ラムダ計算と計算可能性

2011 年 1 月 31 日

6　帰納的関数

Turing 機械は計算可能性の最初のアプローチではない。その少し前に、Kleene は帰納的関数を計算可能性の定義として提案している。

6.1　原始帰納的関数

計算可能なものを定義しようとすると、できるだけ原始的なものから始めたい。原始帰納的関数は計算可能なものの全てではないが、面白い特徴がる。原始帰納的関数の計算は必ず止る。

定義 1 以下の *5* 規則から定義できる自然数上の関数は原始帰納的関数である。

1. $0() = 0$ という *0* 関数
2. $S(x) = x + 1$ という後者関数
3. $\pi_i^n(x_1, \ldots, x_n) = x_i$ という射影関数
4. $f : N^n \to N$ と $g_i : N^m \to N\ (1 \leq i \leq n)$ が原始帰納的関数なら、$f\langle g_1, \ldots, g_n\rangle(x_1, \ldots, x_m) = f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m))$ という f と (g_i) の合成
5. $g : N^n \to N$ と $h : N^{n+2} \to N$ が原始帰納的関数なら、

$$\left\{ \begin{array}{rcl} f(x_1, \ldots, x_n, 0) & = & g(x_1, \ldots, x_n) \\ f(x_1, \ldots, x_n, y+1) & = & h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) \end{array} \right.$$

という原始帰納法で定義された $f : N^{n+1} \to N$

例えば、足し算は以下のように定義できます。

$$\left\{ \begin{array}{rcl} +(x, 0) & = & \pi_1^1(x) \\ +(x, y+1) & = & h(x, y, +(x, y)) \\ h(x, y, z) & = & S(\pi_3^3(x, y, z)) \end{array} \right.$$

前者関数は、次のように定義できます。

$$\left\{ \begin{array}{rcl} P(0) & = & 0() \\ P(y+1) & = & h(y, f(y)) \\ h(y, z) & = & \pi_1^2(y, z) \end{array} \right.$$

そうすると引き算は合成によって定義できます。

$$\left\{ \begin{array}{rcl} -(x, 0) & = & \pi_1^1(x) \\ -(x, y+1) & = & h(x, y, -(x, y)) \\ h(x, y, z) & = & P(\pi_3^3(x, y, z)) \end{array} \right.$$

練習問題 1 $\times(x, y)$, $exp(x, y) = x^y$, $fact(x) = x!$, $max(x, y)$ を定義せよ

定理 1 全ての原始帰納的関数は有限時間で計算できる

補題 1 $g, h : N \to N$ が原始帰納的関数なら、

$$\begin{cases} f(x,0) &= g(x) \\ f(x,y+1) &= f(h(x),y) \end{cases}$$

によって定義された $f : N^2 \to N$ は原始帰納的関数である。

補題 2 $f : N^{n+1} \to N$ が原始帰納的関数ならば、限定最小解関数 $\mu_< f : N^{n+1} \to N$ は原始帰納的関数である

$$\mu_< f(x_1, \ldots, x_n, z) = \begin{cases} y & f(x_1, \ldots, x_n, y) = 0 \text{ なる最小の } y < z \\ z & f(x_1, \ldots, x_n, y) = 0 \text{ なる } y < z \text{ がなければ} \end{cases}$$

練習問題 2 限定最小解関数が原始帰納的関数であることを証明せよ

6.2 帰納的関数

Turing 機械の例から考えれば、計算が必ず止るという時点で、原始帰納的関数の表現力が弱いことが分かる。しかし、一つの新しい構成法を入れると、Turing 機械と同じ表現力が得られる。

定義 2 以下の構成法で得られた関数は帰納的関数である

1. 全ての原始帰納的関数
2. 帰納的関数と帰納的関数の合成
3. 帰納的関数の原始帰納法
4. $f : N^{n+1} \to N$ が帰納的関数ならば、その最小解関数 $\mu f : N^n \to N$

$$\mu f(x_1, \ldots, x_n) = \begin{cases} y & f(x_1, \ldots, x_n, y) = 0 \text{ なる最小の } y \\ \text{未定義} & f(x_1, \ldots, x_n, y) = 0 \text{ なる } y \text{ がなければ} \end{cases}$$

