報告書

「自律計算の可能性を求めて」
(Looking for the possibility of autonomous computing)

第１回　夏休み総決算 !!

2002 年 9 月 13 日（金）
遠藤　友基

1　私の proposal

今回は夏休みの課題（宿題？）について報告するが、その前に私の proposal について述べておきたいと思う．

私は学部のときに理学部物理学科に在籍していた．物理ではご存知のように物体の運動則を扱い、近年でも様々な分野に応用されている．例えば、ナノテクノロジーと密接な関わり合いをもつ物性物理学、超伝導技術に端を発した低温物理学、筋肉の動きなど分子モータの特性を扱う生物物理学、核融合という未来エネルギーを模索するプラズマ物理学、物質の最小構成要素を探す純粋物理の素粒子物理学などなど．それぞれの分野が細分化・多様化されつつあっても、本質的な物理の見方は今後も使える道具であることは変わらないのではないかと考える．

そんな中、私が学部のときから興味があったのが、ミクロとマクロの関係についてだ．古典的な物理の考え方はよく知られている還元論、つまり素粒子物理学の考え方の基礎にある最小構成単位を組み上げたものがマクロな ”もの ”として現れるという考え方だ．しかし、多体問題のように Newton 力学では構成要素が多くなると、そのそれぞれの運動は解析的は解けないという問題に直面することになる．そこで 20 世紀初頭に Boltzmann が、いわゆる Boltzmann 原理によってミクロな現象とマクロな現象とをつなぐ統計力学の基礎を築いた．これにより今までは全く関係しないと思われた熱力学現象が、分子運動による構成論的な説明を可能にした．しかし、熱の対流現象に代表されるようにあるパラメータが変化し、ある閾値を超えると対象物の内部状態が劇的に変化してしまう現象については謎のままだった．このような熱平衡状態から離れた非平衡状態を扱う理論としては Prigogine の散逸構造論が知られている．また、多数自由度を縮約した秩序パラメータが系の状態を決めるとした Haken の隷属原理も、このような自己組織化論の 1 つである．これは物理のみならず、化学や生物の分野をも巻き込んだ新たなパラダイムとなっている．

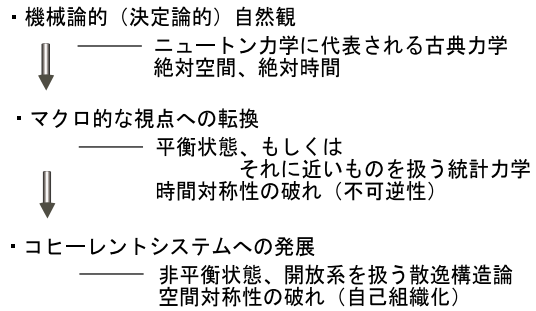


図 1: マクロとミクロの捉え方の変遷

やや前置きが長くなったが、私がここで考えるのは以上のような自己組織化、自律生成の考え方を情報システムの中でも使えないかということである．現在の通信工学などの分野では当たり前に使われている

Shannon の情報理論が Boltzmann 原理に立脚しているように、物理的な考え方のアナロジーで、この問題を解決する手法があるのではないだろうか．Shannon の情報理論が扱うデータに対し、操作情報である OS やソフトウェアの部分を自律的に生成できないのだろうか．この考えが私の proposal となっている部分である．

清水 [1] の考え方によれば、操作情報を生み出すのは拘束条件であるという．ではシステムがどのように拘束条件を保持、あるいは創生していくのか？　この辺りが考える鍵になりそうな部分である．

2　計算 (computation) とは [2], [3]

自律したシステムを考える前に、従来のコンピュータが行ってきた計算理論というものを着実に掴んでおかなくてはならない．ここでは数学的な帰納的定義に始まり、チューリングマシンからアルゴリズムの問題に至るまでをざっと概観する．

2.1　帰納的関数による計算の表現 [4]

計算するとは一連の関数の流れであるといえる．だからこそアルゴリズムは流れ図 (follow chart) として表現できる．流れ図の 1 つ 1 つのコンポーネントが関数に他ならない．まず、ここでは計算を規定する関数というものが、どのようにして構成されるかを紹介する．

2.1.1　帰納的であることの意味

帰納 (induction) とは、演繹 (deduction) の対となる推論法である．演繹が 1 つの普遍的な命題から複数の特殊な命題を引き出すのに対し、帰納とは複数の特殊な命題から 1 つの普遍的な命題を導く推論方法のことを指す．これを数学的に発展させた数学的帰納法については、高等学校で習われているのであえて深入りはしないが、数学的帰納の基本的なエッセンスは、

数学的帰納法　x についての任意の述語 $A(x)$ に対し、
1.$A(0)$ が成り立ち
2. 任意の $k \in \mathbf{N}$ に対し、$A(k)$ ならば $A(k+1)$ が成り立つ
ならば、すべての $n \in \mathbf{N}$ に対して、$A(n)$ が成り立つ．

というものである（$\mathbf{N}$ は自然数の集合を表す．以下同様）．

以上のような数学的帰納法の定義 (以下、帰納的定義という) をもう一度考えてみると、自然数上で規定された述語 $A(n)$ は自動的にすべての自然数 n について成立することが保証されていることを示している．これは自然数自体が帰納的定義によっており、このような性質が公理によって要請されているからである．この帰納的定義は強力であり、対象が述語や関数が対象になっても、これらを帰納的定義によって定めることができる．以下では関数についての議論に移る．

2.1.2　帰納的に定義される関数

計算は関数において表現することができる．例えば、$y = f(x)$ という見慣れた形は、$\forall x \in A$ という集合から、$\exists y \in B$ という集合への写像 f と考えることができる．結局のところ、コンピュータ内で行われる計算も、このような関数による写像の繰り返しといえる．ここで関数自体が帰納的に定義されていれば、かなりの数の関数の値を求めることが可能になるのだ．

ここで自然数集合 $\mathbf{N}$ 上の n 引数関数を作る操作を考える．

1. **定数関数**

$$C_k^n(x_1, \cdots, x_n) = k \tag{1}$$

2. **後者関数**

$$S(x) = x + 1 \tag{2}$$

3. **射影、恒等関数**

$$I_k^n(x_1, \cdots, x_n) = x_k \tag{3}$$

4. **合成**

m 変数関数 g と m 個の n 変数関数 $h_1, \cdots, h_m$ から、n 変数関数 f を作る.

