言語モデル論（２）

--- 言語と算法の基本性質と帰納的関数 その１ ---

4種のdomains

1. 計算モデルのdomain
2. プログラム言語のdomain
3. 算法（アルゴリズム）のdomain
4. 関数のdomain

注意

- あるアルゴリズムA
- あるプログラム言語L
- AをLで記述したプログラム$P_{A,L}$とする

この5つの性質に関するプログラム言語の対応する性質は、A と$P_{A,L}$を同一視して考えると分かりやすい。

算法の基本性質と
プログラム言語の理論的対応

- 実行可能な算法の族を特徴づける5つの性質と実用的なプログラム言語のもつ基本的な機能がおもしろい対応をしている。

- 以下は、数論的関数（自然数の関数）についての議論である。（しかし、記号列の自然な符号化やその逆変換が実行可能な手続きが知られているので、一般性は失われない。ゲーデルの算術化）

ゲーデルの算術化

- 算術化の方法の例:
  長さmの任意の文字列$x_1 \ldots x_i, \ldots x_m$ に対し

  2の$c(x_1)$乗 x
  3の$c(x_2)$乗 x
  … x
  （m番目の素数）の$c(x_m)$乗

という、正整数を対応させる。

但し、 $c(x_i)$は文字x_iに対応する正整数。
一意性は自然数の素因数分解の一意性で保証。

以下、実行可能なアルゴリズムの族を厳密に定義するために、その族の要素がどのような関数を計算するのかを、示すという方法をとる。

あるアルゴリズム（あるいはそれを記述したプログラム）Aがあると、Aが計算する関数F_Aを対象として議論する。

性質1（基本算法）

- （実行可能な）算法の族は、次のような数論的関数を評価する算法を含む。

(1)定数関数$C_m^{(n)}$(n変数関数で値が常にmであるもの、ただしm、nはそれぞれ任意に選んだ自然数)

(2)射影関数$P_i^{(n)}$(n変数関数でi番目の変数の値を、その値とするもの、ただしnは正整数で、$1 \leqq i \leqq n$)

(3)後者関数S、ただし $S(x) = x + 1$

- **通常のプログラム言語は、(1)自然数の定数が使える、(2)n個の列からI番目を取り出す、(3)数値に1を加える等の機能は持っている。**

性質2（算法の合成）

（実行可能な）算法の族が、m変数の関数hを評価する算法と、m個のn変数関数$g_1,g_2,\ldots,g_m$を評価する算法を含むならば、これを合成した関数 $h\circ(g_1,g_2,\ldots,g_m)$を評価する算法もその中に必ず含む

- 実用的なプログラミング言語においては、関数呼び出しやサブルーティン機能を有しているはずである。

性質3（万能算法）

仮定：プログラムは文字列だからindex化可能。 算法の族に対し、その中の各算法には自然数のindexが付されてる。

記法：指標 i を持つ算法が計算するn変数の関数を$F_i^{(n)}$とする。

（実行可能な）算法の族は、任意の正整数nに対して次のような性質をもつn+1変数の万能関数と呼ばれる関数$U^{(n+1)}$を評価する算法をその中に含む。

$$U^{(n+1)}(i,x_1,x_2,\ldots,x_n) = F_i^{(n)}(x_1,x_2,\ldots,x_n)$$

万能関数$U^{(n+1)}$の、指標 i とデータ$x_1,x_2,\ldots,x_n$に対する適用結果

i番目の算法に$x_1,x_2,\ldots,x_n$の引数を適用して得られる結果といつも同じになる。

- indexからプログラムに復号し、データを与え、プログラムを実行
- 通常のプログラム言語でプログラムのインタプリタが記述できる！！！

性質４（選択）

- （実行可能な）算法の族は、次のような4変数関数Cを評価する算法をその中に含む。

$$C(x,y,a,b) = \begin{cases} a & (x = y \text{ のとき}) \\ b & (x \neq y \text{ のとき}) \end{cases}$$

