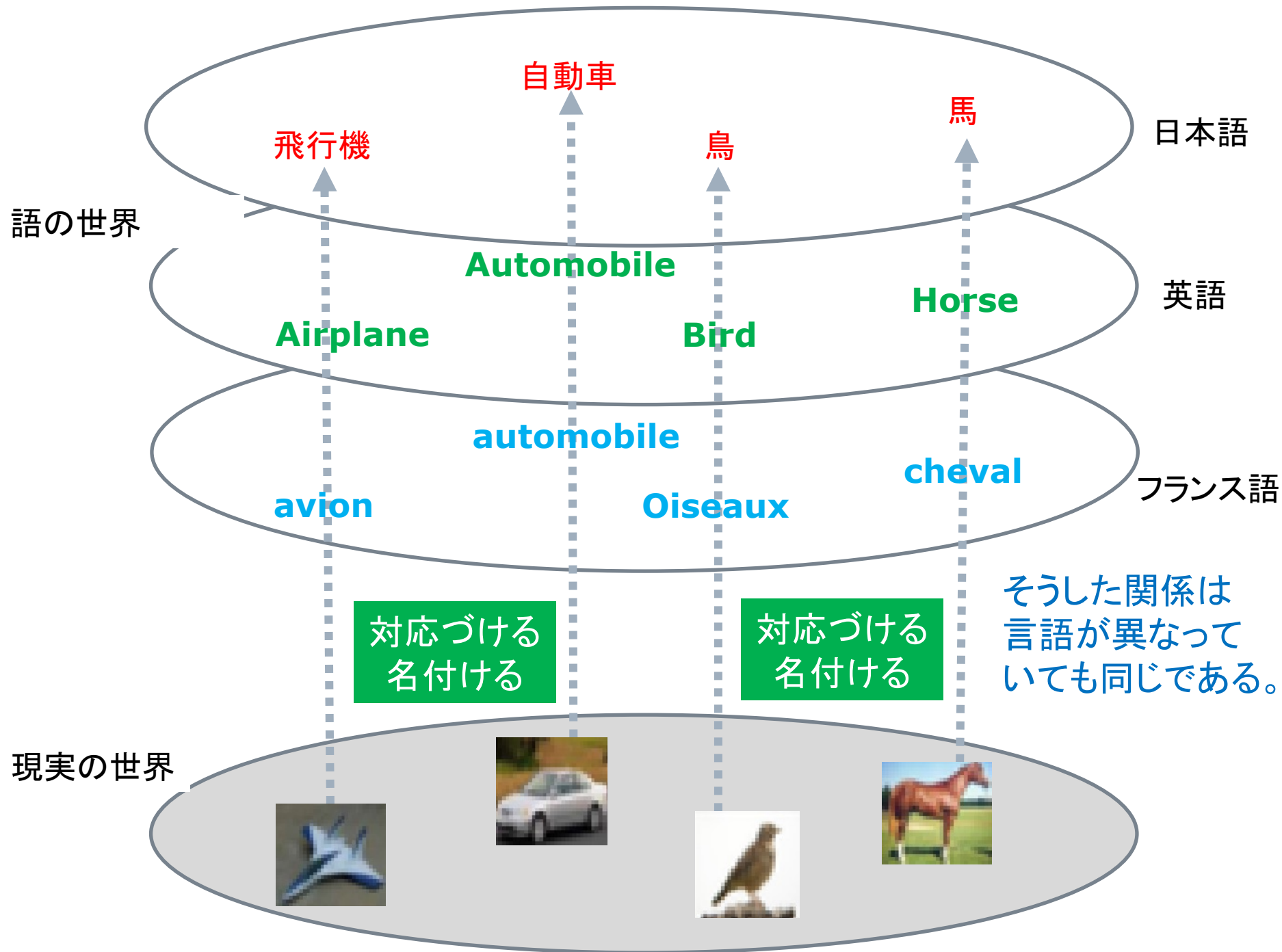
対応づける
名付ける

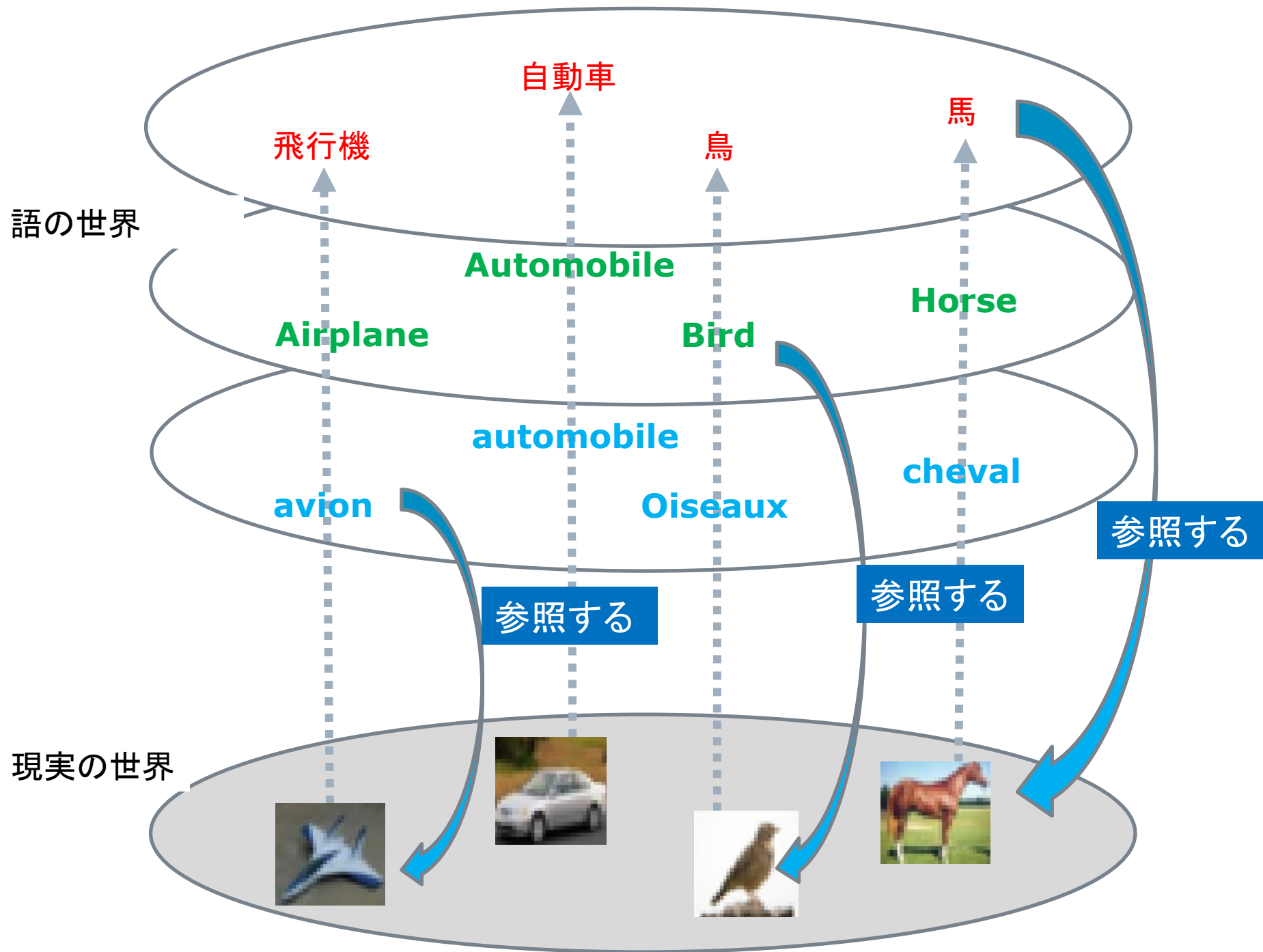


現実の世界

語は、ものを「参照」する







素朴な「対応理論」の弱点

- ただ、こうした素朴な「対応理論」には、弱点もある。
「現実の世界での『馬』」という対象を考えてみよう。こうした対象の「実在性」については、は、いろいろな疑問がうまれる。「現実中存在するのは、個々の具体的なウマだけである。」「いや、『馬』の意味の実在性には、疑問の余地はない。」等々。
- その上、語は(文になればなおさらなのだが)、「物」に付けられた意味を持つ名前だけではない。抽象的な概念(「自由」「平等」「意味」...)や、物ではない「出来事」(「第二次世界大戦」「爆発」...)も名前を持つのだ。具体的な心理的な状態(「悲しみ」「意図」...)も。これらを先の図にマッピングするのは、難しい。
- 「状態の変化」の一群は、「動詞」で表現される。

現実の世界

「同一性」

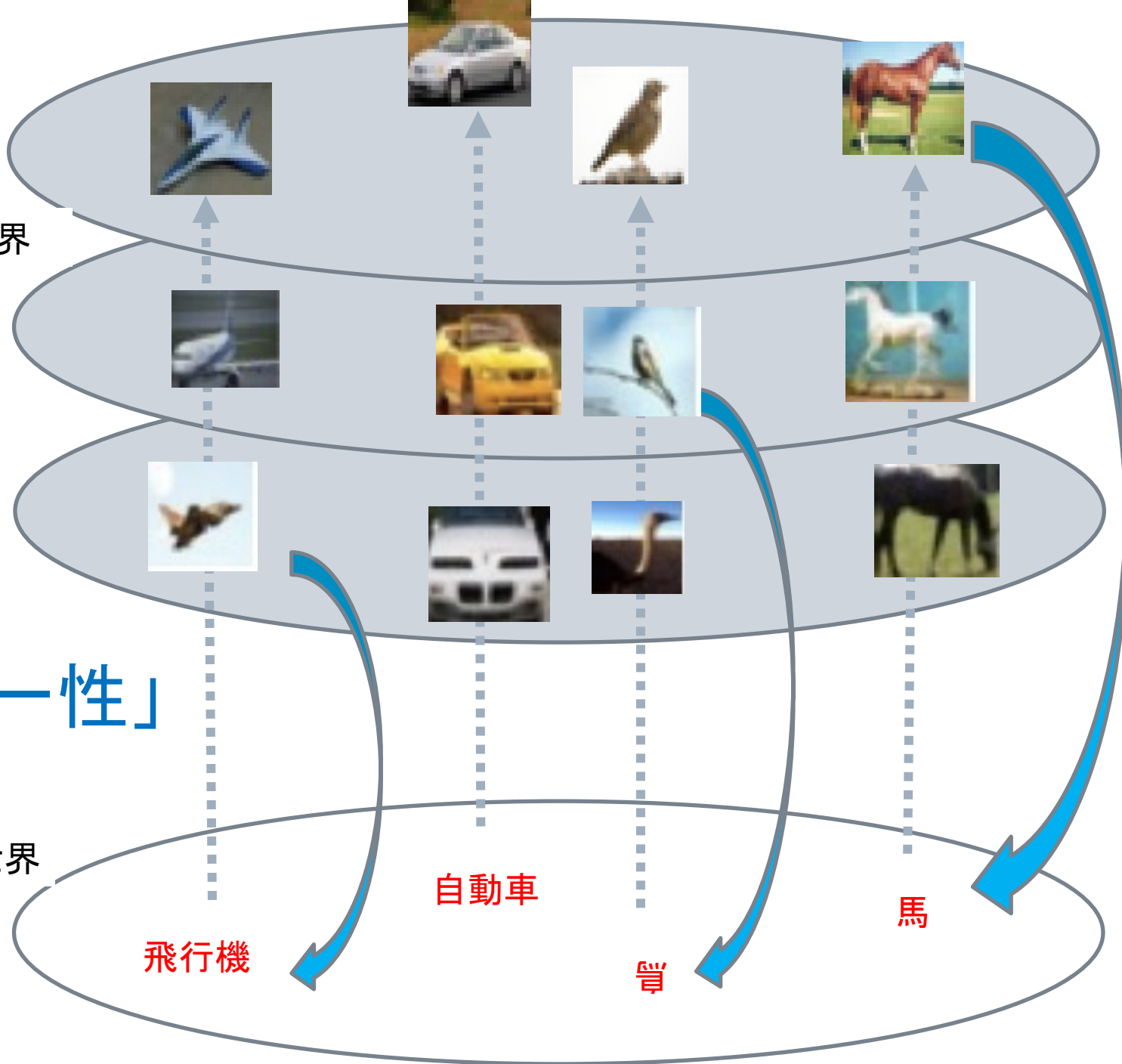
意味の世界

飛行機

自動車

翼

馬



理論とモデル

対応の形式化

「理論」と「モデル」の形式的定義

ここでは、「ルールの世界」と「たとえの世界」の数学的形式化を紹介したい。

ある世界で、基本的な公理と一階の述語論理の演繹規則が与えられているとしよう。この世界を「理論 Theory」と呼ぶことにしよう。この「理論」の世界で、ある命題 ϕ が成り立つのは、この理論の中で、命題 ϕ が証明される場合に限る。

今度は、もう一つ別の世界を考えよう。この世界では、ある命題 ϕ が妥当する（「真」である）ことが、何らかの方法で定義されているとする。こうした世界を「モデル Model」と呼ぼう。

ある理論 T とあるモデル M について、すべての命題 ϕ について、次の関係が成り立つとする。

T において ϕ は証明可能 $\rightarrow M$ において ϕ は真である

この時、 T はモデル M を持つ。あるいは、 M は T のモデルである。という。

ゲーデルの完全性定理

理論とモデルの関係を、「すべてのモデルで真になる命題は、一階の述語論理で証明可能である」という形で初めて述べたのは、ゲーデルである。これをゲーデルの「完全性定理」という。

この定理から、「ある理論が無矛盾ならそれはモデルを持つ」ことが導かれる。<https://goo.gl/5HLMsK>

モデルの存在定理では、次の「レーベンハイム＝スコーレムの定理」が有名である。

「理論が非可算濃度のモデルを持つなら、それは可算モデルも持つ。」

「計算できないもの」の発見

-- ゲーデルの不完全性定理

思考実験

ある人が、「私の言うことはすべて嘘である」と言ったとしよう。
この人のこの言葉は嘘だろうか？ 嘘ではないだろうか？

ゲーデルの不完全性定理

“On Formally Undecidable Propositions of Principia Mathematica and Related Systems”

Kurt Gödel 英訳 by Bernard Meltzer

<https://goo.gl/PMmqYr>

ヒルベルト・プログラムと「有限の立場」

1920年、ヒルベルトは、全ての数学に確実な基礎を提供するという壮大な計画を提出する。その主要内容は、

- 数学は、形式言語で、明確なルールに従って記述されること。
- 数学的に正しい命題は、全て、この形式的システムの中で証明されること。
- この数学の形式化が矛盾を含まないことが証明されること。
その無矛盾性の証明は、有限の数学的対象に対する「有限の立場」からの証明であるべきこと。
- 「実在的な対象」について、「理想的な対象」(例えば、「非可算集合」のような)を用いて行われた証明は、それらを用いなくても証明されるべきこと。
- 全ての数学的命題の真または偽を決定する方法が存在すること。