$$f(x_1, \cdots, x_n) = g(h_1(x_1, \cdots, x_n), \cdots, h_m(x_1, \cdots, x_n)) \tag{4}$$

5. **原始帰納**

(a) 2 変数関数 g から、1 変数関数 f を作る.

$$\begin{cases} f(0) = k \\ f(x+1) = g(x, f(x)) \end{cases} \tag{5}$$

(b) n 変数関数 g と、$n+2$ 変数関数 h から、$n+1$ 変数関数 f を作る.

$$\begin{cases} f(0, x_1, \cdots, x_n) = g(x_1, \cdots, x_n) \\ f(x+1, x_1, \cdots, x_n) = g(x, f(x, x_1, \cdots, x_n), x_1, \cdots, x_n) \end{cases} \tag{6}$$

6. **最小化 (μ-作用素)**

$\mu y P(x_1, \cdots, x_n, y)$ は $P(x_1, \cdots, x_n, y)$ が真となる最小の y を意味する. これにより、$n+1$ 変数述語 P から n 変数関数 f を作る. ただし、P は任意の $x_1, \cdots, x_n$ に対して、$P(x_1, \cdots, x_n, y)$ が真となる y が必ず存在する.

$$f(x_1, \cdots, x_n) = \mu y P(x_1, \cdots, x_n, y) \tag{7}$$

以上のうち、1 から 3 までの関数を初期関数という. この初期関数から始めて、4 の関数の合成、5 の原始帰納 (これは帰納的定義) を用いて構成される関数を**原始帰納的関数 (primitive recursive function)** という. 更に 6 の最小化の適用を許してできる関数を**帰納的関数 (recursive function)** という.

この原始的帰納関数の定義を知ると、私たちが計算で使う関数ほとんどが原始帰納的関数であることが分かる. 以下は例を挙げて、それを確認していこう.

- $f(x) = x + 4$ は

$$f(x) = \underbrace{S(S(S(S(}_{4}x))))$$

と書けるので原始帰納的である. □

- もっと一般的に $plus(x, y) = x + y$ はどうだろうか. これは

$$\begin{cases} plus(x, 0) = x \\ plus(x, y+1) = S(plus(x, y)) \end{cases}$$

と書けるので原始帰納的である．実際に書き下すと

$$\begin{aligned} plus(x,y) &= S(plus(x,y-1)) \\ &= S(S(plus(x,y-2)) \\ &\vdots \\ &= \underbrace{S(S(\cdots(S}_{y}(plus(x,0)))\cdots)) \\ &= \underbrace{S(S(\cdots(S}_{y}(x)))\cdots)) \\ &= x+y \end{aligned}$$

と分かり易い．□

- 乗算 $times(x,y) = x \times y$ は $plus$ が原始帰納的であることを使うと、

$$\begin{cases} times(x,0) = 0 \\ times(x,y+1) = plus(x, times(x,y)) \end{cases}$$

と原始帰納的であることが示せる．これは書き下すと

$$\begin{aligned} times(x,y) &= x + times(x,y-1) \\ &= x + x + times(x,y-2) \\ &\vdots \\ &= \underbrace{x+x+x+\cdot+x}_{y} + times(x,0) \\ &= \underbrace{x+x+x+\cdot+x}_{y} + 0 \\ &= x \times y \end{aligned}$$

となる．□

- 前者関数 $pd(x) = \begin{cases} 0 & (x = 0 \text{ のとき}) \\ x-1 & (x > 0 \text{ のとき}) \end{cases}$ は

$$\begin{cases} pd(0) = 0 \\ pd(x+1) = x \end{cases}$$

と単純に原始帰納的であることが分かる．□

- 自然数上の減算 $x \dot{-} y = \begin{cases} x-y & (x \geq 0 \text{ のとき}) \\ 0 & (x < 0 \text{ のとき}) \end{cases}$ も

$$\begin{cases} x \dot{-} 0 = x \\ x \dot{-} y + 1 = pd(x-y) \end{cases}$$

と原始帰納的であることが示せる．ここで前者関数 pd は原始帰納的であるので、自然数上の減算の定義として使用できた．□

- x の符号関数 $sg(x) = \begin{cases} 0 & (x = 0 \text{ のとき}) \\ 1 & (x > 0 \text{ のとき}) \end{cases}$ は

$$\begin{cases} sg(0) = 0 \\ sg(x+1) = 1 \end{cases}$$

と原始帰納的である．□

- 最小値を求める関数 $min(x, y) = \begin{cases} x & (x < y \text{ のとき}) \\ y & (x \geq y \text{ のとき}) \end{cases}$ は

$$min(x, y) = y \dot{-} (y \dot{-} x)$$

と書けるので原始帰納的である．ここで $\dot{-}$ は原始帰納的であることは示してある．同様に $plus$、$times$ と前者関数 sg を用いて

$$min(x, y) = plus(times(sg(y \dot{-} x), x), times(sg(x \dot{-} y), y))$$

と示すこともできる．□

- 最大値を求める関数 $max(x, y) = \begin{cases} x & (x \geq y \text{ のとき}) \\ y & (x < y \text{ のとき}) \end{cases}$ は

$$max(x, y) = plus(x, y) \dot{-} min(x, y)$$

と書けるので原始帰納的である．ここで $plus$、$\dot{-}$、min は原始帰納的であることは示してある．同様に $times$ と前者関数 sg を用いて

$$max(x, y) = plus(times(sg(y \dot{-} x), y), times(sg(x \dot{-} y), x))$$

と示すこともできる．□

2.1.3 原始帰納的な述語

引数に応じて真／偽を返すものを**述語 (predicate)** という．もう少し厳密に言えば、$\mathbf{N^n}$ から真、偽への関数と表現される．$\mathbf{N^n}$ の述語 P に対して、

$$f(x_1, \cdots, x_n) = \begin{cases} 0 & (P(x_1, \cdots, x_n) \text{ のとき}) \\ 1 & (\neg P(x_1, \cdots, x_n) \text{ のとき}) \end{cases} \tag{8}$$