- 通常のプログラム言語には、与えられたデータや途中結果の値によって実行の流れを変える機能がある。

性質５（パラメータ化）

- （実行可能な）算法の族は、任意に選んだ正整数m,nに対して、次のようなm+1変数の全値関数$S_{n,m}$を評価する算法をその中に含む。（ただし等式は任意の正整数$i,x_1,x_2,\ldots,x_m,y_1,y_2,\ldots,y_n$に対して成り立つ。）

  $S_{n,m}(i,x_1,x_2,\ldots,x_m) = k$ として、

  $F_k^{(n)}(y_1,y_2,\ldots,y_n) = F_i^{(m+n)}(x_1,x_2,\ldots,x_m,y_1,y_2,\ldots,y_n)$が成立

- 算法の指標をプログラムとみなせば、プログラム***P*** とそれが使うデータ$(x_1,\ldots,x_m,y_1,..y_n)$の一部$(x_1,\ldots,x_m)$から新しいプログラム***P'***つくり、***P'*** に残りのデータ$(y_1,..,y_n)$を与えると、もとの***P***と同じ機能をもたすことができる。

そのような***P*** から***P'***へ変換する実行可能な算法がある。

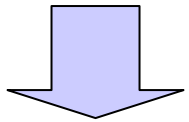
- 実行可能なアルゴリズムを記述できるほど強力なプログラム言語ならコンパイラや部分評価器が記述可能！

$$S_{m,n}(i, x_1, x_2, \ldots, x_m)$$

- $i, x_1, x_2, \ldots, x_m$はSに引数
- iは問題にしているプログラム（算法）に対応するインデックス
- m, nはSの添え字
- 部分評価器に対応する！！

Church's Thesis（チャーチの提唱)

- 算法とは、関数を計算する手続きと定義した。
- 性質１から5を満たす算法の族が、実際に我々が実行できる手続きの全体と「一致する」であることにしよう！！　という「提案・提唱」
- 証明可能な命題ではないので、「提唱」と呼ぶ。
- 新しい算法（アルゴリズム）の枠組みや記述体系が提案されても、どれも性質１から5を満たす。



- 算法（アルゴリズム）とは何かという問題と、その限界等に関する安定した概念。

帰納的関数
Recursive Functions

原始帰納的(primitive recursive)関数

定義 : 原始帰納的関数（primitive recursive function,略して原始的関数)という関数の族を次の(1)~(3)により定義する。

(1) 零関数　zero: $N^0 \to N$　ただし zero( ) = 0

後者関数　succ: $N \to N$　ただし $succ(x) = x+1$

射影関数 P_i^n: $N^n \to N$　ただし $P_i^n(x_1,x_2,\ldots,x_n) = x_i$,

$1 \leqq i \leqq n$ は原始的関数である。

(2) $g:N^m \to N$ と $g_j:N^n \to N$ $(j = 1,2,\ldots,m)$が原始的関数のとき

$$f(\vec{x}) = g(g_1(\vec{x}), g_2(\vec{x}), \ldots, g_m(\vec{x}))$$

で定義される関数$f:N_n \to N$は原始的関数である。

(3) $g:N^n \to N$ と $h:N^{n+2} \to N$　とが原始的関数のとき

$$f(\vec{x},0) = g(\vec{x}),\quad f(\vec{x},y+1) = h(\vec{x},y,f(\vec{x},y))$$

で定義される関数$f:N^{n+1} \to N$は原始的関数である。

ここで、(2)は関数の合成、(3)は関数の再帰的な定義である。

(3)の再帰の形を原始帰納法による関数定義とよぶ

原始関数の簡単な例

例：１変数関数　$pred(x) = x \dot{-} 1$

$pred(0) = 0 = zero(\ )$

$pred(x+1) = x = p_1^2(x, pred(x))$

例：自然数の足し算$add(x,y) = x+y$は原始的関数である。
実際、addは(3)の形の次の条件を満足する。

$add(x,0) = x,\ add(x,y+1) = suc(add(x,y))$

同様に引き算$sub(x,y) = x \dot{-} y$とかけ算$mult(x,y) = x \times y$もそれぞれ

$sub(x,0) = x,\ sub(x,y+1) = pred(sub(x,y)),$

$mult(x,0) = 0,\ mult(x,y+1) = add(x,mult(x,y))$

を満足するから、やはり原始的関数である。

原始関数の簡単な問題

問：$\exp(x,y) = x^y$, $\mathrm{fact}(x) = x!$, $\max(x,y)$, $\min(x,y)$ は原始的関数であることを示せ。