1930年 ゲーデルの不完全性定理（1）

ゲーデルの不完全性定理とは、次のような定理である。

「ある形式的体系Sを作ったとしよう。その体系では、初等数論を定義でき、矛盾がないとする。その時、Sに属する命題Gで、Sの中では証明も反証もできないような命題Gが存在する。」

例えば、次のような命題Gを考えてみよう。

G:「Gは証明できない。」

この命題Gが証明できたとする。それは、Gは証明できないことを証明したことになる。

この命題Gの否定が証明できたとする。それはGは証明できること、すなわち、Gは証明できないことを証明したことになる。

嘘つきのパラドックス

これは、つぎのような「嘘つきのパラドックス」と、同じような構造である。

あるクレタ人が語った。「クレタ人の言うことは全て嘘である。」

このクレタ人の言うことが嘘ではなく正しいとしよう。それは、「クレタ人の言うことは全て嘘である。」というのが正しいということなので、このクレタ人の言葉も嘘になる。

このクレタ人の言うことが嘘だとしよう。それは、彼が嘘をついていないことを意味する。

チューリング・マシンの「停止問題」

関数halts(f)は、ある関数fの計算が停止するなら真を、そうでないなら偽を返すものとする。

この時、次の関数gを考える。

```
def g():  
    if halts(g):  
        loop_forever()
```

halts(g)が真を返すなら、それは、関数gの計算が停止すると言うことなのだが、gの定義を見れば、その時関数gの計算は、無限ループに入って終わらない。

halts(g)が偽を返すなら、それは関数gの計算が停止しないと言うことなのだが、gの定義を見れば、それはすぐに終わる。

1930年 ゲーデルの不完全性定理 (2)

それだけだと、そんなものかと思われるかもしれないが、ゲーデルの不完全性定理が、数学の世界に強い衝撃を与えたのは、不完全性定理の次の形である。

Fを、初等数論を含む、無矛盾の形式的体系としよう。この時、この形式的体系Fの無矛盾性は、この体系Fの中では証明出来ない。

この結果は、ヒルベルト・プログラムが、実行不可能であることを意味していた。

「計算できる」ことの特徴付け
チャーチ=チューリングのテーゼ

不完全性原理は、我々の認識の限界？

ただ、この不完全性定理は、確かに驚くべきものではあるのだが、ほとんど役に立たない定理である。

というのは、言い過ぎなのだが、この定理から、我々人間の（同時に、それは人工知能にとっての）認識能力の限界を議論するのは、あまりに、荒い議論であると僕は考えている。

一つには、不完全性定理による原理的な「限界」以外にも、形式的演繹による認識には、実質的には、様々な「限界」が存在しているからだ。我々の形式的認識の「限界」を構成しているのは、不完全性定理だけではないのだ。

（あとで見るように、複雑性理論は、もっと精緻に、その「限界」を境界づけようとしている。）

「計算とはなにか？」「証明とは何か？」 より基本的な「問い」の始まり

あと一つには、ゲーデルの不完全性定理は、「証明不可能性」「形式的体系の無矛盾性」についてのショッキングな定理なのだが、いかなる計算、いかなる証明が「可能」であるかについては、あまり、多くのことを我々に教えてはくれないのだ。

歴史的に見ると、不完全性定理は、証明あるいは計算の性質についての探求の中で生まれた「最後の結論」ではないのだ。事実、その逆である。1930年代に、この分野は、不完全性定理に強烈な刺激を受けて、「証明とは何か？」「計算とは何か？」という基本的な「問い」に突き進むことで発展する。

知的な展開としては、それに少し先行した、ハイゼンベルグやシュレジンガーやディラックといった若い天才たちが大活躍した量子力学の基礎付けの時代と比肩しうる、エキサイティングな「知的な黄金時代」だったと思う。（現代は、本当に、「イノベーティブ」なのかと、ふと思う。）

様々な「計算可能性」へのアプローチと その同値性の認識

ゲーデルの結果を受けて、「計算可能性」についての探求が一斉に始まる。

- ゲーデル＝エルブランの「帰納関数論」
- チャーチの「ラムダ・カリキュラス」
- チューリング＝ポストの「チューリング・マシン」

が続々と登場する。

この時代の白眉は、「計算可能性」についての見かけは全く異なるこれらのアプローチが(もちろん、それは、ゲーデルの結果を再現できるものだ)、次々と、「同値」であることが証明されていくことだ！

チャーチ=チューリングのテーゼ

1940年代には、今日、「チャーチ=チューリングのテーゼ」と呼ばれる「計算可能性」についての認識は、確立されることになる。
(これが、「複雑性理論」の第一世代である。)

「チャーチ=チューリングのテーゼ」というのは、次のような提案である。

Every effectively calculable function (effectively decidable predicate) is general recursive

すべての実効的に計算可能な関数(実効的に決定可能な述語)は、一般帰納的である。

すべての実効的に計算可能な関数(実効的に決定可能な述語)は、チューリング・マシンで計算可能である

50年代



チャーチ=チューリング
のテーゼ

計算可能性理論

「証明可能＝計算可能」なものには限界があることは、
1950年代には、一般に認められるようになった。
ただ、それは、我々の認識の「限界」としては、非常に
荒い、かつ、抽象的で原理的な限界を与えるものでしかなかった。

チューリング・マシンで計算する

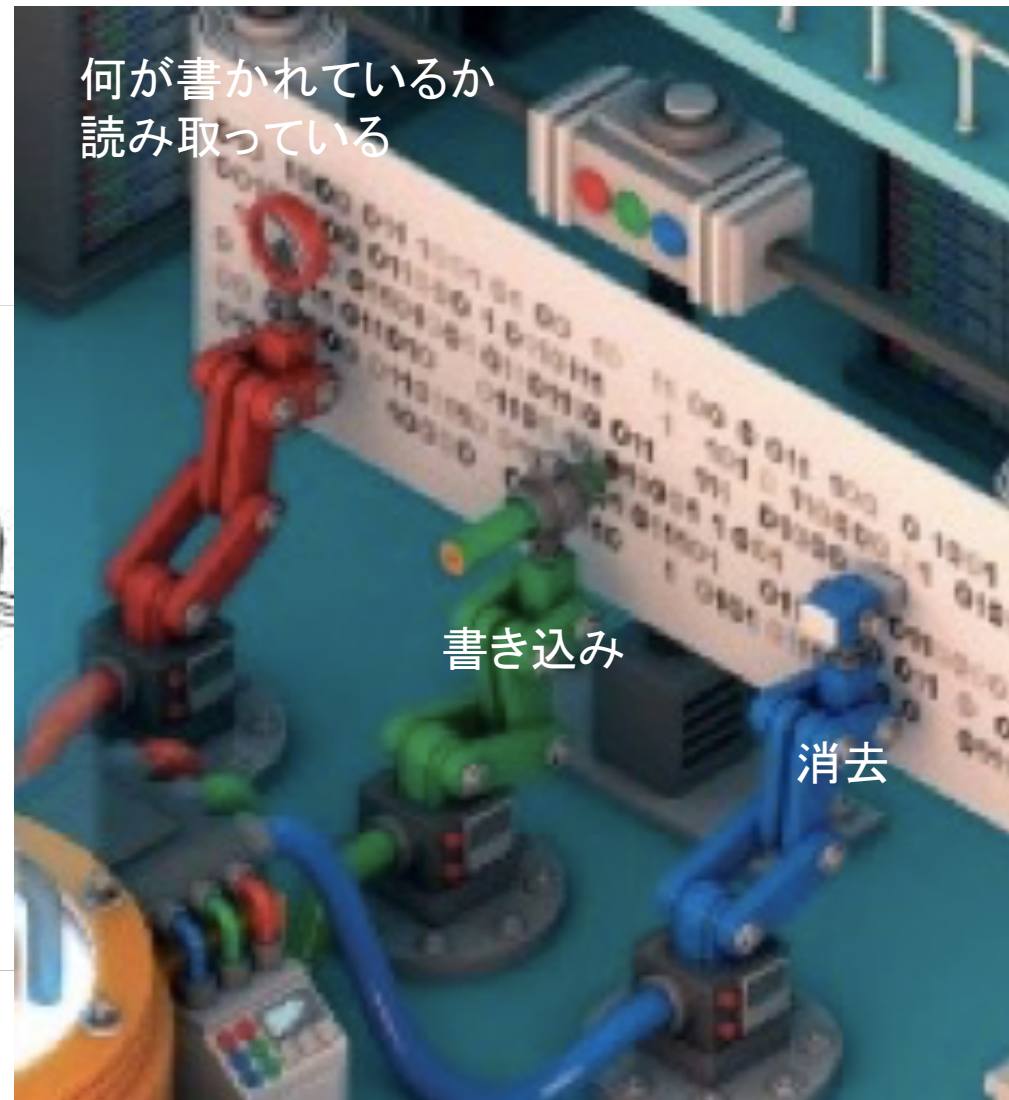
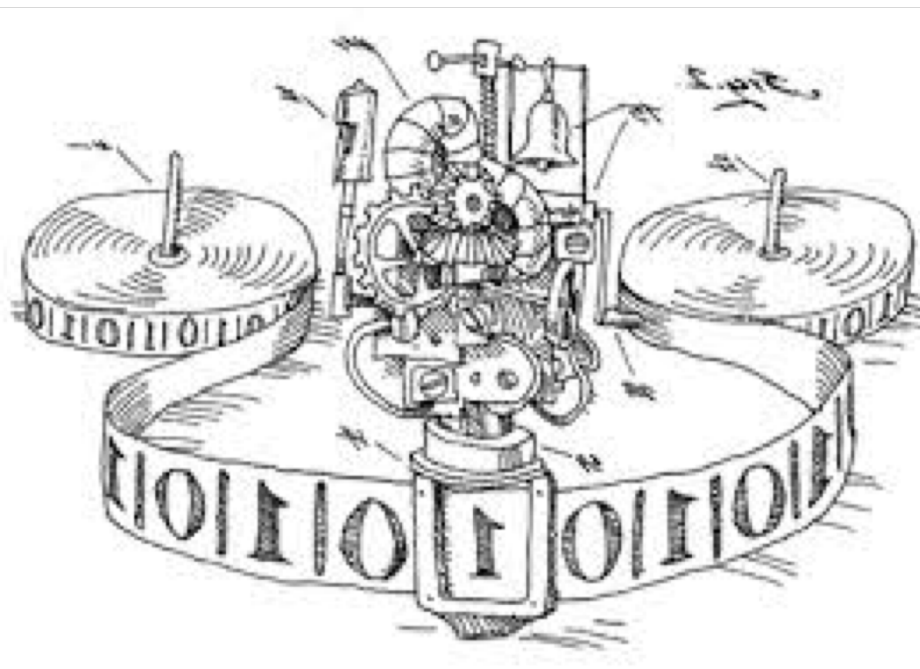
“ON COMPUTABLE NUMBERS, WITH AN
APPLICATION TO THE ENTSCHEIDUNGSPROBLEM”

Alan Turing 1936年

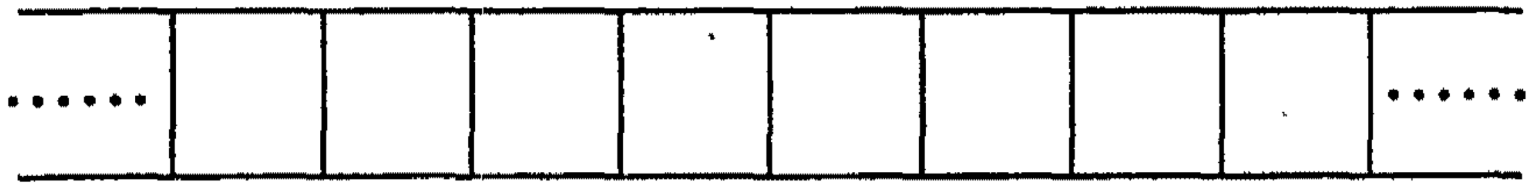
<https://goo.gl/mi4MBb>

チューリング・マシンとは何か？

チューリング・マシンのイメージ



チューリング・マシンのメカニズム



テープ (マスで区切られている)



テープ: ひとマスずつに区切られており、左右に移動する。長い。

ヘッド: テープのひとマスの記号を読み取り、マシンの「状態」に応じて、次の動作をする。

可能な動作:

1. マス目に記号を書き込む(消す+書く)。あるいは、そのままにして何も書き込まない。
2. マシンの状態を変える。あるいは状態を同じに保つ。
3. ヘッドを、右あるいは左に移動する。(テープが動く)

チューリング・マシンのプログラムの例

Turingマシンのプログラムの例を示す。この表の、例えば、最初の State Aの行で、On '0' の欄の 'B1R' は、「状態Aで、ヘッドが'0'の上にあるなら、状態をBに変えて、1を書き込んで、ヘッドを右(R)に移動する」という命令を表す。

State	on 0	on 1	on 0			on 1		
			Print	Move	Goto	Print	Move	Goto
A	B1R	D0L	1	right	B	0	left	D
B	C1R	F0R	1	right	C	0	right	F
C	C1L	A1L	1	left	C	1	left	A
D	E0L	H1L	0	left	E	1	left	H
E	A1L	B0R	1	left	A	0	right	B

Hは特別な「状態」で、この状態の時、マシンは「停止」する

プログラム本体

プログラムの説明

チューリング・マシンは、 計算可能なすべての計算のモデル

「チャーチ＝チューリングの提言」によれば、計算可能なすべての計算は、チューリング・マシンで計算できることになる。その割には、チューリング・マシンのメカニズムは、驚くほど単純なように見える。それは、チューリング・マシンが計算の本質を、見事に捉えているからなのだが、このマシンの振る舞いは、見かけ以上に複雑である。