で定まる関数 f を述語 P の**特性関数**あるいは**特徴関数 (characteristic function)** という．この特性関数が原始機能的である述語を**原始帰納的述語 (primitive recursive predicate)**、帰納的である述語を**帰納的述語 (recursive function)** という．

ここで述語 $P(x_1, \cdots, x_n)$、$Q(x_1, \cdots, x_n)$ を原始帰納的述語とすると、上述からそれぞれの述語の特性関数 $f(x_1, \cdots, x_n)$、$g(x_1, \cdots, x_n)$ も原始帰納的関数といえる．ここで述語 P、Q を用いて次のような論理を考える．

$$\neg P(x_1, \cdots, x_n)$$
$$P(x_1, \cdots, x_n) \vee Q(x_1, \cdots, x_n)$$
$$P(x_1, \cdots, x_n) \wedge Q(x_1, \cdots, x_n)$$

これらを特性関数で書くと

$$1 \dot{-} f(x_1, \cdots, x_n)$$
$$f(x_1, \cdots, x_n) \times g(x_1, \cdots, x_n)$$
$$f(x_1, \cdots, x_n) + g(x_1, \cdots, x_n) \dot{-} f(x_1, \cdots, x_n) \times g(x_1, \cdots, x_n)$$

とそれぞれが原始帰納的であるので、以上の論理は原始帰納的述語であるといえる．このように論理も述語や特性関数を使って、帰納的であることを示せるのである．

2.1.4 帰納的関数

ここでは飛ばしていた帰納的関数の定義について考える．帰納的関数とは初期関数に帰納的定義、合成と更に最小化の定義を満足したものである．

最小化の定義について考えよう．原始帰納的述語 P に対し、$\mu y P(x_1, \cdots, x_n, y)$ とは述語 P が真となる最小の y を意味している．難しく書けば

$$\mu y P(x_1, \cdots, x_n, y) = \begin{cases} min\{y | P(x_1, \cdots, x_n, y)\} & (\exists y) P(x_1, \cdots, x_n, y) \text{ のとき} \\ \text{無定義} & \neg(\exists y) P(x_1, \cdots, x_n, y) \text{ のとき} \end{cases} \tag{9}$$

ということになる．このとき述語 $P(x_1, \cdots, x_n, y)$ が真にするような y の値が少なくとも 1 つ存在することが分かっていれば、$\mu y P(x_1, \cdots, x_n, y)$ で y は取り出せる．この μ を **μ 作用素 (μ-operator)** という．

また、有界の範囲で行うこともある．この場合は

$$(\mu y)_{<z} P(x_1, \cdots, x_n, y) = \begin{cases} min\{y | y < z \wedge P(x_1, \cdots, x_n, y)\} & (\exists y)_{<z} P(x_1, \cdots, x_n, y) \text{ のとき} \\ z & \text{それ以外のとき} \end{cases} \tag{10}$$

と書くことができる．この $(\mu y)_{<z}$ を**有界 μ 作用素 (bounded μ-operator)** という．

μ 作用素はプログラムの while 文に、有界 μ 作用素は for 文に対応している．

最小化のところの文をもう一度、書き出すと、

- **最小化 (正則最小化)**
  $\mu y P(x_1, \cdots, x_n, y)$ は $P(x_1, \cdots, x_n, y)$ が真となる最小の y を意味する．これにより、$n+1$ 変数述語 P から n 変数関数 f を作る．ただし、P は任意の $x_1, \cdots, x_n$ に対して、$P(x_1, \cdots, x_n, y)$ が真となる y が必ず存在する．

$$\begin{aligned} f(x_1, \cdots, x_n) &= \mu y P(x_1, \cdots, x_n, y) \\ &= \mu y (g(x_1, \cdots, x_n, y) = 0) \end{aligned}$$

前半はこの節で解説した μ 作用素の定義、そして後半の部分が最小化を定義している．ここで述語 P は任意の $x_1, \cdots, x_n$ に対して、$P(x_1, \cdots, x_n, y)$ が真となる（これは転じて言えば、述語 P の特性関数 $g(x_1, \cdots, x_n, y) = 0$ ということ）y が必ず存在するとき、**正則**であるという．同様に関数 $f(x_1, \cdots, x_n, y)$ は任意の $(x_1, \cdots, x_n)$ に対して、$f(x_1, \cdots, x_n, y) = 0$ なる y が存在するとき、**正則**であるという．以上のことを踏まえ、初期関数と合成、原始帰納以外の関数から関数を作る操作が、上記に示す最小化なのである．なお特に、この場合の最小化は述語 P（もしくは特性関数 g）が正則であることが保証されているので、**正則最小化**という．このとき得られる関数 f は 全域的である ことが保証される．

これに対して、一般に正則が成り立たなくても最小化は定義される．このとき得られる関数は（たとえ$n+1$関数が全域的であっても）全域的であるとは限らない．ゆえに得られた関数は**部分帰納的関数 (partial recursive function)** といわれる．

以後、原始帰納的ではないかが帰納的な関数などの議論もあるが、ここではこれ以上深入りしない．

2.2 チューリング機械による計算過程の表現 [5]

Turing による抽象機械は、1936 年に発表されたのだが、それは機械的に見て、今日のコンピュータによく似ている．彼は人間による計算過程を克明に捉え、それを見事に抽象化している．ここでは彼が開発したチューリング機械を紹介し、チューリング機械が計算過程を見事に表現していることを紹介したい．そして実は、チューリング機械で計算可能な関数こそ、帰納的関数であることを示したいと思う．

2.2.1 チューリング機械とは

Turing の抽象機械の概念は、人間の計算過程を克明に観察した結果、考えられたものである．彼は人間が扱う数字を 1 つの計算記号とみなし、計算を行う過程はその計算記号を次々と変化させていくものと捉えたのである．それはあたかも長いテープに刻み込まれた 1 つ 1 つの状態を遷移させていくものとし、計算者はそのテープの状態を読み書きして動かすヘッドと、状態を記憶する制御部の役割があると考えたのである．これが**チューリング機械 (Turing Machine または略して TM)** の原理であるのだ．