問：原始的関数$f:N\to N$に対して、$f^{\#}:N^2\to N$を次のように定める。

$$f^{\#}(x,y) = \overbrace{f(f(\ldots f}^{y\text{ 個}}(x)\ldots)).$$ すなわち、

$f^{\#}(x,0) = x$, $f^{\#}(x,1) = f(x)$, $f^{\#}(x,2) = f(f(x))$,….

とおく。このとき、$f^{\#}$は原始的関数であることを示せ

補題および原始関数・述語

補題：$f(\vec{x},y)$が原始的関数ならば、$f(\vec{x},0),f(\vec{x},1),\ldots,f(\vec{x},y-1)$の和$f'(x,y) = \sum_{z<y} f(\vec{x},z)$と積$f''(\vec{x},y) = \prod_{z<y} f(\vec{x},z)$も原始的関数である。

ただし、$f'(\vec{x},0) = 0$, $f''(\vec{x},0) = 1$とする。

原始的述語と特徴関数：

真または偽を値とする関数$p:N^n \rightarrow$ {真, 偽}を(n変数の)述語という。述語pに対してその特徴関数$c_p:N^n \rightarrow N$,

ただし

$$C_p(\vec{x}) = \begin{cases} 0 & (p(\vec{x}) = \text{真のとき}) \\ 1 & (p(\vec{x}) = \text{偽のとき}) \end{cases}$$

が原始的関数であるとき、pを原始的述語とよぶ。

原始関数・述語の例

例：述語「$x = 0$」(すなわち、xが0のとき、かつそのときのみ真である述語)は原始的述語である。なぜなら、その特徴関数

$$C_{=0}(x) = \begin{cases} 0 & (x = 0 \text{ のとき}) \\ 1 & (x > 0 \text{ のとき}) \end{cases}$$

は原始的関数$1 \dot{-} (1 \dot{-} x)$に等しい。

また述語「$x = y$」および「$x \leqq y$」の特徴関数をそれぞれ $C_{=}(x,y)$, $C_{\leqq}(x,y)$ とすると

$$C_{=}(x,y)=C_{=0}((x \dot{-} y)+(y \dot{-} x)), \quad C_{\leqq}(x,y)=C_{=0}(x \dot{-} y)$$

である。よって、これらの述語も原始的である。

補題

補題: $p(\vec{x})$, $q(\vec{x})$, $r(\vec{x},y)$ が原始的述語のとき、次の述語も原始的である。

(1) $\neg p(\vec{x})$ (p の否定)

(2) $p(\vec{x}) \wedge q(\vec{x})$ (p または q)

(3) $p(\vec{x}) \vee q(\vec{x})$ (p かつ q)

(4) $(\exists z < y)\, r(\vec{x},z)$
(yより小さいある自然数zに対して$r(\vec{x},z)$が成り立つ)

(5) $(\forall z < y)\, r(\vec{x},z)$
(yより小さい全ての自然数zに対して$r(\vec{x},z)$が成り立つ)

(6) $p(f_1(\vec{x}),f_2(\vec{x}),\ldots,f_n(\vec{x}))$
ただし$f_1,f_2,\ldots,f_n$は原始的関数

例

例：前スライドの補題より「$x < y$」や「$x \times y = z$」は原始的述語である。また、補題を繰り返し適用することにより

$$\mathrm{div}(x,y) \overset{\mathrm{def}}{\Longleftrightarrow} (\exists z < y+1)\,(x \times z = y)$$

$$\mathrm{prime}(x) \overset{\mathrm{def}}{\Longleftrightarrow}$$

$$(x > 1) \wedge (\neg(\exists u < x)(\exists v < x)(u \times v = x))$$

で定義される述語div, primeも原始的であることが分かる。(注：補題(4)と(6)より、rが原始的述語でfが原始的関数のとき、$(\exists z < f(\vec{x}))\ r(\vec{x},z)$は原始的述語である)