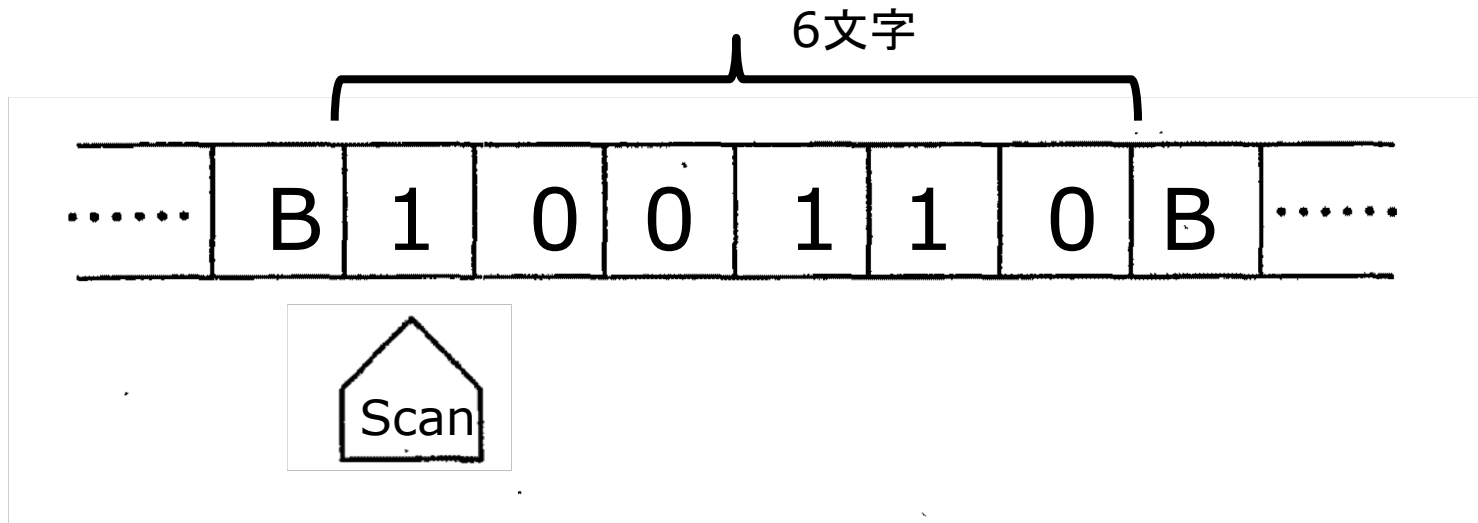
A,B,C,D,E,Fという6つの「状態」を持ち、'0','1'に対応した場合の12個のプログラムで定義されただけの次の単純なチューリング・マシンが、極めて複雑な振る舞いをすることを、あとで紹介しようと思う。

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC

チューリング・マシンでプログラムする

例1： 文字列の長さを求める

問題1：B(Blank:何も書かれていない空白のマス目)とBのあいだにある、0と1の記号の列(文字列)の長さを求めよ。



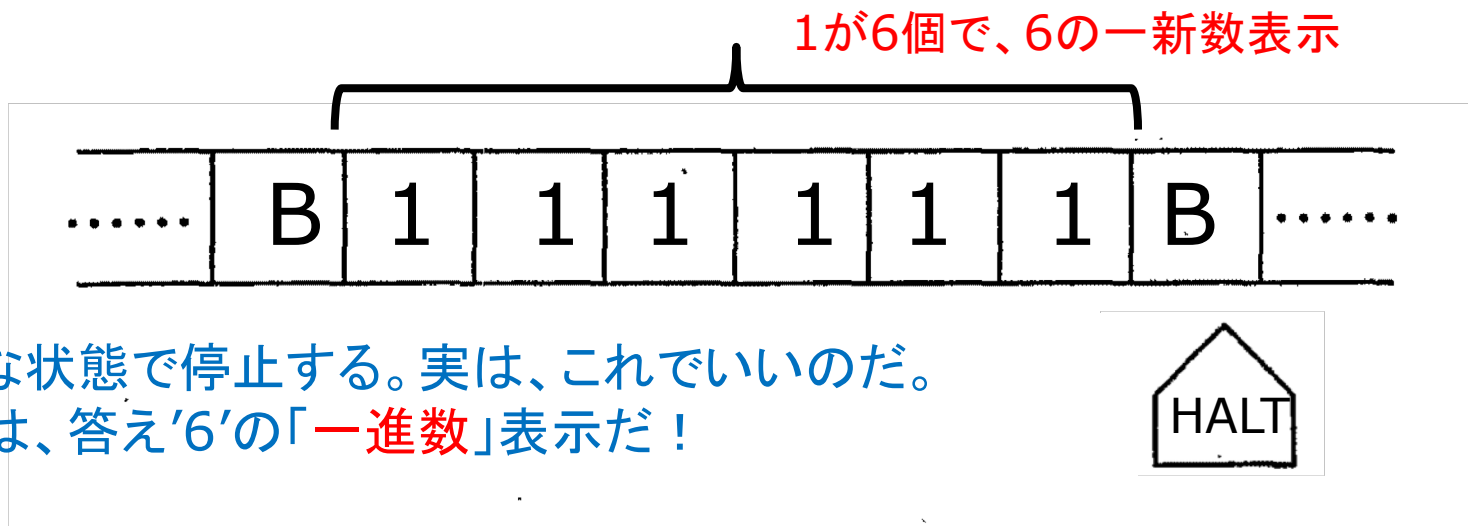
この例なら、6を返せばいい。

例1： 文字列の長さを求める

これは、状態一つでも解ける。この状態をScanとしよう。

状態Scanは、

1. '1'の上だと何もしないで、ヘッドを右に進める。状態はそのまま。
2. '0'の上だと'0'を'1'に書き換えて、ヘッドを右に進める。状態はそのまま。
3. 'B'の上だと、状態を'HALT'に変えて、停止する。

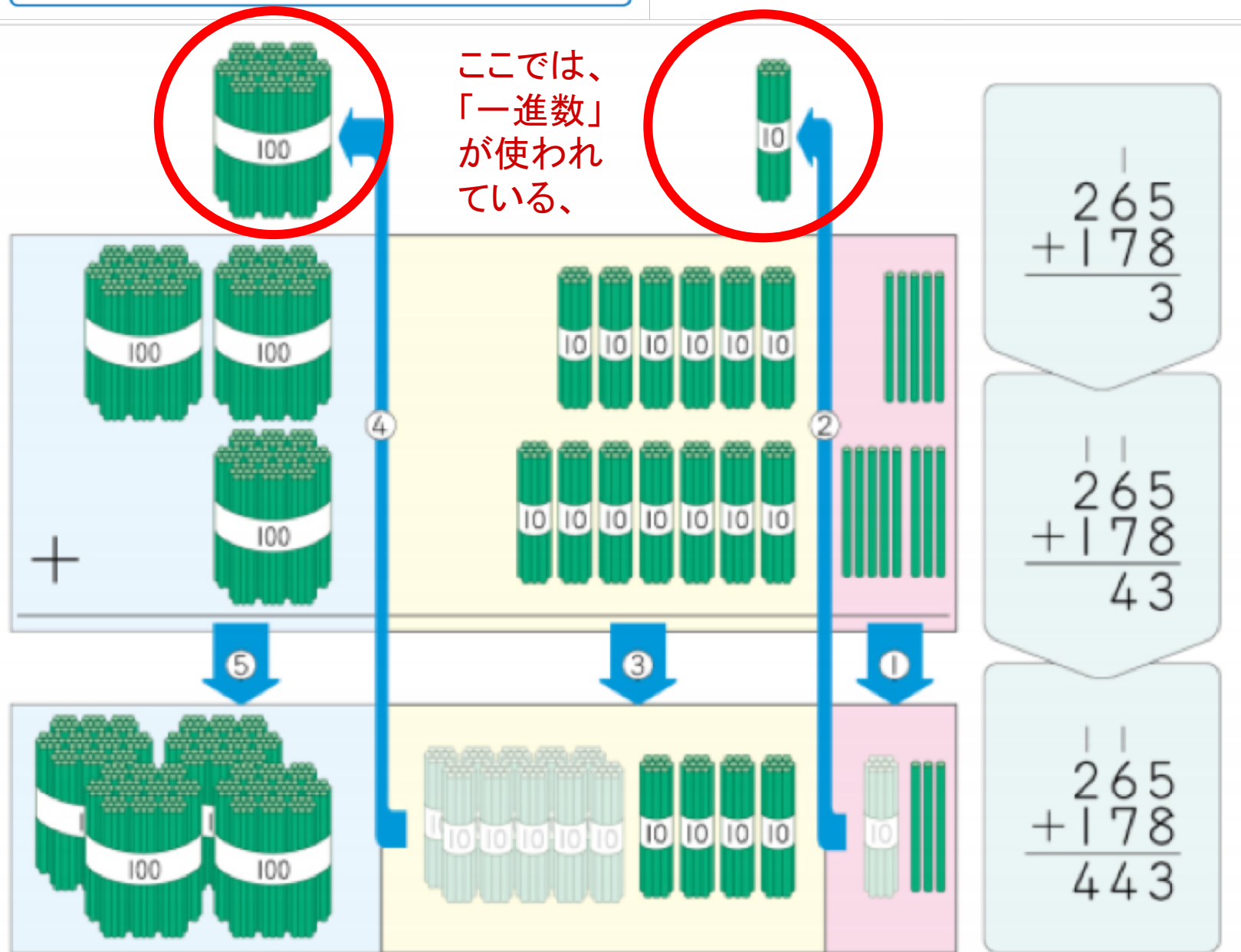


3

265+178 の筆算のしかたを考えましょう。

啓林館

3年上 p.57



例2: 文字列中の'1'の数の偶奇を求める

問題2: BとBのあいだにある、0と1の文字列中の1の数の偶奇を求めよ。偶数なら0を奇数なら1を返せ。

.....	B	1	0	0	1	1	0	B	
-------	---	---	---	---	---	---	---	---	-------



この例では、1は3個なので、奇数。1を書いて停止する

.....	B	1	0	0	1	1	0	B	1
-------	---	---	---	---	---	---	---	---	---



例2: 文字列中の'1'の数の偶奇を求める

二つの状態 Even と Odd を使う。初期状態をEvenとする。

状態Evenの時、

1. '0'の上だと何も印字しないで、ヘッドを右に進める。状態はそのまま。
2. '1'の上だと何も印字しないで、ヘッドを右に進める。状態はOddに。

状態Oddの時、

1. '0'の上だと何も印字しないで、ヘッドを右に進める。状態はそのまま。
2. '1'の上だと何も印字しないで、ヘッドを右に進める。状態はEvenに。

ただ、これだと文字列末の'B'にヘッドは来た時の動作が定義されていないし、結果の印字もできていない。

新たに、二つの状態、PrintEvenとPrintOddを追加する。

例2: 文字列中の'1'の数の偶奇を求める

状態Evenの時、

1. '0'の上だと何も印字しないで、ヘッドを右に進める。状態はそのまま。
2. '1'の上だと何も印字しないで、ヘッドを右に進める。状態はOddに。
3. 'B'の上だと何も印字しないで、ヘッドを右に進める。状態はPrintEvenに。

状態Oddの時、

1. '0'の上だと何も印字しないで、ヘッドを右に進める。状態はそのまま。
2. '1'の上だと何も印字しないで、ヘッドを右に進める。状態はEvenに。
3. 'B'の上だと何も印字しないで、ヘッドを右に進める。状態はPrintOddに。

状態PrintEvenの時、

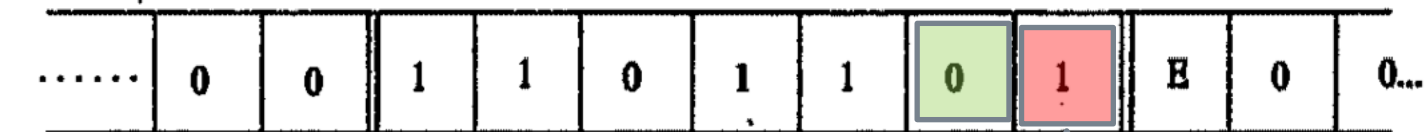
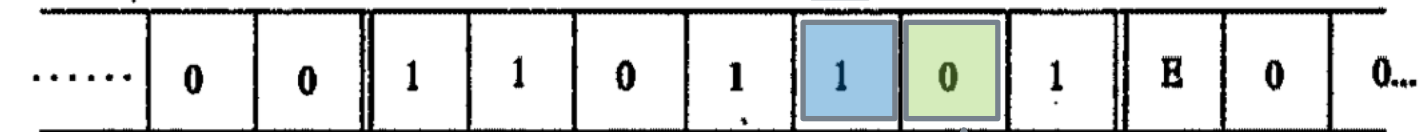
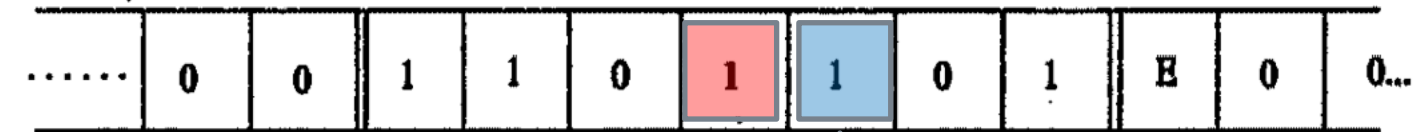
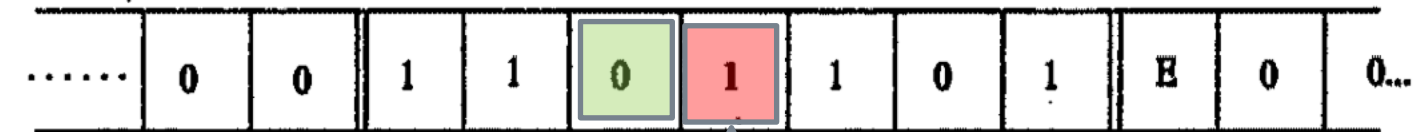
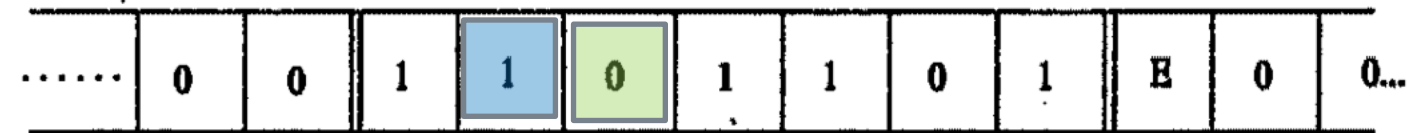
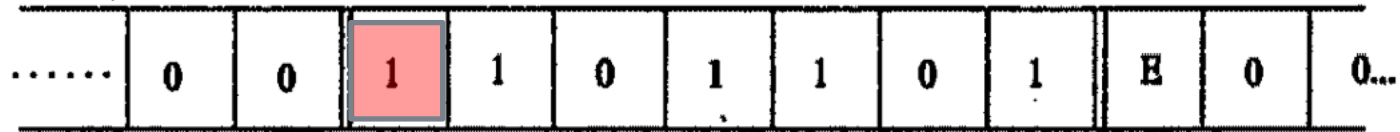
1. 'B'の上だと、0を印字して停止する。

状態PrintOddの時、

1. 'B'の上だと、1を印字して停止する。

初期状態 = '偶'