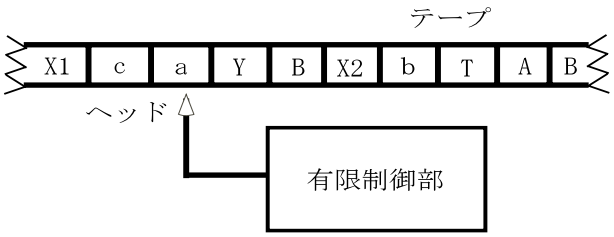


図 2: チューリング機械

以上の概念をもう少し正確に示そう．まず、チューリング機械は次の 3 つの部分からなる．

1. **有限制御部**
   チューリング機械の現在の状態を記憶する．ただし、ここでは高々有限個の記憶しかできない．
2. **1 つのテープ（両無限）**
   マス目 1 つ 1 つに記号が書き込める．何も書き込まれてない部分は空白記号 B が入る．
3. **読み書きできるヘッド**
   ヘッドはテープ上の 1 つのマス目に置かれ、その部分の記号を読み書きできる．また、左右に 1 マスずつ移動できる（動かさなくてもいい）．

これを模式的に書いたのが図 2 である．実際に行われる計算はチューリング機械がテープに読み書きしながら、動作することで表現できるのである．

それでは更に数学的に表現するために、以下のようなものをチューリング機械と定義しよう．

1. K：空でない有限集合で、要素は**状態**である．

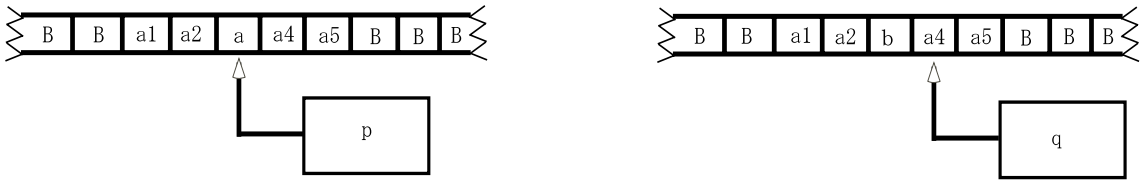
2. Σ：空でない有限集合で、要素は**記号**である．なお、空白記号 B も含まれる．

3. q_S：**初期状態**を示す．$q_S \in K$ である．

4. q_H：**停止状態**を示す．$q_H \subset K$ である．

5. $\delta : (K - q_H) \times \Sigma \to \Sigma \times \{R, L, N\} \times K$ なる部分関数で**遷移関数**という．ここで $\{R, L, N\}$ はテープを { 右、左、不動 } に動かす動作を表している．

このような 5 つ組 $M = (K, \Sigma, \sigma, q_0, H)$ をチューリング機械とする．



a ）時点表示 α　　b ）時点表示 β

図 3: チューリング機械の遷移

ここで今、テープ上の記号が図 3 のように $\cdots B, a_1, a_2, a, a_4, a_5, B \cdots$ であるとする．このとき図 a) のようにヘッドが a の位置にあるとし、有限制御部の内部状態が p であるときを

$$a_1 a_2 p a a_4 a_5 \tag{11}$$

というように書く．このような表現を M の**時点表示 (instantaneous description)** という．今、この時点表示を α とし、チューリング機械 M の動作によって

$$a_1 a_2 b q a_4 a_5 \tag{12}$$

に移ったとする．この時点表示を β とする．このとき遷移関数は $\delta(p, a) =< b, R, q >$ と表現できる．これは α のとき制御部の内部状態 p、ヘッドが参照しているコマの状態 a が、この遷移関数によって制御部の内部状態 q、参照していたコマの内部状態 b に変えて、ヘッドが右に動いたことを表している．これは図 3 を見れば、一目瞭然であろう．このようにチューリング機械 M によって、計算状況が α から β に変わったことを

$$\alpha \quad \vdash_M \beta \tag{13}$$

と書く．このような例からも分かるように、数学的に定まったチューリング機械 M は関数と等価であることが分かる．すなわち写像 M を繰り返しながら、計算を行い、最終的な値を求めようとするわけである．

2.2.2 チューリング機械における計算

チューリング機械がどういうものか分かったところで、次にチューリング機械における計算処理を具体的に見ていこう．まず、計算を行うにはチューリング機械におけるテープに載せる符号化を考えねばならない．この符号化は計算機を実装する段階で、計算機にあったものを選ぶ必要がある．私たちが普段用いるコンピュータは、0 と 1 のデジタル情報を扱っている．今、仮にテープ上では B と 1 しか載せることはできない (つまり、入力記号が B と 1 である) とする．すると自然数 $x (x \in \mathbf{N})$ に対し、

$$\overline{x} = 11 \cdots 1 (x + 1 \text{ 個}) = 1^{x+1} \tag{14}$$

と符号化することができる．これを**単項コード (単項記法)** という．$\overline{0}=1$、$\overline{1}=11$、$\overline{2}=111$、…である．また一般に、自然数 n 組 $(x_1,\cdots,x_n)\in\mathbf{N}^n$ のコード化は

$$\overline{(x_1,\cdots,x_n)}=\overline{x_1}B\overline{x_2}B\cdots B\overline{x_n} \tag{15}$$

で定義される．

それではどのような関数 f がチューリング機械で計算可能なのかを考える．今、関数 f の定義域を $\mathcal{D}(f)$ で表したとき、関数 $f(x_1,\cdots,x_n)$ は、

1. 任意の $(x_1,\cdots,x_n)\in\mathcal{D}(f)$ に対し、

$$\overline{(x_1,\cdots,x_n)}q_SB\vdash_M^*\overline{(x_1,\cdots,x_n,f(x_1,\cdots,x_n))}q_HB \tag{16}$$

が成り立つ．

2. $(x_1,\cdots,x_n)\notin\mathcal{D}(f)$ のとき、$\overline{(x_1,\cdots,x_n)}q_SB$ から始まる M の計算は存在しない．

を満たすとき、チューリング機械 M は関数 f を**部分的に計算する**という．また、f が全域的で部分的に計算可能なとき、f は**計算可能である**といい、そのときチューリング機械 M は関数 f を計算するという．なお、上で示した $\vdash_M^*$ は $\alpha\vdash_M\beta$ の処理において、チューリング機械 M の時点表示の列 $\alpha_1,\alpha_2,\cdots,\alpha_k$ が存在して、

$$\alpha=\alpha_1\vdash_M\alpha_2,\cdots,\alpha_{k-1}\vdash_M\alpha_k=\beta \tag{17}$$