この定義で分かるように、帰納的関数がある n に対して未定義な場合があるので、帰納的関数を部分帰納的関数と呼ぶこともある。

定理 2 (Kleene 標準形) 全ての帰納的関数は一つの原始帰納的関数と一つの原始帰納的関数の最小解関数の合成で表せる。

証明 任意の帰納的関数 $f : N^n \to N$ について、$h(x_1, \ldots, x_n, y) = 0 \Rightarrow h(x_1, \ldots, x_n, y+1) = 0$ と $(\forall y)\ h(x_1, \ldots, x_n, y) = 0 \Rightarrow f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n, y)$ が成り立ち、しかも $f(x_1, \ldots, x_n)$ が定義されているときは $\mu h(x_1, \ldots, x_n)$ も定義されている、ような原始帰納的関数 $g, h : N^{n+1} \to n$ が存在することを帰納的関数の構成に関する帰納法で証明する。

原始帰納的関数について、$g(x_1, \ldots, x_n, y) = f(x_1, \ldots, x_n)$ でいい。

合成について、$f : N^m \to N$ と $f_i : N^n \to N$ に対し、$f' = f\langle f_1, \ldots, f_m \rangle$ を構築する。$h'(x_1, \ldots, x_n, y) = h(f_1(x_1, \ldots, x_n), \ldots, f_m(\ldots), y) + h_1(x_1, \ldots, x_n, y) + \ldots + h_m(x_1, \ldots, x_n, y)$ と $g'(x_1, \ldots, x_n, y) = g(g_1(x_1, \ldots, x_n, y), \ldots, g_m(x_1, \ldots, x_n, y), y)$ でいい。

最小解関数について、$f : N^{n+1} \to N$ に対し、$f' = \mu f$ を構築する。
$h'(x_1, \ldots, x_n, y) = \sum_{i=0}^{y-1} h(x_1, \ldots, x_n, i, y) + \mathrm{N}(y - \mu_{z<y} g(x_1, \ldots, x_n, z, y))$ と $g'(x_1, \ldots, x_n, y) = \mu_{z<y} g(x_1, \ldots, x_n, z, y)$ でいい。ここで $\mathrm{N}(0) = 1$, $\mathrm{N}(y+1) = 0$ と置く。

他の場合は簡単。 □

帰納的関数の構成法を与えたので、全ての帰納的関数のリストを作ることができる。1 引数の関数のリストを作り、そのリストの n 番目を f_n と呼ぶ。同様に、原始帰納的関数の n 番目を p_n と呼ぶ。$f_n(x)$ および $p_n(x)$ を n と x の帰納的関数として定義できる。

定理 3 完全帰納的関数で原始的でない関数がある。

証明 $F(x) = p_x(x) + 1$ は完全だが、どの p_n とも一致していない。 □

定理 4 どの原始帰納的関数よりも早く増加する完全帰納的関数がある。

定理 5 どの完全関数にも含まれない部分関数がある。
また、ある完全関数のリストを与えられれば、そのリストに含まれない完全関数が作れる。

証明 $f_x(x)$ が定義されている場合は $F(x) = f_x(x)$ とする。帰納的関数なので、ある関数 f_n とは一致しているが、$F(n)$ は未定義でなければならない。
T_x は完全関数のリストならば、$F(x) = T_x(x)$ はそのリストに含まれない。 □