表示されているのは読み取り後のマシンの状態



例3: 文字列のコピー

問題3: AとBとの間にある、0と1からなる文字列をBCの間にコピーする。

A011011BC --> A011011B011011C

A101010BC --> A101010B101010C

A1011BC --> A1011B1011C

A101BC --> A101B101C

例3: 文字列のコピー

- 状態: Fetch /* 文字列から一文字読み取る。*/
 - '0'であれば、それを'X'に書き換え、状態をCopy0に変え、右に進む
 - '1'であれば、それを'Y'に書き換え、状態をCopy1に変え、右に進む
 - 'B'であれば、コピー終了 / それ以外なら、状態を変えず、右に進む
- 状態: Copy* /* ヘッドを'C'が見つかるまで右移動する。*/
 - 'C'以外なら、状態を変えず、右に進む
 - 'C'で、状態がCopy0であれば、その位置に'0'を書き込み、状態をMark0に変え、右に進む
 - 'C'で、状態がCopy1であれば、その位置に'1'を書き込み、状態をMark1に変え、右に進む
- 状態: Mark* /* 'C'を書き込み、状態をBack0 / Back1 に変える。*/ (略)
- 状態: Back* /* ヘッドを、'X'あるいは'Y'が見つかるまで戻す。*/
 - 'X', 'Y' 以外なら、左に一つ進む
 - 'X'で、状態がBack0であれば、'X'の位置に'0'を書き込み、状態をFetchに変えて、右に一つ進む
 - 'Y'で、状態がBack1であれば、'Y'の位置に'1'を書き込み、状態をFetchに変えて、右に一つ進む

A101BC --> A101B101C

Fetch: A101BC

Copy1: AX01B1

Mark1: AX01B1C

Back1: A101B1C

Fetch: A101BC

Copy0: A1Y1B10

Mark0: A1Y1B10C

Back0: A101B10C

Fetch: A101B10C

Copy1: A10XB101

Mark1: A10XB101C

Back1: A101B101C

Fetch: A101B101C

Halt

例4: 括弧のバランスのチェック

問題4: EとEとの間にある、左括弧 '(' と右括弧 ')' のバランスをチェックせよ。

正しくないバランスの括弧の例

$E(())()(()))(())E$

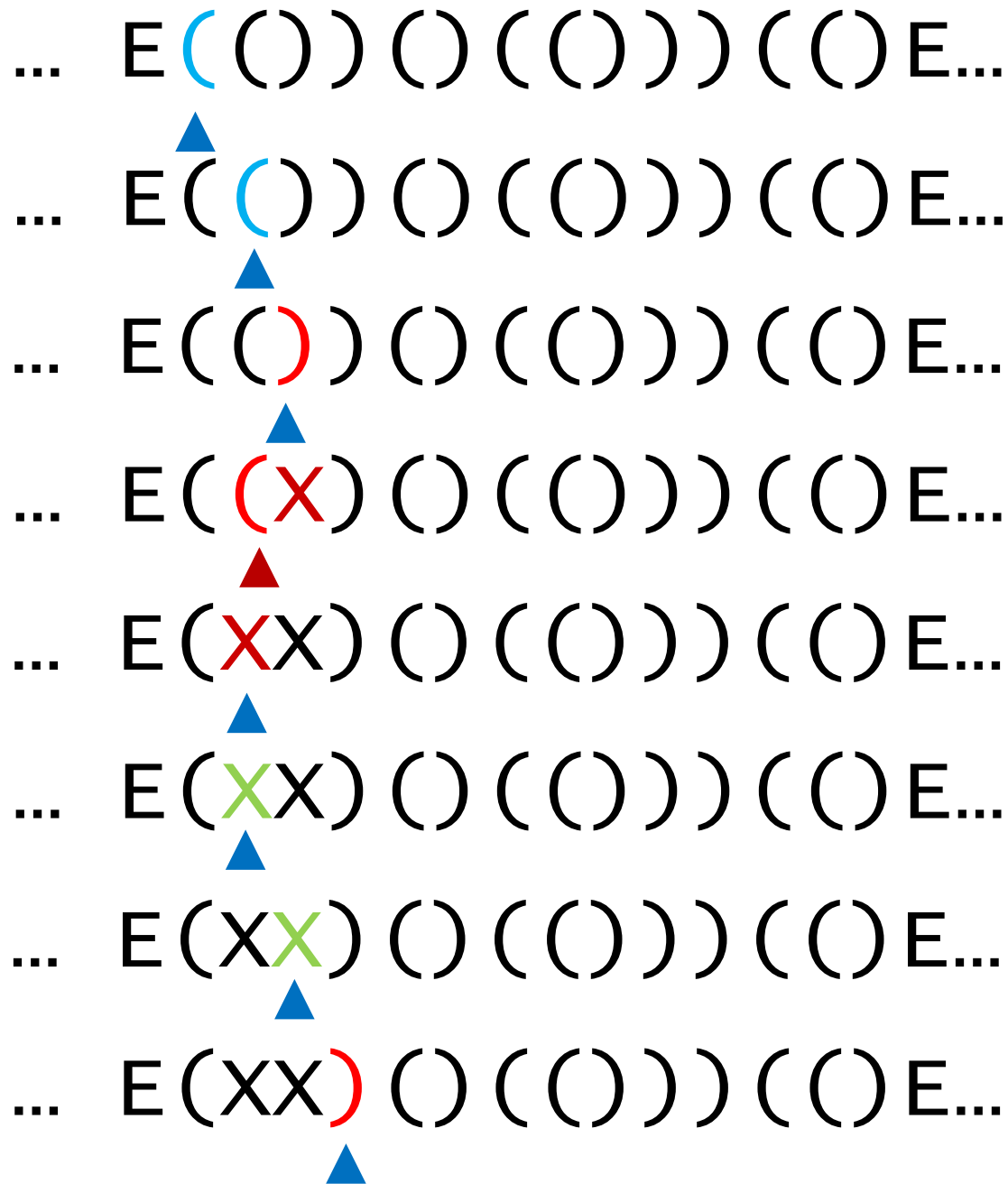
正しいバランスの括弧の例

$E(())()(())(())E$

基本的な考え方

1. 右に進んで最初に会った右括弧をXに書き換える。
2. 今度は左に進んで最初に会った左括弧をXに置き換える。
3. また、1.の手順を繰り返す。
4. 全てが、Xになっていれば、正しいバランス。そうでなければ正しくないバランス。

正しくない場合



正しくない場合

... E (X~~XX~~) (()) (()) E...



... E (~~X~~XX) (()) (()) E...



... E (~~(~~XXX) (()) (()) E...



... ~~E~~X~~X~~XX (()) (()) E...



... E~~X~~XXX (()) (()) E...



... EX~~X~~XX (()) (()) E...



... EXX~~X~~X (()) (()) E...



... EXXX~~X~~ (()) (()) E...



正しくない場合

... EXXXX() (()) (() E...

... EXXXX() (()) (() E...

... EXXXX() (()) (() E...

... EXXXX) (()) (() E...

... EXXXX) (()) (() E...

... EXXXXX) (()) (() E...

... EXXXXXX() (()) (() E...

... EXXXXXX() (()) (() E...

正しくない場合

... EXXXXXX (()) () E...

... EXXXXXX (**(X)**) (() E...

... EXXXXXX(**xx**))(() E...

... EXXXXXX(xx))(() E...

... EXXXXXX (xX)) (() E...

... EXXXXXX (xx) (() E...

... EXXXXXX (X~~X~~X) (() E...

... EXXXXXX (xxx) () E...

正しくない場合

... EXXXXXXXX (XXX) () E...
▲

... EXXXXXXXX XXXX) () E...
▲

... EXXXXXXXX XXXX) () E...
▲

... EXXXXXXXX XXXX) () E...
▲

... EXXXXXXXX XXXX) () E...
▲

... EXXXXXXXX XXXX) () E...
▲

... EXXXXXXXXXXX) () E...
▲

... EXXXXXXXXXXX XX () E...
▲

正しくない場合

... EXXXXXXXXXXX~~X~~XX () E...



... EXXXXXXXXXXX~~X~~XXX () E...



... EXXXXXXXXX~~X~~XXXXX () E...



... EXXXXXX~~X~~XXXXXX () E...



... EXXXXX~~X~~XXXXXXXXX () E...



... EXXXX~~X~~XXXXXXXXXX () E...



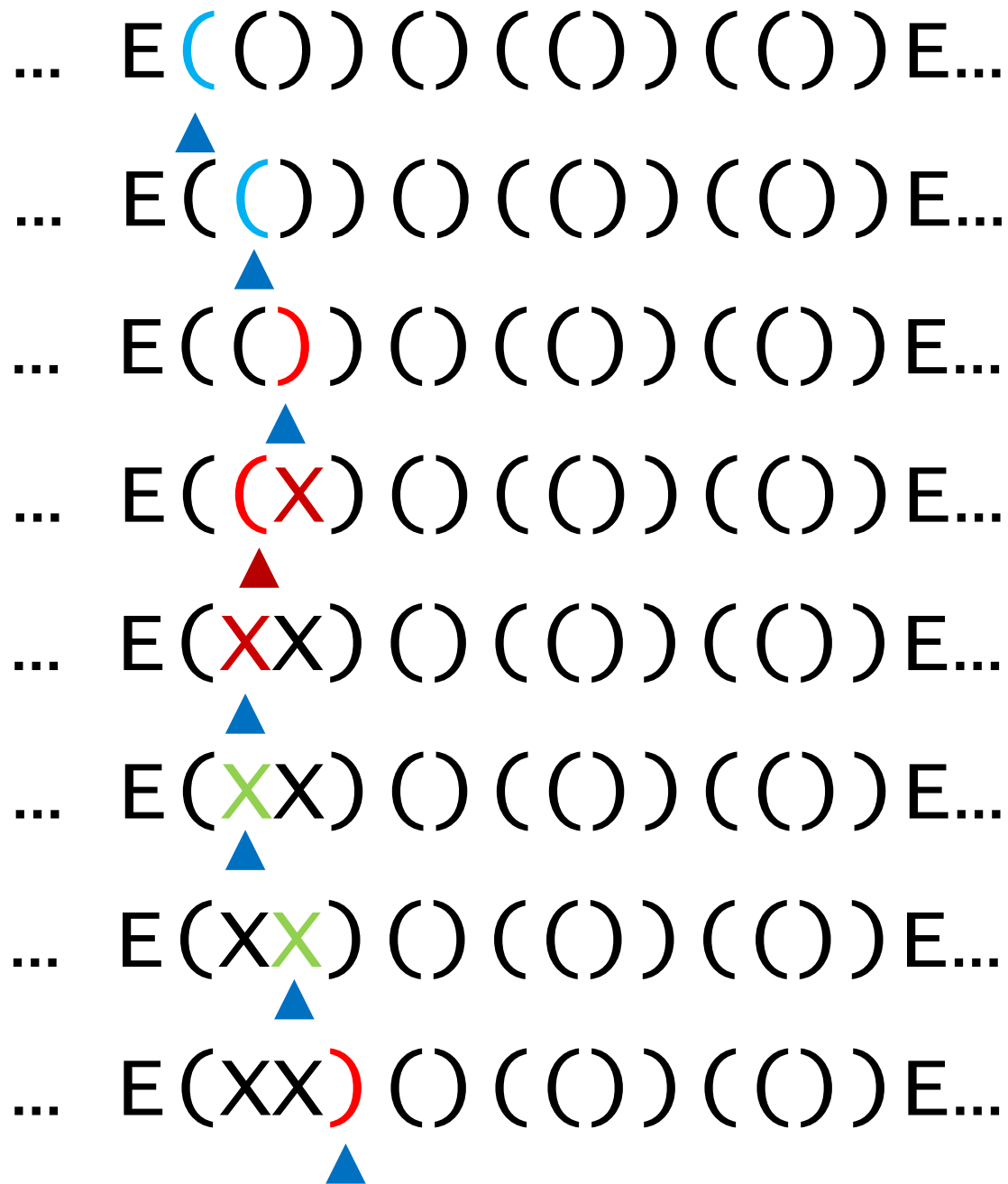
... 省略 ...

停止

... ~~E~~XXXXXXXXXXXXXXXXX () E...



正しい場合



正しい場合

... E (X~~X~~X () () ()) E ...
 ▲
... E (X~~X~~XX () () ()) E ...
 ▲
... E (X~~X~~XX () () ()) E ...
 ▲
... E X~~X~~XXX () () ()) E ...
 ▲
... E X~~X~~XXX () () ()) E ...
 ▲
... E X~~X~~XXX () () ()) E ...
 ▲
... E X X~~X~~XX () () ()) E ...
 ▲
... E X X X~~X~~ () () ()) E ...
 ▲

正しい場合

... EXXXX() () () E...
▲

... EXXXX() () () E...
▲

... EXXXX(X () () E...
▲

... EXXXXXX () () E...
▲

... EXXXXXX () () E...
▲

... EXXXXXX () () E...
▲

... EXXXXXX () () E...
▲

正しい場合

... EXXXXXXXX () (()) E...


... EXXXXXXXX () (()) E...


... EXXXXXXXX () (()) E...


... EXXXXXXXX () (()) E...


... EXXXXXXXX () (()) E...


... EXXXXXXXX () (()) E...


... EXXXXXXXX ( ) (()) E...


... EXXXXXXXX () (()) E...


正しい場合

... EXXXXXXXX (XXX (()) E...


... EXXXXXXXX XXXX (()) E...


... EXXXXXXXX XXXX (()) E...


... EXXXXXXXX XXXX (()) E...


... EXXXXXXXXXX XX (()) E...


... EXXXXXXXXXXXX X (()) E...


... EXXXXXXXXXXXXX (()) E...


... EXXXXXXXXXXXXX (()) E...


正しい場合

... EXXXXXXXXXXXXX (()) E...
▲

... EXXXXXXXXXXXXX ((X)) E...
▲

... EXXXXXXXXXXXXX (X X) E...
▲

... EXXXXXXXXXXXXX (X X) E...
▲

... EXXXXXXXXXXXXX (X X) E...
▲

... EXXXXXXXXXXXXX (X X) E...
▲

... EXXXXXXXXXXXXX (X X X E...
▲

... EXXXXXXXXXXXXX (X X X E...
▲

正しい場合

... EXXXXXXXXXXXXXX (XXXE...

... EXXXXXXXXXXXXXX XXXXE...

... EXXXXXXXXXXXXXX XXXXE...

... EXXXXXXXXXXXXXX XXXXE...

... EXXXXXXXXXXXXXX XXXE...

... EXXXXXXXXXXXXXX XE...

... EXXXXXXXXXXXXXX E...