が成立することを示している．この $\vdash_M^*$ を反射的推移閉包という．

例：後者関数 $S(x)=x+1$ を計算するチューリング機械 M を考える．このとき、

$$\overline{x}q_SB\vdash_M^*\overline{x}B\overline{x+1}q_HB$$

となるような M を作れば良い．そのような M の1つとして、次のようなものが考えられる．

$$K=\{q_S,q_1,q_2,q_3,q_4,q_5,q_6,q_7,q_H\}$$

$$\Sigma=\{B,1\}$$

$$Q=\left\{\begin{matrix}q_SBBLq_1, & q_11BRq_2, & q_1BBRq_6, & q_211Rq_2,\\ q_2BBRq_3, & q_311Rq_3, & q_3B1Lq_4, & q_411Lq_4,\\ q_4BBLq_5, & q_511Lq_5, & q_5B1Lq_1, & q_611Rq_6,\\ q_6BBRq_7, & q_711Rq_7, & q_7B1Rq_H, & \end{matrix}\right\}$$

ここで集合 Q は遷移関数 δ の代わりのものであり、例えば $\delta(q_S,B)=<B,L,q_1>$ を

$$Q=\{q_SBBLq_1\mid\delta(q_S,B)=<B,L,q_1>\}$$

と表したものである．

以上を使えば、後者関数 $S(x)$ をチューリング機械 M を使って計算できる．例えば、S(2) の計算は

$$111q_0B\vdash_M^*B111B1111q_HB$$

を実行するものであり、別紙1のような動作によって実現できる．□

2.2.3 チューリング機械の結合

ここではもう少し複雑なチューリング機械を考える．その前にチューリング機械 M の結合について、述べておく．チューリング機械 M が計算を終えた状態（または、計算できずある状態に留まる）を M の**停止状態** q_H という．そして 2 つのチューリング機械 M、M' の結合を次のように定義する．

1. M が停止したとき、M' の初期状態 q'_S と M の停止状態 q_H が同一視できるとき、M の最後の時点表示から M' の計算を求めるチューリング機械を、M と M' の**結合**といい、

   $$MM'$$

   で表す．

2. M の停止状態が 2 つ以上ある場合を考える．例えば、停止状態が 2 つある場合、それぞれの停止状態が q_{H1}、q_{H2} と表せるとする．そして、M のそれぞれの停止状態が M' と M'' という 2 つのチューリング機械の初期状態と同一視できるとき、以下の図 4 のように結合を表現できる．以下、停止状態が複数ある場合も同じように定義できる．これを**ダイヤグラム**という．

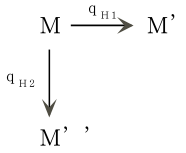


図 4: ダイヤグラム

一般に計算を行うチューリング機械は、単純な動作をするチューリング機械の結合という形で表現できる．基本的なチューリング機械のリストを別紙 2 [5] に示す．以下では、この基本的なチューリング機械を用いて、計算はどのようなチューリング機械を用いれば計算できるのかを考えるために、いくつかの例題を取り上げよう．

- 定数関数 $C_3^n(x_1, \cdots, x_n) = 3$ は

  $$M_1 = r1r1r1r1r = (r1^4)r$$

  でチューリング計算可能である．□

- 後者関数 $S(x) = x + 1$ は

  $$M_2 = K1r$$

  でチューリング計算可能である．例えば、前節の $S(2)$ の具体例を見てみると

  $$\begin{aligned} B111\underline{\mathrm{B}} &\vdash^* B111B111\underline{B}{:}\, K \\ &\vdash^* B111B111\underline{1}\,{:}\, 1 \\ &\vdash^* B111B1111\underline{B}\,{:}\, r \end{aligned}$$

  とチューリング機械は動いている（このときの下線はヘッドの位置を示している、以下同様）．□

- 射影 $I_k^n(x_1, \cdots, x_n) = x_k$ は

  $$M_3 = K_{n-k+1}$$

  でチューリング計算可能である．□

- 和 $plus(x,y)=x+y$ は

$$M_4 = K_2^2 LlTlB$$

でチューリング計算可能である．このチューリング機械の動作は次の通りである．

$$\begin{aligned}
\bar{x}B\bar{y}\underline{\mathbf{B}} &\vdash^* \bar{x}B\bar{y}B\bar{x}B\bar{y}\underline{B} && : K_2^2\\
&\vdash^* \bar{x}B\bar{y}B\overline{x-1}\underline{1}B\bar{y}B && : Ll\\
&\vdash^* \bar{x}B\bar{y}B\overline{x-1}1\bar{y}\underline{B} && : T\\
&\vdash^* \bar{x}B\bar{y}B\overline{x-1}1\overline{y-1}\underline{B}B && : lB
\end{aligned}$$

となる．ここで

$$\overline{x-1}1\overline{y-1} = \underbrace{1\cdots1}_{x-1}\underbrace{1\cdots1}_{y} = \underbrace{1\cdots1}_{x+y-1} = \overline{x+y}$$

である．□

2.2.4 チューリング機械と計算可能性

様々な関数が、チューリング機械をよって計算できることが分かったと思う．そこで、ここではどのような関数がチューリング計算可能なのかを考える．結論から言ってしまえば、帰納的という概念と計算可能の概念は同様である．帰納的関数は計算可能であることを示すには、帰納的関数がチューリング機械で表現できることが必要である．よって、ここでは帰納的関数がチューリング機械で表現できることを示していく．なお、ここでは時間の都合上、それぞれの詳細な動作については省略する．

補題 1　初期関数　初期関数は計算可能である．

証明　すでに前節で初期関数について、計算を実行するチューリング機械は示してある．□

補題 2　合成　関数 $g(y_1,\cdots,y_r)$、$h_1(x_1,\cdots,x_n),\cdots,h_1(x_1,\cdots,x_n)$ が計算可能であれば、合成関数

$$f(x_1,\cdots,x_n) = g(h_1(x_1,\cdots,x_n),\cdots,h_1(x_1,\cdots,x_n)) \tag{18}$$