6.3 帰納的関数と Turing 機械

定理 6 *Turing* 機械のステップ関数は原始帰納的関数である。

この定理を証明するために、Turing 機械の動きを自然数上の関数として定義しなければならない。簡単のために、記号が B を含めて二つしかないとする。

$\ldots$	b_2	b_1	b_0	s	c_0	c_1	c_2	$\ldots$

のテープ上に、ヘッドが s を指し、状態が q というのを 4 つの自然数で表現する

$$q, \quad s, \quad m = \sum_0^\infty b_i 2^i, \quad n = \sum_0^\infty c_i 2^i$$

まず、ステップ関数をこの 4 つの自然数を変える関数として定義する。例えば、右に移る場合を考える。$\delta(q, s) = (q', s', R)$ ならば、新しい 4 組は次のようになる。

$$q', \quad P(n), \quad s' + 2m, \quad H(n)$$

ここで、$P(n)$ と $H(n)$ は n の 2 の割り算の結果と余りである。すなわち、$P(n) \in \{0, 1\}$ かつ $P(n) + 2H(n) = n$。以下の原始的関数でいい。

$$\begin{cases} \mathrm{N}(0) = 1 \\ \mathrm{N}(y+1) = 0 \end{cases} \qquad \begin{cases} P(0) = 0 \\ P(y+1) = \mathrm{N}(P(y)) \end{cases} \qquad \begin{cases} H(0) = 0 \\ H(y+1) = H(y) + P(y) \end{cases}$$

実際に定義するために、以下の 3 つの関数を仮定する

$$\delta(q, s) = (q', s', d) \Leftrightarrow \begin{cases} Q(q, s) = q' & \text{状態} \\ R(q, s) = s' & \text{記号} \\ D(q, s) = d & \text{方向} \end{cases}$$

方向は 0 が右、1 が左とする。

そうするとステップ関数がこうなる

$$\begin{aligned} q^* &= Q(q,s) \\ s^* &= P(m) \times \overline{D}(q,s) + P(n) \times D(q,s) \\ m^* &= (2 \times m + R(q,s)) \times D(q,s) + H(m) \times \overline{D}(q,s) \\ n^* &= (2 \times n + R(q,s)) \times \overline{D}(q,s) + H(n) \times (q,s) \\ \overline{D}(q,s) &= 1 - D(q,s) \end{aligned}$$

定理 7 *Turing* 計算可能な関数は帰納的関数である。

ステップ関数を最小解関数で停止状態まで回せばいい。しかし、このままでは複数の引数の対して帰納法を使うことになる。帰納法に使う引数を一つにまとめる必要がある。

$Hd(Cons(x,y)) = x$ と $Tl(Cons(x,y)) = y$ となるような三つの関数を定義する。様々な方法は考えられるが、ここでは $Cons(x,y) = 2^x(2y+1)$ という符号化を選ぶ。

Hd と Tl はかなり複雑になる。まず偶数だけを割る関数 $J(x) = H(x) \times \mathrm{N}(P(x))$ を定義する。

2^n で割れるときは 1、割れないときは 0：$Div(x,0) = \mathrm{N}(\mathrm{N}(x)), Div(x,n+1) = Div(J(x),n)$

割れる回数を数えて、Hd を調べる：$Count(x,0) = 0, Count(x,n+1) = Div(x,n+1) + Count(x,n)$

そうすると、$Hd(x) = Count(x,x)$ と $Tl(x) = H^{Hd(x)+1}(x)$ になる。ただし Div と Tl の定義は補題 1 を利用している。

この三つの補助関数が定義されると、1 引数のステップ関数は以下のような形になる。

$$\begin{aligned} Step(x) = Cons(&q^*(Hd(x), Hd(Tl(x)), Hd(Tl(Tl(x))), Tl(Tl(Tl(x))), \\ &Cons(s^*(Hd(x),\ldots), Cons(m^*(Hd(x),\ldots), n^*(Hd(x),\ldots))))) \end{aligned}$$

n ステップ計算する原始的関数は $T_0(x,n) = Step^n(x)$

そして、停止状態が 0 だとすると、停止状態まで動かす帰納的関数は

$$\begin{aligned} T(x) &= T_0(x, \mu E(x)) \\ E(x,n) &= Hd(T_0(x,n)) \end{aligned}$$

定理 8 任意の帰納的関数を計算する *Turing* 機械がある

証明　任意の帰納的関数は λ 項として表現でき、かつ λ 項を正規化する Turing 機械が存在する。□