明らかにE(((Eもこの状態になる。
先頭のEまで戻って、(が含まなければならない。

計算可能だが、
とても「計算が難しい」ものがある

一般帰納関数として定義された
原理的には計算可能な「アッカーマン関数」

一般帰納関数の計算可能性

アッカーマン関数

次のような関数を考える。

$$\text{Ack}(m, n) = \begin{cases} n + 1, & \text{if } m = 0 \\ \text{Ack}(m - 1, 1), & \text{if } n = 0 \\ \text{Ack}(m - 1, \text{Ack}(m, n - 1)), & \text{otherwise} \end{cases}$$

この関数は、一般帰納関数として定義されているので、チャーチ＝チューリングのテーゼの条件を満たしている。

ただ、 m, n の値が、少しでも大きくなると、この関数の値は、爆発的に増大して、普通のコンピュータでは、事実上、計算が不可能になる。

次ページに、 $\text{Ack}(m, n)$ の値を示す。

$A(m, n)$ の値

$m \backslash n$	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2 = 2 + (n + 3) - 3$
2	3	5	7	9	11	$2n + 3 = 2 \times (n + 3) - 3$
3	5	13	29	61	125	$2^{n+3} - 3$
4	13	65533	$2^{65536} - 3$	$2^{2^{65536}} - 3$	$A(3, A(4, 3)) = 2^{2^{2^{65536}}} - 3$	$\underbrace{2^{2^{\cdot^{\cdot^2}}}}_{n+3 \text{ twos}} - 3$
5	65533	$\underbrace{2^{2^{\cdot^{\cdot^2}}}}_{65536 \text{ twos}} - 3$	$A(4, A(5, 1))$	$A(4, A(5, 2))$	$A(4, A(5, 3))$	$A(4, A(5, n - 1))$
6	$A(5, 1)$	$A(5, A(6, 0))$	$A(5, A(6, 1))$	$A(5, A(6, 2))$	$A(5, A(6, 3))$	$A(5, A(6, n - 1))$

我々に手の届かない有限について Busy Beaver 問題

Busy Beaver 問題

「忙しいビーバー」問題は、状態の数がN個のTuringマシンを全部考えて、全て0で埋まっているテープから初めて、一番長いステップで停止するTuringマシンとそのステップ数を求めよという問題である。その最大値をBB(N)で表すことにすると、次の結果が知られている。

$$BB(1) = 1.$$

$$BB(2) = 6.$$

$$BB(3) = 21$$

$$BB(4) = 107$$

実は、まだ、誰も、BB(5)の値を知らない。

$$BB(5) \geq 47,176,870$$

というのはわかっている。(誰かが、状態の数が5つで、47,176,870ステップで停止するTuringマシンを作ったということ。ただ、それが「最大」だということとはわかっていない。)

そんなものかと思われるかもしれないが、雲行きが怪しくなるのは、BB(6)あたりから。

$BB(6) \geq 7.412 \times (10^{36,534})$

だという。36,534桁の数字になる。先のBaezのblogに、「具体的に」この数字が紹介されている。延々と数字が続く。3万6千桁以上の数字が。

よくそんなことわかるなと思われるかもしれないが、それは、先にも述べたように、実際にそういう6状態のTuringマシンを構成して見せたからだ。実は、先に紹介した、6個の状態と12個の命令からなるTuringマシンが、そういう動き方をするのだ。

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC

「プログラム自体は、単純である。」と先に書いたのだが、正しくないかもしれない。このプログラムは、えんえんと走り続け、そして、最後に停止する。(停止すること自体、不思議な気がする)

74120785350949561017417256114460496971828161169529
80089256690109516566242803284854935655097454968325
70980660176344529980240910175257246046044979228025
75771151854805208765058993515648321741354119777796
52792554935324476497129358489749784615398677842157
90591584199376184970716056712502662159444663041207
99923528301392867571069769762663780101572566619882
47506945421793112446656331449750558811894710601772
36559599539650767076706500145117029475276686016750
65295229541290448711078495182631695097472697111587
32776867610634089559834790714490552734463125404673
70809010860451321212976919019625935072889346503212
31429040253205457633515626107868771760119916923828
74680371458459216127416939655179359625797982162506
60314494227818293289779254614732935080486701454668
21939225145869908038228128045266110256571782631958
92689852569468996669587422751961691118179060839800
19742149153987715916968833647534774800748757776661
12880815431005997890623859440699663891591940342058
44534513595160016891606589527654143070266884025253
13506538908970519826326303859380836606399479857223
51182179370081120817877269116559398341767052162655
91720120243332830032375287823064283197180507239529
73532295517310483710218115976193011673158268680311

73532295517310483710218115976193011673158268680311
96305710080419119304779715796727850295295594397972
94500432643483372677378872480519292318129075141594
30017042374851948272409014919776223009518089664572
07992743507711148214208698063203898384663486444006
34378985820511533007233636175063664244348363429381
71686527767592717386843513084430760538597788497854
57039288736337621694394446002685937650424904397470
70469396499307353236961551408770635423051623551580
20502046433028802860624333066826817081190086396068
05449212705508208665911831651713621990622680221463
40355100698032539208500321737935916572824167109243
92179632770888698806462239286922365720234049353262
29319760109739336919551555856149717013936570835108
29337138019755841416703048425473095781543947284695
30557891048303296887105203445899799657005379321565
69516567462536723341045028533527590132444550041637
29329181785691615066366432555395455966924963740865
92347905851808900172725987566577427348338512074400
14671953393767922133480111674181667283600828299423
61956450241322000295203110123701834947807654204715
50872484529282734610014634854736508912653344685939
21925381546951641870418349782007870841424352960246

81621927943512800969148833336725244172361946188547
20963814880877462649154982581243612288332193203522
41878334479833109122807808425070272194370533803098
15576207436359412405607991428582586135325000600450
34044570755614842353801605755138514728637444538199
91270653752636827482388619627545586769702982563550
21579190594347001678124794147520737780545710725238
09263070578948251221705378404200557518123183429763
74391628221317569903581457033268409573939140317537
92951945222572832854376667354589981221872208463941
92173302390371597480313550832469764638860694385735
23920879420538715366934472880272772745254215764827
22658077210282649639911775387884460117481542574020
98604710597497363577816224906240529468176628001243
37027642430572009172724680494845807607875336391296
35595374936463756499152341721363955306855005063147
84058597424341392606532443545414393065035952175386
45638222669677018885647203581909266795843083183075
45078527771378321186170870661268143673661410440879
12940056479135404302810843179830761186081727156785
64098233869131431441387859096996327035057252704630
66502399928551829284912464275322457081097679293349
77256939108943396587781783827866809713105339479801
94252766851530607523746692117596241149131822801952

28443629054406029354014078168448839468854310977774
64971341943282207403959764566321636691138805567444
40040338242074160211898209536897949768268405300638
55020960995862149067127133242923955216689288381118
44058888209904636044250765206694970737949801463627
09477533118591401481473656166664409698471099509772
67427126852419476331478775678441642258692918630399
93094799728916927267392317125315003396591007151226
51178203821487177649041575923087270542299624201754
57070699334124035606469963629320951287401378625068
24738877579310019018071588005151785675631912063264
63879171290239696789072427830303321073398269847363
45629019926879365533487397619450023732220399774409
78878227032599708324913637134795947392057672257001
88982988598790346622775744604841009275542606005747
73489847857869077885071196141198763333605601699259
29619179821052298166718147760782068394323831299733
35022262733114475600303463447691380426320406458971
00672747110856632877598317707068109285992387448288
96303378384828434300860309575950421131368727514681
55719991530038357093718116282958853868614897146782
54967080333475258187514533923483088945298272830364
47705805899342687014494268126923276698359373141482

44674928751199880396209243410394589961770757345685
64543015202758133002674347175029218497929457573786
65467651289853262700253064391422801251703856704304
39933674129974352639333328279021853998732340493391
52439866339607669375777654460893584609291320819482
35450294931218519646257691473024784635292462805654
60565812545710189564825444516533104654549626307774
13759501791681585406819992391995410832229305735031
79743073679478401659929359532688412917629928632646
23531586811022948074724601440807980838830594224020
70309042157187760781779401504832688794481346645989
97848941467191367820110325917723306165886986506294
31831412363348418517790881203602332829570523258317
17949497171355624217480958863040591015617418162750
63724305524405091305861072355204855928475793642357
60246280346642123778396019497710295060768286004460
92308617300735231303196433523457326103470236858178
28414431828300312373660803542368392168889742383761
80821770347653926560938368076716961022633531512872
88504183987441820108573621090001961860938961602460
20885687763639795010335265588767970024085673382770
49445342795888876360045283740672969599921612120088
19088433242165085295954910707655578576243692034551

省略

46076619939139036787876060299301996426764725427403
64011297421238171255442319681637731281152043413431
85060894391599524452878044334616995055050095307491
88403737341503747816446538950384710455426239295288
28998531209709108913722340683342671325423513366061
63474488522182030819462026736771976453958939935828
32050459497790182712782471389976977879426526570877
30339646201949991550918186788970663139748645058559
72718469140463791494759075349756092667385993283854
20778335173462088786041537266587943086744729710814
62579154103447702796046390902553975412328182470276
13052411058118309311147944260478608

steps and then halts!

有限だが計算できない数の発見について

最近になって、「忙しいビーバー」Busy Beaver 問題で、とても重要な発見があった。

何がわかったかを述べる前に、少し、状況を整理しよう。

Busy Beaver 問題というのは、 N 個の状態を保つTuringマシンが、実際に計算可能な最大ステップ数 $BB(N)$ を求めよという問題だ。

まず、 N 個の状態を持つTuringマシンが何個あるか考えよう。これは、昨日の遷移表を眺めていれば、簡単にわかる。

'B1R' や 'H1L' といった「命令」の数は、一文字目が $N+1$ 個 (状態の数 + 停止状態)、二文字目が2個 (1 か 0 だから)、三文字目が2個 (RかLだから) の可能性があるから、全部で、 $(N+1) \times 2 \times 2$ 通りの組み合わせがある。

Turingマシンは、これらの $N \times 2$ 個の命令 (一つの状態について、1の場合と 0の場合について、実行する命令を置いたものだから)から構成される。

だから、 N 個の状態を持つTuringマシンの総数は、 2^N 個のマスを、 $4(N+1)$ 個のもので埋めればいいので、 $4(N+1)$ の 2^N 乗だということがわかる。

これは、大きな数になるかもしれないが、「有限」な数だ。だから、これをかたっぱしから試していけば、「原理的」には、 $BB(N)$ は求まるはず。

と思うかもしれないが、この手続きはうまくいかない。だって、試すべきマシンの数が「有限」でも、「無限」ループにおちいるマシンにぶつかれば、この手続きは「有限」の時間では終わらないから。

ただ、こうしたループの「落とし穴」ではない、もう一つの問題がある。それは、状態の数 N やチェックすべきTuringマシンの数と比べて、 $BB(N)$ が、きわめて急速に巨大な数に膨れ上がるということだ。(単純なループにはまるのと、答えを求めて、えんえんと計算を続けるのは、多分、別のことだ。 $BB(N)$ は、そうした最長の数を求める。)

先に、 $BB(6)$ が、36,534桁の数字より大きいという話をした。2014年に、Wythagoras(怪しい名前だ)は、 $BB(7)$ について、次のような評価を示した。右側の数は、とても巨大な数だが、「有限」な数字である。

$$BB(7) > 10^{10^{10^{10^7}}}$$

冒頭に述べた「Busy Beaver 問題での重要な発見」の話をしよう。

今年(2016年)の4月に、Adam Yedidia と Scott Aaronson は、BB(7910)が、普通の集合論の枠組みの中では決定できないことを証明した。

"A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory"

<http://arxiv.org/pdf/1605.04343v1.pdf> (Scottの次のblogが参考になる。

<http://www.scottaaronson.com/blog/?p=2725>)

それ以来、BB(N)が計算不能になる、より小さいNの探索が、インターネット上で活発に行われ、この "logic hack" で、つい二週間ほど前に、BB(1919)も計算不能であることが示されたのである！

確かに、僕が生きているうちは、 $BB(10)$ の値も決まらないだろう
と思って、ほぼ、間違いないので、 $BB(7910)$ とか $BB(1919)$ と
かは、とんでもなく大きな数である。ただ、これらの数字は「有限」
で、しかも、 $BB(N)$ 自体は、とても明確に定義されたものである。
要するに、「有限」の値を持つのだが、どういう数学的な手段を持
ちいても、我々が知り得ない数字があるということ。さらに、その限
界が、7910とか1919といった、比較的小さな数字で特徴付けら
れるというのが、驚きである。

複雑さについての多様なアプローチ

コルモゴロフの複雑性

コルモゴロフの複雑性

計算機科学の分野では、「複雑さ」の尺度に、「コルモゴロフの複雑性」という概念がよく用いられる。それは、ある対象の「複雑さ」を、その対象を出力するプログラムの「大きさ」に置き換えて説明しようというアイデアに基づいている。

簡単なものを出力するプログラムは短く、複雑なものを出力するプログラムは長い。これは、プログラマーなら経験的・直感的に理解できることだと思う。

また、これもプログラマーならわかると思うのだが、同じ結果を返すその意味では同じプログラムでも、ある人のプログラムは冗長で長く読みにくく、ある人のプログラムは、コンパクトでわかりやすい。

同じ結果を返すいくらでも冗長なプログラムはかけるのだから、出力の「複雑さ」を、プログラムの「大きさ」に対応づけようとするのなら、その結果を返す「最も短いプログラム」の大きさを考えるのがいいことになる。ある対象の「**コルモゴロフの複雑性**」と言うのは、**その対象を出力する「最も短いプログラム」の「大きさ」のことである。**

もちろん、どの言語を選ぶか (Java, Python, Assembler,)、どのようなハードウェア・システム構成 (GPUの有無、データベースの有無、...) を選ぶかで、この「最も短いプログラム」の「大きさ」は変化する。

ただ、プログラムの実行環境を固定すれば、その環境のもとでの「コルモゴロフの複雑性」は、その意味では相対的にはあるが、一意に定まるはずである。

繰り返しになるが、ある対象の「コルモゴロフの複雑性」と言うのは、その対象を出力するプログラムの大きさが最小の自然数である。ある対象を出力する、同じ機能を持つ多数のプログラムがあったとしても、そのプログラム・サイズからなる自然数の全体の集まりを考えれば、その中に、最小の自然数が、一つは存在するはずである。

実行環境を固定しても、実際には、ある対象の「コルモゴロフの複雑性」を具体的に「計算」することは難しいかもしれない。この辺りが、「コルモゴロフの複雑性」がわかりにくいと感じられるところかもしれないのだが。

ただ、ここは「コルモゴロフの複雑性」理解の上で、飛躍が必要なポイントでもある。重要なことは、具体的にその数字を示すことができなくとも、また具体的にそのプログラムを示すことができなくとも、最小のプログラム・サイズが存在するのは確かであるということである。

複雑さとエントロピーについて

複雑さとエントロピーは一致するか？

コルモゴロフの複雑さでは、何の規則性もないランダムな文字の並びが、一番複雑であることになる。その意味では、エントロピーの大きさと複雑さの大きさは一致する。

ただ、それは、我々が複雑さに対して持つ直感と必ずしも一致しない。

ジャズでもクラシックでも、印象的で耳に残る複雑な和音がある。ただ、「複雑さ＝エントロピー」と考えれば、一番複雑な音は、いろんな周波数成分をランダムに含んだ音である。ホワイトノイズを聞いても、我々は、それを単なる雑音とを感じるだけで、複雑さを感じることはない。