も計算可能である．

証明　関数 $g,h_1,\cdots,h_r$ を計算するチューリング機械を $G,H_1,\cdots,H_r$ とする．このとき関数 f を計算するチューリング機械は

$$r1rK_{n+1}^n L^n lB\mathcal{R}H_1K_{n+1}^n H_2\cdots K_{n+1}^n H_r K_{r+(r-1)n}K_{r+(r-2)n}\cdots K_r GC \tag{19}$$

で構成できる．□

補題 3　原始帰納　関数 $g(x_1,\cdots,x_n)$、$h(x_1,\cdots,x_n,y,z)$ が計算可能であれば、原始帰納によって定義される関数 f

$$\begin{aligned}
f(x_1,\cdots,x_n,0) &= g(x_1,\cdots,x_n)\\
f(x_1,\cdots,x_n,y+1) &= h(x_1,\cdots,x_n,y,f(x_1,\cdots,x_n,y))
\end{aligned} \tag{20}$$

も計算可能である．

図 5: 原始帰納によって定義される関数 f を計算するチューリング機械

証明　関数 g, h を計算するチューリング機械を G, H とする．このとき関数 f を計算するチューリング機械は図 5 のように構成できる．□

補題 4　正則最小化（全域最小化）　関数 $g(x_1, \cdots, x_n)$、$h(x_1, \cdots, x_n, y, z)$ が計算可能かつ正則であれば、μ 作用素によって定義される関数 f

$$f(x_1, \cdots, x_n) = \mu y(g(x_1, \cdots, x_n, y) = 0) \tag{21}$$

も計算可能である．

図 6: μ 作用素によって定義される関数 f を計算するチューリング機械

証明　関数 g を計算するチューリング機械を G とする．このとき関数 f を計算するチューリング機械は図 6 のように構成できる．□

以上のことから、チューリング計算可能関数全体のクラスと帰納的関数全体のクラスは一致するという結論を得ることができる．しかし、この結論は考えている関数を全域的関数だけに限定していることに注意が必要である．チューリング機械は入力系列によっては計算が停止しない場合もあるので、全域的でない関数（すなわち、部分関数）を計算しえるということができる．

この場合、帰納的関数の定義における全域最小化演算を、正則条件を撤廃した最小化演算で得られた部分帰納的関数全体のクラスが真の意味で計算可能であるといえる．よって結論としては、

定理　チューリング計算可能な部分関数全体のクラスは、部分帰納的関数全体のクラスと一致する．

といえる．

2.3　万能チューリング機械

n 引数の部分関数 f を計算するチューリング機械 M_f があったとき、M_f の仕様を符号 $\overline{M_f}$ で表現し、

$$U_n(\overline{M_f}, x_1, \cdots, x_n) = f(x_1, \cdots, x_n) \tag{22}$$

となる部分帰納的関数 U_n を**万能関数**という．この万能関数 U_n はどんな機械 M でも、その符号 $\overline{M}$ にしたがって、M が計算する関数を計算することができる．また、U_n は帰納的なので、同じ能力のチューリング

機械 U_n が存在することになる．このチューリング機械を**万能チューリング機械 (universal Turing machine)** という．

この万能チューリング機械の存在意義は次のようなところにある．前節まで定義してきたチューリング機械においては、あらかじめ計算手順がチューリング機械に組み込まれており、変更ができなかった（これは**布線論理 (wired logic)** とよばれる.）．しかし、実際の計算機においては自由にプログラムを入れ替えることが可能な**プログラム内臓方式 (stored program)** をとっている．これは言い換えれば、各々の計算を実行するチューリング機械 M を模倣できる動作をもつ、別のチューリング機械が別に存在していることを意味している．その模倣機械こそ、ここで定義した (22) 式で定義した万能チューリング機械なのであり、これによってチューリング機械はプログラム内臓方式の計算モデルとしても十分といえるのである．

2.4 Church の提唱

計算とは何かということを考えるために、帰納的関数からチューリング機械における計算というところまで考えてきた．次に「アルゴリズム」とは何かという問題を考えてみよう．

通常、**アルゴリズム (algorism)** とは十進法における算法・算術法を意味する．その語源は 780 年のアラビアの数学者、フワーリズミー（アル・クワリズミ）に帰していると言われるが、今日の数学では問題を解くための演算手順を示したもの（出典：百科事典マイペディア）を指す．

具体的にある問題がアルゴリズムを持つこととは、

一定の規則にしたがって、有限回の操作で機械的に解を求めることができるような手続き

と示すことができる．このことを捉えると今までの議論から、

- 帰納的関数はそれを計算する「アルゴリズム」をもつ
- チューリング機械はそれ自身「アルゴリズム」をもつ

と考えることができる．そこで 1936 年に、Church はアルゴリズムをもつ関数（述語）と帰納的関数（述語）を同一視しようという提唱をした．これを **Church の提唱**という．計算とは何かを捉える方法として今までの議論以外にも

1. Gödel や Kleene による帰納的関数
2. Church による λ 定義可能性
3. Turing によるチューリング機械
4. Post による Post 機械

などが提案されているが、これらはことごとく同一であることが知られている．よって、Church の提唱は

アルゴリズムをもつ関数（述語）は、すべてチューリング機械で計算可能である

と言い換えることができる．

2.5 決定問題

計算過程はチューリング機械によって実現できることは示した．しかし、我々は計算する上で常に考えるのは、計算が本当にできるのかという問題である．この章の最後に、チューリング機械における決定問題について少し触れておこう．

自然数上の述語 $P(x_1, \cdots, x_n)$ に対して、

任意の自然数 $x_1, \cdots, x_n \in \mathbf{N}$ に対して、$P(x_1, \cdots, x_n)$ が真となる（すなわち、成立する）か否かを決定するアルゴリズムを作る

という問題を述語 $P(x_1, \cdots, x_n)$ に対する**決定問題 (decsion problem)** という．このようなアルゴリズムが作れれば、Church の提唱により、述語 $P(x_1, \cdots, x_n)$ は帰納的となる．つまり、決定問題とは帰納的か否かを問う問題ともいえるのだ（集合 $S \subseteq \mathbf{N^n}$ に対しても同様の議論ができる）．

このような決定問題に対して、決定のアルゴリズムが作れるとき、**決定可能 (decidable)** または**可解 (solvable)** であるといい、そのようなアルゴリズムが存在しないとき、**決定不能 (undecidable)** または**非可解 (unsolvable)** であるという．非可解な決定問題の例として、代表的なものはチューリング機械の**停止問題 (halting problem)** がある．これは簡単にいえば、次の 2 点の問題である．

1. プログラムと入力を任意に与えたとき、その入力に対してプログラムが停止するかどうかを決定するアルゴリズムを作る．
2. プログラムは固定し、入力を任意に与えたとき、その入力に対してプログラムが停止するかどうかを決定するアルゴリズムを作る．

この 2 つの停止問題について、このどちらも非可解、つまりアルゴリズムは存在しないことが証明されている．

3 帰納的集合と帰納的可算集合

今までは、関数の計算可能性を中心にして考えてきた．しかし関数だけでなく、集合論一般に考えたときも帰納的定義は重要である．ここでは計算に密接した概念であって、後述する形式言語論を話をするうえで重要な概念について少し触れておく．

ここである有限集合 S を考える．今、S が

- $S \subseteq \mathbf{N}$
- S の特性関数が帰納的関数

であるとき、S を**帰納的集合 (recursive set)** という．別の表現をすれば、**N** の任意の集合に対して、それがその集合の要素かどうかを帰納的関数で判定できる集合が帰納的集合であるといえる．

また、上記集合 S に対して、$\mathrm{Im}(f) = S$ となるような部分帰納的関数 f が存在するとき、S は**帰納的可算集合 (recursively enumerable set)** という．ここでいう可算というのは、S と **N** との間に 1 対 1 写像が f が存在していることであり、「帰納的に可算」とは写像 f が帰納的関数であるということだ．したがって、f という「手続き」を使えば S の要素をすべて並べることができることを意味している．ここで f を S の**生成関数 (generating function)** または**枚挙関数 (enumerateing function)** という．f は帰納的関数であるので、帰納的可算集合は計算可能である．

4 形式言語論超入門

前章では計算 (computation) とは、どのような過程において実現するのかに焦点をあてて解説した．その上で取り扱ったのは、自然数上の関数や述語、集合に関するものであった．しかし、抽象的な数理言語という世界でも、明確なモデルが与えられている．以下に展開する形式言語論は私たちが日常用いる自然言語とは相容れない部分（複雑な面が多い）があるもの、コンピュータ処理の必要上比較的単純なコンピュータ用言語などとよい適応性を見せる．また Church の提唱にも見てとれるように、形式言語理論も広い意味の ”計算 ”を抽象している．ここでは超入門講座として、形式言語論を概観する．

4.1 Post の対応問題

次のような集合・記号を考える

- Σ：アルファベット（要素：記号）
- 語：記号を有限個並べたもの
- 空語 ε：記号を 0 個並べたもの
- $\Sigma^* = \Sigma$ の語全体
- $\Sigma^+ = \Sigma^* - \{\varepsilon\} = \Sigma$ の空語以外の語全体

この Σ を 2 個以上の記号を含むアルファベットとする．このとき α と β を Σ^+ の語の有限列、すなわち、

$$\alpha = x_1, \cdots, x_k, \beta = y_1, \cdots, y_k (x_i, y_i \in \Sigma^+) \tag{23}$$

としたとき、これを $P(\alpha, \beta)$ と表す．この α、β に対して、

$$x_{i_1}, \cdots, x_{i_n} = y_{i_1}, \cdots, y_{i_n} (n \geq 1, 1 \leq i_j \leq k) \tag{24}$$

となる添え字の列

$$i_1, \cdots, i_n \tag{25}$$

が存在するとき、$P(\alpha, \beta)$ は**解をもつ**といい、その添え字の列 $i_1, \cdots, i_n$ または $x_{i_1}, \cdots, x_{i_n}$ を $P(\alpha, \beta)$ の**解**という．

例：$\Sigma = \{a, b\}, \alpha = x_1, x_2, x_3, \beta = y_1, y_2, y_3$ を次の表で与える．

i	x_i	y_i
1	abb	ab
2	ab	bab
3	a	ba

このとき、

$$x_1x_2x_1x_3 = \underbrace{abb}_{x_1}\underbrace{ab}_{x_2}\underbrace{abb}_{x_1}\underbrace{a}_{x_3} = abbababba$$

$$y_1y_2y_1y_3 = \underbrace{ab}_{y_1}\underbrace{bab}_{y_2}\underbrace{ab}_{y_1}\underbrace{ba}_{y_3} = abbababba$$

となる．したがって、$P(\alpha, \beta)$ は解 1,2,1,3 をもつ．□

任意に与えられた α と β に対して、$P(\alpha, \beta)$ が解をもつか否かを決定する問題を、**Post の対応問題**という．この Post の対応問題は非可解なことが 1946 年に、Post によって証明されている．

4.2 様々な文法とその計算可能性

次のような集合・記号を考える．

- N：非終端アルファベットとよばれる空でない有限集合（要素：**非終端記号 (nonterminal symbol)**）
- Σ：終端アルファベットとよばれる空でない有限集合（要素：**終端記号 (terminal symbol)**）

- P：$N \times (N \cup \Sigma)$ の有限部分集合（要素 $A \to \alpha$：**生成規則 (production)**）
- **開始記号 (stard symbol)**S：$S \in N$

これら 4 つ組によって定義される

$$G = (N, \Sigma, P, S) \tag{26}$$

を**句構造文法 (phrase structure grammar)** または単に**文法**という．

句構造文法 $G = (N, \Sigma, P, S)$ に対して、$V = N \cup \Sigma$ とする．記号 $u, v \in V^*$ が

1. $u = xAy, v = x\alpha y (x, y, \alpha \in V^*, A \in N)$
2. $A \to \alpha \in P$

を満たすとき、$u \Rightarrow_G v$ とかく．これは直観的には、u に出現する（１つの）非終端記号 A を、句構造文法 G の生成規則 $A \to \alpha$ を用いて、α に置き換えることで、u を v に変換することを表している．