ほったて小屋より、ガウディのサグラダ・ファミリア教会の方が複雑である。ただ、もしも、サグラダ・ファミリアが、震災か戦災で、がれきの山になったとすれば、そのがれきの山の方が、エントロピーが増大しているので、元の建物より複雑さが大きいことになる。

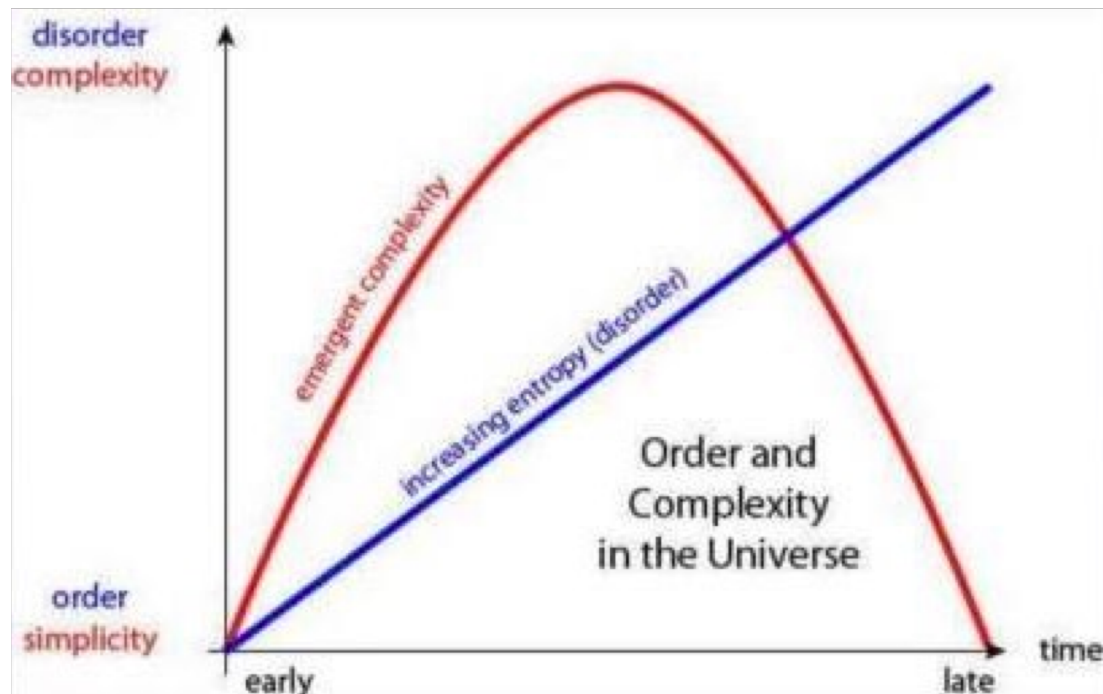
ビッグバンと宇宙の「熱的死」の複雑さ

現在の宇宙が始まったビッグバンの瞬間は、エネルギーは巨大だが、その構造はシンプルで、エントロピーは小さかった。宇宙のエントロピーは増大し続け、最後には全てがブラックホールに飲み込まれてエントロピー最大の「熱的死」を迎えるのだが、我々は、ビッグバンの瞬間も最後のブラックホールも、複雑なものとは感じないような気がする。

その中間に出現する、恒星系・銀河系の構造、生命、人間の脳の構造 等々に、我々は、複雑さを見出す。それらは決して、エントロピーが最大の状態ではない。

Scott Aaronsonは、次の図で表されるような「複雑さ」の尺度を提案している。秩序から無秩序への移行としてのエントロピーの増大(図の青い線)と、我々が感じる複雑さ(図の赤い線)とは、異なる。

すなわち、エントロピーの増大とともに複雑さも増大するのだが、ある時点を越えれば、エントロピーの増大(無秩序化)とともに、複雑さは、減少するというのだ。



彼があげている例が面白い。

コーヒーにミルクを加える。最初(図の左)は、コーヒーとミルクの層が完全に分離しているので構造はシンプルで、エントロピーは低く、複雑さも低い。時間とともにコーヒーとミルクは混じり始める(図の中央)ののだが、この状態は、エントロピーは、中程度だが、複雑さは高い。

最後(図の右)には、二つが完全に混ざり合う。この時、エントロピーは最大になるのだが、複雑さは低くなる。



チャイティンの不完全性定理

チャイティンの不完全性定理

コルモゴロフは、旧ソ連の数学者で、「コルモゴロフの複雑性」が定式化されたのは、1960年代のこと。今から50年近く前の話だ。「コルモゴロフの複雑性」が奇妙な性質を持つことを、驚くような形でまとめたのは、チャイティン(G. J. Chaitin)である。

「どのような特定のビット列をとっても、そのコルモゴロフの複雑性が L 以上であることを証明できないような数 L が存在する。」

これを「チャイティンの不完全性定理」という。

どう言うことか？

いくらでも複雑なものは存在する。どんな数に対しても、その数以上のコルモゴロフ複雑性を持つビット列は、無限に存在することを、我々は証明できる。ただ、特定のビット列が与えられた時、そのコルモゴロフ複雑性が L 以上であることを証明できない L が存在すると言うのだ。

いくらでも複雑なものを、我々は、無限に考えることができる。複雑なビット列を考えて、それとは違うもっと複雑なビット列を考えて、もっと考えて、..... これを無限に繰り返しても、具体的なビット列を考える限り、その複雑さは L を超えないと言うのだ。

まるで、孫悟空がいくら飛び回っても、お釈迦様の手のひらから飛び出すことができないように。複雑さを求めて、宇宙の果てまで来たつもりだったのに、そこには超えられない壁 L があることを見つけたと言うことだ。

「複雑さ」(我々が具体的に考える)には、乗り越えられない壁 L があるのだ。

そうすると、 L は、とんでもなく巨大な数だろうと考えたくなるのだが、 L は、驚くほど小さいらしい。

compositionality というのは、複雑なものが単純なものから構成されるという性質のことだと言ったが、単純なものから複雑なものを構成するルールを N ビットで記述できるとすると、複雑さの壁 L は、

$$L \leq N + 2359$$

程度だという。正確に、 L の値を求めることはできないが、あるコンピュータ科学者は、数キロバイトだと推測している。

小さい！

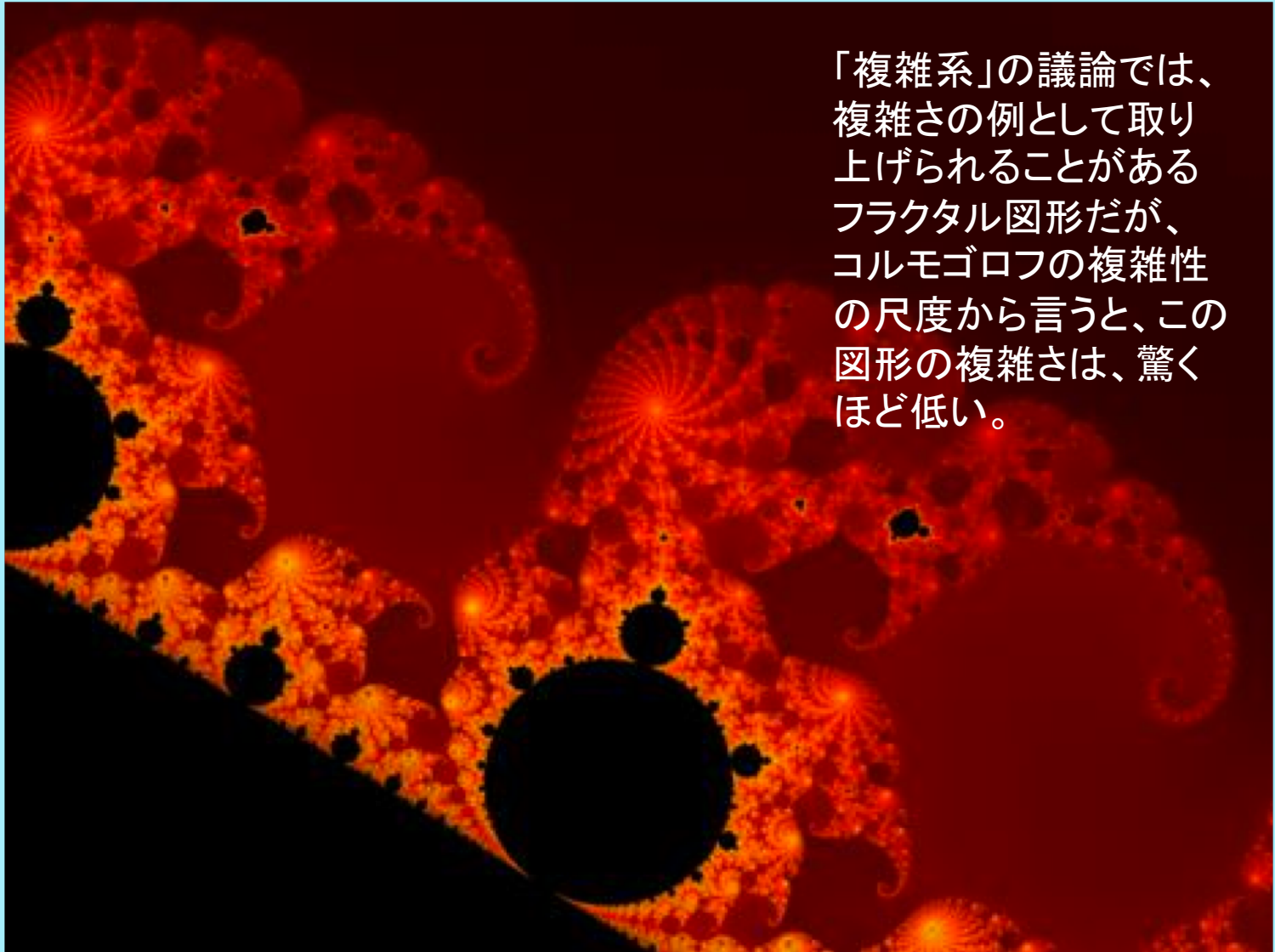
特定のビット列を出力するプログラムは、そのビット列がどんなに複雑でも、数キロバイトに収まるということで、皆さんが普段作っているプログラムが、数キロバイトになるはずだということではないので、ご安心を。

チャイティンは、こういう言い方もしている。

「あるプログラムを、それより小さいサイズのプログラムが同じ出力を行うことがない時、"エレガント"と呼ぶことにしよう。あるプログラムが "エレガント" であることを、我々は証明できない。」

実際に、小さなプログラムでも、複雑なビット列を生み出すことができることの例を二つ紹介しよう。

一つ目は、マンデルブローのフラクタル図形の例である。この画像は、1ピクセルあたり24bitカラーを用いていて、全体で、162万bitあるのだが、これを生成するプログラムは、本当に小さい。



「複雑系」の議論では、複雑さの例として取り上げられることがあるフラクタル図形だが、コルモゴロフの複雑性の尺度から言うと、この図形の複雑さは、驚くほど低い。

二つ目の例は、DemoSceneのコンテストで優勝した動画。見知らぬ惑星の描写だが、音楽含めて、プログラム・サイズは、たったの4Kだという。信じられない！

<https://www.youtube.com/watch?v=I5CTFMuFvb0>

「やさしい計算」と「難しい計算」 --「計算複雑性理論」

計算複雜性理論

「計算の難しさ」を計算に要する 「時間」と「メモリー」で評価する

ゲーデル、チューリングらが明らかにした「証明可能＝計算可能」なものには限界があることは、1950年代には、一般に認められるようになった。

ただ、それは、我々の認識の「限界」としては、非常に荒い、かつ、抽象的で原理的な限界を与えるものでしかなかった。

70年代に入ると、「計算の難しさ」に対する新しいアプローチが活発になる。それは、「計算の難しさ」を、計算に要する「時間」と「メモリー」で評価しようというものである。それを「計算複雑性理論」という。

「多項式時間」と「指数関数的時間」

計算時間について言えば、このアプローチで重要なのは、「多項式時間」と「指数関数的時間」を区別することだ。

多項式の例:

$$n^3 + 10n^2 + 8n + 1$$

$$5n^4 + 9n^3 + 7n^2 + 3n + 5$$

指数関数の例:

$$3^n + 5^n$$

$$2^{2^n} + 3^n + 5$$

その上で、計算時間が「多項式時間」で表される計算を「やさしい計算」、指数関数で表される計算を「難しい計算」と考えるのだ。

Ackerman関数

$m \backslash n$	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2 = 2 + (n + 3) - 3$
2	3	5	7	9	11	$2n + 3 = 2 \times (n + 3) - 3$
3	5	13	29	61	125	$2^{n+3} - 3$
4	13	65533	$2^{65536} - 3$	$2^{2^{65536}} - 3$	$A(3, A(4, 3)) = 2^{2^{2^{65536}}} - 3$	$\underbrace{2^{2^{\cdot^{\cdot^2}}}}_{n+3 \text{ twos}} - 3$
5	65533	$\underbrace{2^{2^{\cdot^{\cdot^2}}}}_{65536 \text{ twos}} - 3$	$A(4, A(5, 1))$	$A(4, A(5, 2))$	$A(4, A(5, 3))$	$A(4, A(5, n - 1))$
6	$A(5, 1)$	$A(5, A(6, 0))$	$A(5, A(6, 1))$	$A(5, A(6, 2))$	$A(5, A(6, 3))$	$A(5, A(6, n - 1))$

Busy Beaver関数

2-state	3-state	4-state	5-state	6-state	7-state
4	6	13	≥ 4098	$> 3.5 \times 10^{18\ 267}$	$10^{10^{10^{10^{18\ 705\ 353}}}} >$

計算複雑性理論の基本的クラス

P問題

あるチューリング・マシンが、入力の文字列の長さを n とするある問題を、 n の「多項式時間」で解く時、この問題を「P問題」という。

「P問題」は、「やさしい計算」問題と考えていい。

計算複雑性の理論で、「P問題」と並んで重要なクラスに「NP問題」がある。「NP問題」は、解くのに「指数関数的時間」がかかる「Not-P問題」ではない。それについては、次に説明する。それでも、

「NP問題」は、「難しい計算問題」と考えていい。

NP問題

NPクラスの説明によく用いられるのは、素因数への分解問題である。ある自然数 N を「多項式時間」で素因数に分解するアルゴリズムは、現在では、まだ見つかっていない。(多分、そんな多項式時間アルゴリズムは見つからないだろうと感じている人が多いのだが、かといって、そのことが確定しているわけではない。)

一般の数 N ではなく、条件を緩めて、 N は二つの素数の積であることがわかっていたとしても、 N が大きな数になると、 N を素因数に分解すること、すなわち、 $N=pq$ であるような二つの素数 p, q を見つけることは、極めて難しくなる。