$w_0, w_1, \cdots, w_n \in V^*$ に対して、

$$w_0 \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n (n \geq 0)$$

となるとき、w_n は w_0 から**導出される (derived)** といい、$w_0 \Rightarrow_G^* w_n$ とかく．

開始記号 S より終端アルファベット Σ 上の文字列 w が句構造文法 G の生成規則によって導出されるとき、w を G **を生成する語**という．G **の生成する言語**を

$$L(G) = \{w \in \Sigma^* | S \Rightarrow_G^* w\} \tag{27}$$

とかく．

言語 $L \subseteq \Sigma^*$ に対して、$L = L(G)$ となるとき、G は L を**生成する**という．逆に、Σ 上の言語 $L \subseteq \Sigma^*$ に対して、L を生成する句構造文法が存在するとき、L を**句構造言語 (phrase structure language)** という．この句構造言語という概念は、チューリング機械で受理される語の集合という概念と同じである．また、句構造文法は帰納的可算言語族である．

句構造文法 $G = (N, \Sigma, P, S)$ に対して、制限が加わったものを以下のように定義する．

1. P のすべての生成規則 $A \to \alpha$ が、条件 $|A| \leq |\alpha|$ を満たすとき、つまり、A の長さが α を超えないとき、G を**文脈依存文法 (context-sensitive grammar)** といい、$L(G)$ を**文脈依存言語 (context-sensitive language)** という．
2. P のすべての生成規則 $A \to \alpha$ が、条件 $A \in N$ かつ $\alpha \in V^+$ を満たすとき、つまり、A は 1 つの変数であり、α は空語でないとき、G を**文脈自由文法 (context-free grammar)** といい、$L(G)$ を**文脈自由言語 (context-free language)** という．
3. P のすべての生成規則 $A \to \alpha$ が、条件 $A \in N$ かつ $\alpha \in (\Sigma \cup N\Sigma)$ を満たすとき、つまり、A は 1 つの変数であり、α は b または bB の形（ただし、$b \in \Sigma$、$B \in N$）、G を**正則文法 (regular grammar)** といい、$L(G)$ を**正則言語 (regular language)** という．

これら制限が加わった各文法は制限を加えた形のチューリング機械で受理可能である．文脈依存言語はヘッドが原則として入力が置かれているます目の範囲の外には動かない線形有界オートマトン、文脈自由言語はプッシュダウンスタックと呼ばれる特殊記憶メカニズムをもったプッシュダウンオートマトン、正則言語はヘッドは右に動くだけでテープには書きこまない有限オートマトンでそれぞれ受理可能である．これら

のオートマトンは、一般のチューリング機械に制限を加えた制限機械である．

例：G_1 を次のような文脈自由文法とする

$$G_1 = (\{S\}, \{a, b\}, \{S \to ab|aSb\}, S)$$

このとき、$n \geq 1$ に対して、

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow^* a^{n-1}Sb^{n-1} \Rightarrow a^n b^n$$

となる．したがって、

$$L(G_1) = \{a^n b^n | n \geq 1\}$$

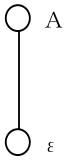
となる．□

文脈自由文法 $G = (N, \Sigma, P, S)$ に対して、**構文木 (parse tree)** というものが次のように帰納的に定義される．ここで $V = N \cup \Sigma$ とする．

1. $A \in V$ に対して、記号 A をラベルとする 1 つの頂点のみからなる木

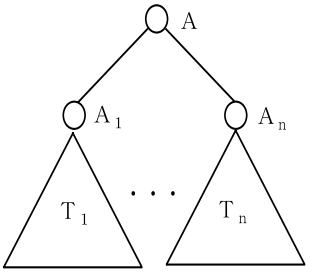
   は構文木である．

2. 生成規則 $A \to \varepsilon$ に対して、



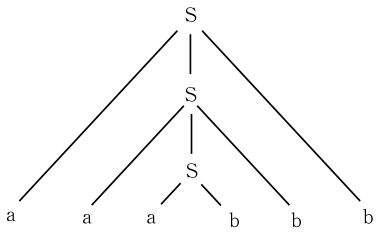
   は構文木である．

3. $T_1, \cdots, T_n$ を、根のラベルが $A_1, \cdots, A_n \in V$ となるような構文木とする．このとき、G の生成規則 $A \to A_1 \cdots A_n$ に対して、



   は構文木である．

例：上述の例の文脈自由文法 G_1 に対して



は構文木である．□

文脈自由文法 G は、ある $w \in L(G)$ に対して、2 個以上の異なる構文木が存在するとき、**曖昧である (ambiguous)** という．

5　まとめと今後の予定

今回は proposal の発表と、計算メカニズムについて概観した．計算のメカニズムとして取り上げたのは帰納的関数とチューリング機械であって、Church の提唱により（他の計算理論も含め)、同一視されることが分かった．また、次回以降の話の準備として、形式言語理論の超入門として、特に Post の対応問題と文脈自由文法について軽く触れた．この第 1 回目は次回以降の考察にも必要となる知識であるが、非常に簡単にまとめたため、必要事項があれば適宜ゼミで取り上げるようにしていきたい．

次回は具体的な自律計算への可能性を探るために、ナチュラル・コンピューティングの 1 つである分子計算について取り上げる．分子計算は 1994 年の Adlemann の実験以降、急速に広がっている新しい計算分野であるが、近年では自己組織化による新しい計算パラダイムも登場している．分子計算の広い視野から、具体的な話を進めていきたいと考える．

参考文献

[1] 清水博. 「操作情報を自己創出するシステム」. ホロニック技術研究会誌, Vol. 3, , 8 1995.

[2] 犬塚信博. 名古屋工業大学　知能情報システム学科　知能プログラミング入門講義資料「計算とプログラムの理論入門」. http://www-wada.elcom.nitech.ac.jp/inuzuka/Prog-th/.

[3] 平田耕一. 九州工業大学　情報工学部　知能情報工学科　計算理論講義資料. http://www.dumbo.ai.kyutech.ac.jp/hirata/lecture/.

[4] 細井勉. 「計算の基礎理論」, 新しい応用の数学, 第 9 巻. 教育出版, 4 1986. ISBN 4-316-37600-4.

[5] 田中尚夫. 「情報の数理　計算論理入門」. 裳華房, 10 1997. ISBN 4-7853-1505-9.