ただ、 N, p, q が具体的に与えられているなら、「 N は、素数 p と素数 q に、素因数分解される」という主張は、簡単に検証できる。 p と q を掛けて、その結果が N に等しいことを示せばいいのだから。掛け算は、もちろん簡単な計算で、多項式時間で終わる。

こうした、その問題を解くのが「多項式時間」で終わるかどうかはわからなくても、具体的な値で与えられたその問題の「答え」と言われるものが、本当に「正しい答え」であるかどうかを「多項式時間」でチェックできる問題のクラスを、NPという。素因数分解問題は、NP問題である。

(ちなみに、「答え」(なるもの)が与えられた時、それが「間違った答え」であるかどうかを「多項式時間」でチェックできる問題のクラスを、co-NP問題という。素因数分解問題は、co-NP問題でもある。)

こうしたことや、NP問題である素因数分解問題に現れる「難しさ」と「易しさ」の「非対称性」が、現在の公開キー暗号の基礎になっていることは、多くの人が既に知っていることかもしれない。

P=NP?問題

「計算複雑性理論」の中心問題は、このPとNPの関係である。もしも、PとNPが実際には等しいクラスであることが証明できれば、見かけは「むずかしい計算問題」に見えるNP問題が、実は「やさしい計算」で解けることになる。

多くの数学者は、PとNPは、等しくないと考えている。

ただ、この「P=NP?問題」は、現在に至るも解決されていない。

この問題は、現代数学上の未解決問題の中でも最も重要な問題の一つであり、2000年にクレイ数学研究所のミレニアム懸賞問題の一つとして、この問題に対して100万ドルの懸賞金がかけられている。

NP-hard

**NP-
complete**

NP

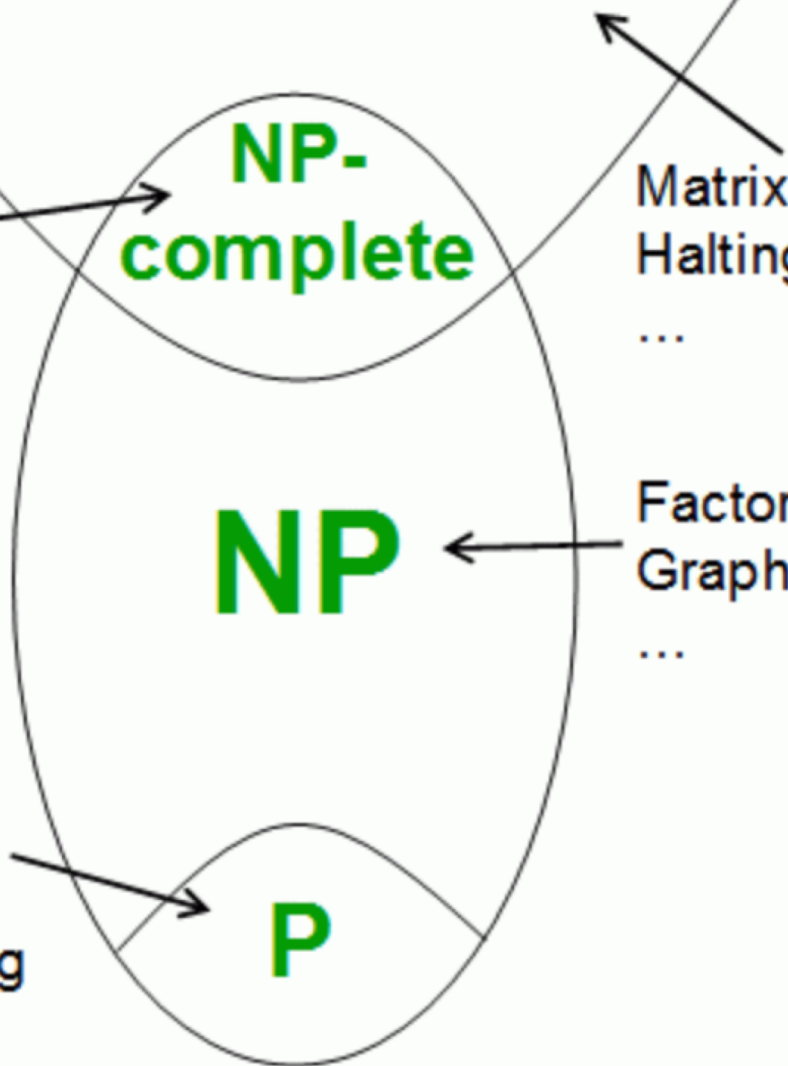
P

Hamilton cycle
Steiner tree
Graph 3-coloring
Satisfiability
Maximum clique
...

Graph connectivity
Primality testing
Matrix determinant
Linear programming
...

Matrix permanent
Halting problem
...

Factoring
Graph isomorphism
...



50年代



チャーチ=チューリング
のテーゼ
計算可能性理論

「証明可能＝計算可能」なものには限界があることは、
1950年代には、一般に認められるようになった。
ただ、それは、我々の認識の「限界」としては、非常に
荒い、かつ、抽象的で原理的な限界を与えるものでしかなかった。

70年代

$P=NP?$

計算複雑性理論

チャーチ=チューリング
のテーゼ

計算可能性理論



計算複雑性理論の基本問題である、「 $P = NP ?$ 問題」
(それは、「難しい計算」が「易しい計算」には還元できないこと
を主張する)は、現在でも未解決の問題である。

複雑性理論の発展

量子コンピュータの提唱と 「量子複雑性理論」の登場

「計算可能性理論」が、大きな転機を迎えるのは、1980年代に入ってからだ。

ファインマンが、コンピュータでは量子力学の法則に従う自然のシミュレートができないことに気づく。彼は、自然のシミュレートが可能なコンピュータは、量子力学の法則に従ったコンピュータでなければならないと主張する。

この指摘が、「量子コンピュータ」研究の始まりである。

数学的体系と同様に、自然もまた、我々の認識の対象である。数学だけでなく、物理学もまた、我々の認識の可能性と限界について、強い関心を持っているのだ。

ドイッチェは、ファインマンの考えを受けて、チューリングマシンの量子版を構成し、こうした量子チューリング・マシンで計算可能なものが計算可能であるとする。これを、計算可能性についての「チャーチ＝チューリング＝ドイッチェのテーゼ」という。

一見すると、同じような定式化に思えるのだが、本質的な違いがあるのだ。

「チャーチ＝チューリングのテーゼ」は、抽象的・形式的・数学的な「計算可能性」の定義についての提言なのだが、「チャーチ＝チューリング＝ドイッチェのテーゼ」は、「計算可能性」が、実在的・物理的に定義されねばならないと主張しているのだ。

「チャーチ＝チューリングのテーゼ」は、いわば、遠くの雲の上に抽象的に存在する原理だったのだが、「チャーチ＝チューリング＝ドイッチェのテーゼ」は、地上に降りた現実の物理的原理だ。

重要なことは、こうした「計算可能性」概念の「物理化」の背景にある思想である。それは、情報過程が、けっして抽象的なものではなく、物理的なものに支えられた物理過程に他ならないという考えである。

ただ、80年代は、まだ、量子コンピュータは概念としてしか存在していなかった。「計算可能性」の物理化という画期的な変化も、まだまだ、抽象的な議論だった。

80年代

P=NP?

計算複雑性理論

チャーチ=チューリング
のテーゼ

計算可能性理論

計算可能性概念の「物理化」
情報過程＝物理過程

チャーチ=チューリング=ドイッチェ
のテーゼ

物理的計算可能性



80年代は、まだ、量子コンピュータは概念としてしか存在していなかった。「計算可能性」の物理化という画期的な変化も、まだまだ、抽象的な議論だった。

ショアの素因数分解アルゴリズムと 新しい計算複雑性のクラスBQPの発見

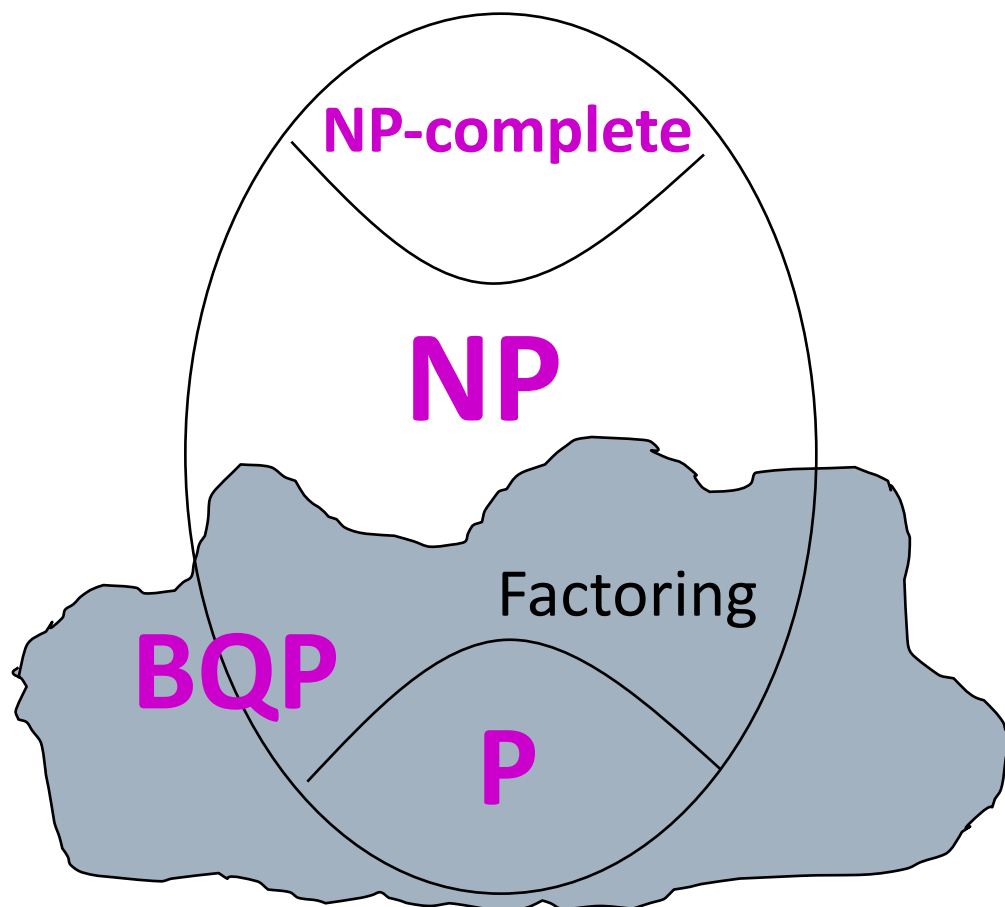
1994年、ショアは驚くべき発見をする。(論文は、1997年)
量子コンピュータでは、古典的なコンピュータでは指数関数的時間を要する素因数分解が、多項式時間で解けるという発見である。
現代のセキュリティ技術の根幹部分が、RSA暗号のように素因数分解の困難性に基礎を置いている事情から、この「ショアのアルゴリズム」の発見は、コンピュータ・サイエンスの枠を超えた、社会的と言っていい大きな反響を呼び起こした。

*If computers that you build are quantum,
Then spies everywhere will all want 'em.
Our codes will all fail,
And they'll read our email,
Till we get crypto that's quantum, and daunt 'em.*

BQP (Bounded-Error Quantum Polynomial-Time): The class of problems solvable efficiently by a quantum computer, defined by Bernstein and Vazirani in 1993

Interesting

Shor 1994: Factoring integers is in **BQP**



それは、量子コンピュータで多項式時間で計算可能なクラスBQPに、素因数分解問題が含まれるという発見であった。素因数分解は、量子コンピュータでは、「やさしい計算問題」になるのだ！

90年代

P=NP?
計算複雑性理論

チャーチ=チューリング
のテーゼ
計算可能性理論

計算可能性概念の「物理化」
情報過程=物理過程

チャーチ=チューリング=ドイッチェ
のテーゼ

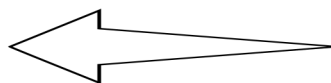
物理的計算可能性



拡大されたチャーチ=チューリング
のテーゼ



BQP



量子コンピュータ

量子複雑性の理論が登場し、BQPという新しい複雑性の概念が確立する。ショアのアルゴリズムは、広い関心をあつめた。

物理学と量子情報理論

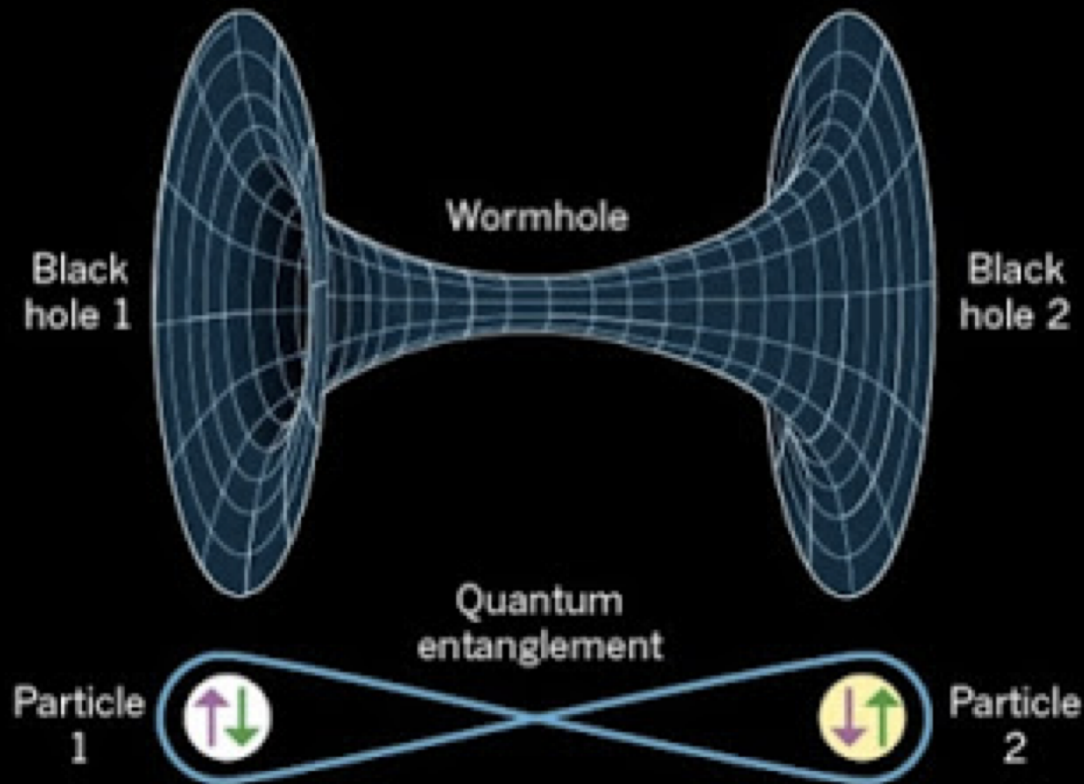
Susskind



21世紀に入ると、物理学の世界で、量子情報理論と複雑性理論への関心が急速に拡大する。

ER = EPR

Also in 1935, Einstein and Rosen (ER) showed that widely separated black holes can be connected by a tunnel through space-time now often known as a wormhole.

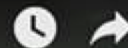


Physicists suspect that the connection in a wormhole and the connection in quantum entanglement **are the same thing, just on a vastly different scale.** Aside from their size there is no fundamental difference.

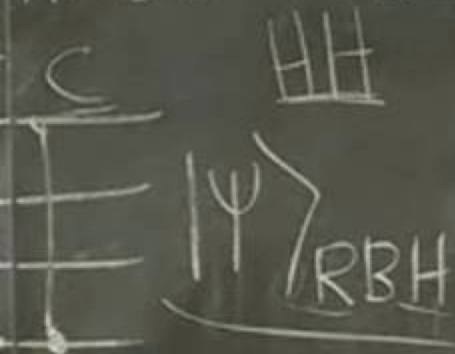
Natureの2015年11月の記事 "The quantum source of space-time"
<https://goo.gl/x0HqA2>

Aaronson

Scott Aaronson - Computational complexity underpinnings of the Harlow-Hayden argument



HAYDEN ARGUMENT



Size (#gates)
 $\leq n^{O(1)}$

$\exists$ 1 qubit of
distillable entanglement
between R & B

Alice's task

Apply U_R to R
so that

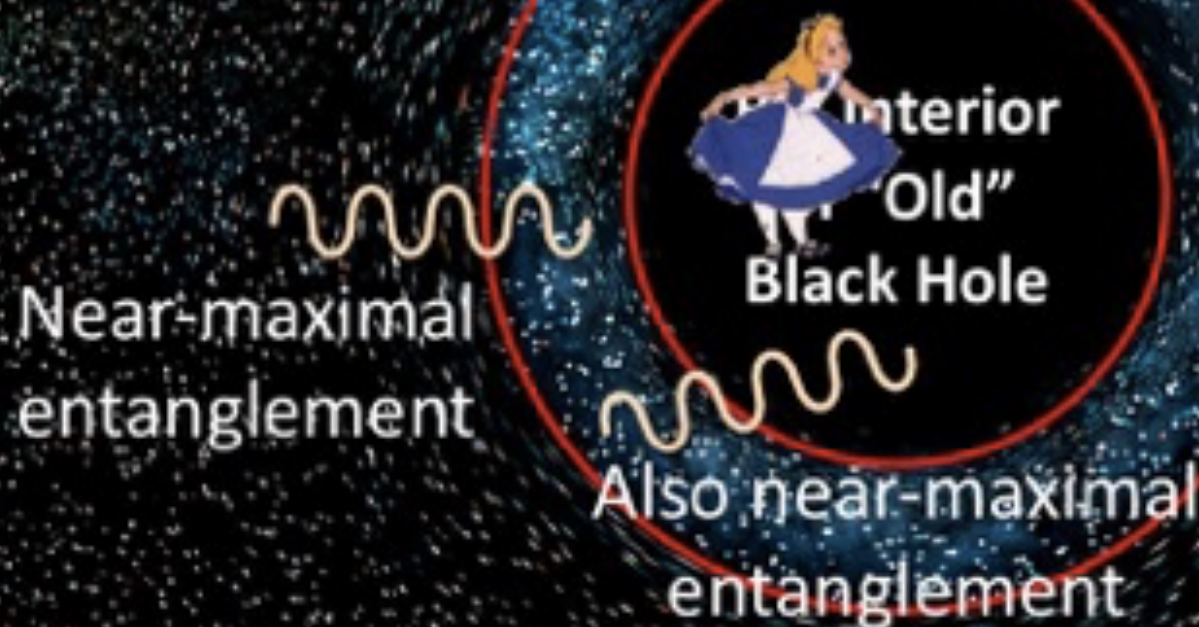
23:57 / 1:17:37



The Firewall Paradox (AMPS 2012)

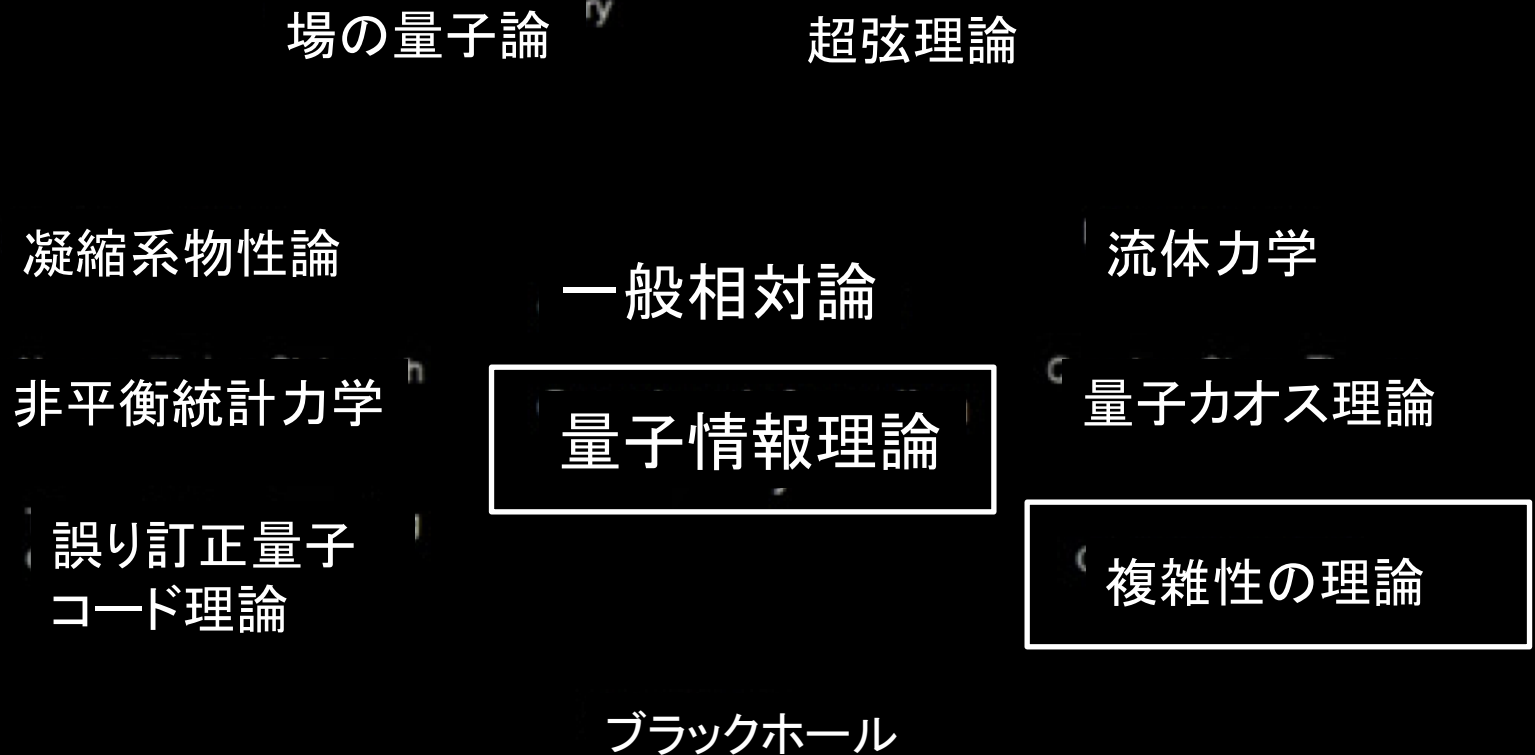
R = Faraway Hawking Radiation

H = Just-Emitted Hawking Radiation



Violates monogamy of entanglement! The same qubit can't be maximally entangled with 2 things

現代物理学の俯瞰図 by Susskind



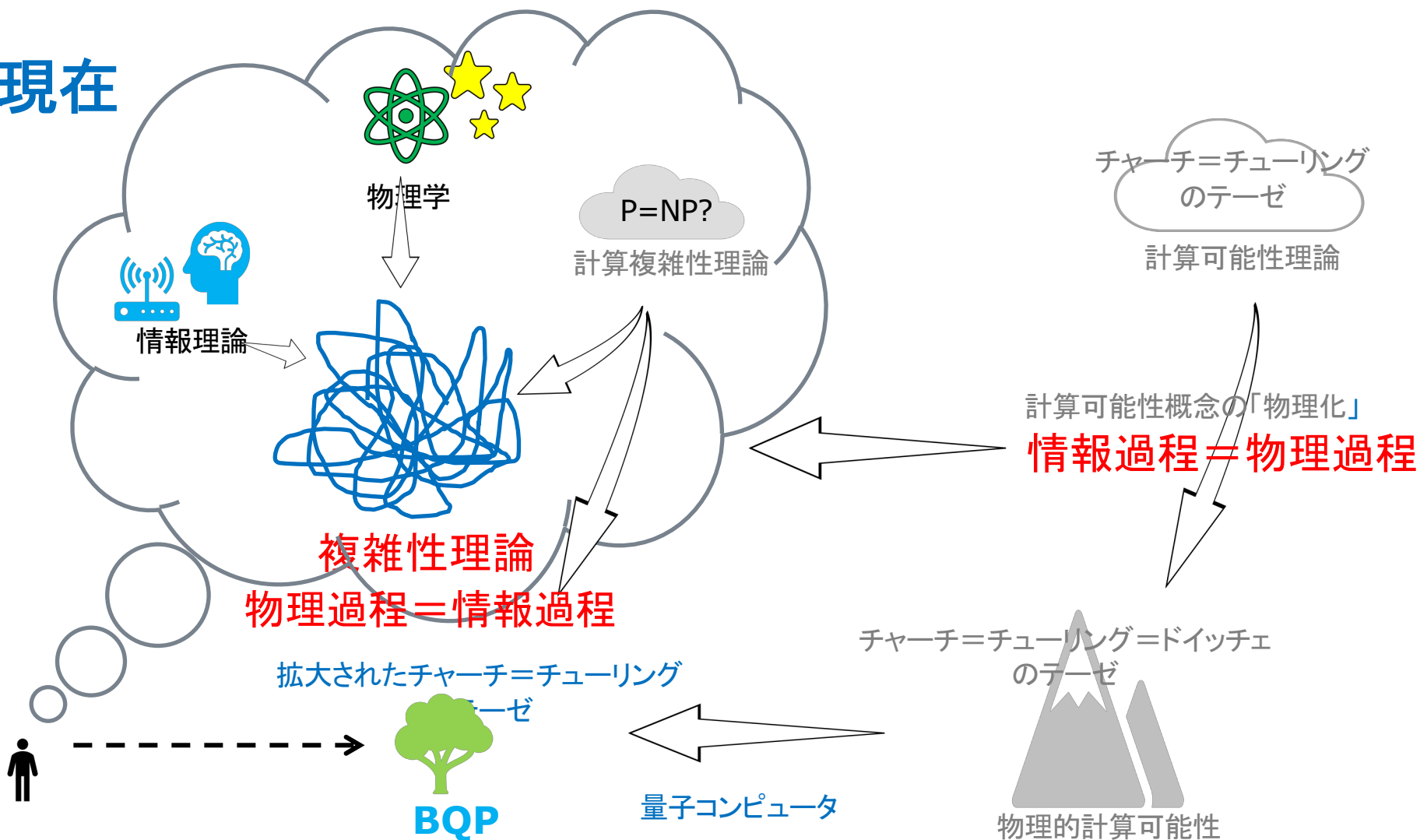
Susskind "Entanglement and Complexity: Gravity and Quantum Mechanics"
<https://goo.gl/J0wSkf>

複雑性理論の現在と 人工知能論の課題

我々の認識の「限界」の認識が進む一方で、 我々の世界の認識が拡大する

- 複雑性理論は、かつての数学的・抽象的な認識の限界の理論（「チャーチ=チューリングのテーゼ」）から、物理的・具体的ではあるが、観念の中にしかないものを対象にした限界の理論（「チャーチ=チューリング=ドイッチェのテーゼ」）へと発展してきた。かつ、それは、量子コンピュータが現実味を増すにつれて、より具体的な量子複雑性のもとでの限界の理論（「拡大されたチャーチ=チューリングのテーゼ」）へと変化してきた。
- それは、我々の認識の「限界」の認識が、より一層、精緻で現実的なものに進んだということである。
「我々が知りうることは、限られている。」
- その一方で、そうした限界の認識の深化は、我々の世界の認識が拡大する過程と一体なのだ！
「我々が知りうることは、拡大している。」

現在



計算の「限界」は、我々の身近の具体的な問題に近づいてきている。物理学の複雑性理論の発展もめざましい。

機械と人間の関係を、改めて考える

- 量子複雑性の問題を考えると、機械と人間の関係で新しい問題が提起されていることに気づく。今までは、人間の知能の本質を「計算」として、その抽象の下では、機械と人間の等値が可能だと論じてきた。ただ、BQP問題を解く能力は、機械に属し、人間には属さない！
- ペンローズのように、人間の知能の根底には、量子論的な過程があるとする人工知能論は既に存在する。ただ、ペンローズが無邪気に想定していたように、量子過程は、人間の知能の謎を全て解くほど万能ではないのだ。NP完全問題は、量子コンピュータでも解決不能である。そのことを、量子複雑性の議論は教えてくれる。
- ただ、機械と人間の関係を、こうした知見の下で、改めて考える必要はある。

人間と機械の系として、人工知能を考える

- 人工知能技術を「機械に何ができるか？」という視点でだけ捉えるのは、狭いのだと思う。この分野で重要なことは、「機械に何ができるか？」ということではなく、「機械と人間で何ができるのか？」ということではないかと考えている。複雑性理論での、MAあるいはAM問題の捉え方は示唆的である。どちらかを人間にどちらかを機械と考えればいい。
- 実際、COQのような「証明支援システム」をつかってすぐに気づくのは、それは人間の証明を機械が支援するシステムなのだが、それは、機械の証明を人間が支援するシステムでもあるということだ。ここでは、人間と機械が「共生」している。
- 既に、物理・科学の世界では、機械はなくてはならない、我々の認識の「同伴者」である。そこで得られた認識は、人間のものである。ただ、機械の助けなしには、我々は、そうした認識を獲得できない。

言語的認識能力の解明が重要な課題

- 数学的認識の解明は、なかなか難しい課題だと感じている。
- 人工知能研究で、言語能力の解明に力が注がれるのは間違い無い。言語が人間の認識活動の中で示す大きな役割を見れば、それは、当然である。(ただ、それには言語研究者とAI研究者の協力が不可欠だと僕は考えている。)
- 言語能力が万人に与えられてはいるが、個人の能力として現れるのに対して、数学的・科学的認識は、我々人類全体に帰属すべきものではあるが、個人の能力を超えている。

こうしたギャップを、人工知能技術が補完すると考えるのは楽しいことだ。