言語モデル論(5)
―λ計算 その2―

λ計算が帰納的関数を計算できることを示してゆきたい

- λ計算で数を表現する
- データ構造を表現する

数の表現

- A. Churchは、自然数nを関数 f の変数（または関数）x へのn回の適用として表現した。

$$0 \equiv \lambda f \lambda x x$$
$$1 \equiv \lambda f \lambda x (fx)$$
$$2 \equiv \lambda f \lambda x (f(fx))$$
$$\vdots$$
$$n \equiv \lambda f \lambda x (\underbrace{f(\cdots(fx)\cdots)}_{n個})$$

- この表現に合わせるとSUCC関数は:

$$SUCC \equiv \lambda n \lambda f \lambda x (f((nf)x))$$
$$(SUCC\ 3) \equiv (\lambda n \lambda f \lambda x (f((nf)x)) \lambda f \lambda x (f(f(fx))))$$
$$\overset{\beta}{\Rightarrow} \lambda f \lambda x (f((\lambda f \lambda x (f(f(fx)))f)x))$$
$$\overset{\beta}{\Rightarrow} \lambda f \lambda x (f(\lambda x (f(f(fx)))x))$$
$$\overset{\beta}{\Rightarrow} \lambda f \lambda x (f(f(f(f(x))))) \equiv 4$$

- この自然数の表現法に従うと、ある自然数nの表現をNとすると(Na)は $\lambda x(\underbrace{a(a\cdots(}_{n個}ax)\cdots))$という形である。これをさらにbに適用すれば((Na)b)は $(\underbrace{a(a\cdots(}_{n個}ab)\cdots))$という形になる。そこで別の自然数mの表現Mをaに適用したもの(Ma)すなわち $\lambda x(\underbrace{a(a\cdots(}_{m個}ax)\cdots))$を、((Na)b)に適用する。すると((Ma)((Na)b))の結果は $(\underbrace{a(a\cdots(}_{m個}a((Na)b))\cdots))$となり、さらに $(\underbrace{a(a\cdots(a(}_{m+n個}a\cdots(ab)\cdots))\cdots))$となる。これはaのn+m重のbに対する適用である。この観察から加算が次のように表現される。

諸演算の表現

問：以下の加算、乗算、べき乗の表現を確かめよ。

TIMES≡ λ m λ n λ f(m(nf))

PLUS≡ λ m λ n λ f λ x((mf)((nf)x))

EXP≡ λ m λ n(nm)　(mのn乗)

2つの対象*A,B*の対(pair)の表現

- 対、pair、cons(A,B)を　λ z((zA)B)で表現
  - CONS≡ λ x λ y λ z((zx)y)
  - (CAR <A,B>)≡(λ w(w(λ u λ vu)) λ z((zA)B))
    ⇒$_\beta$(λ z((zA)B) (λ u λ vu))
    ⇒$_\beta$((λ u λ vu A)B)
    ⇒$_\beta$(λ vA B) ⇒ A
- CAR≡ λ w(w(λ u λ vu)) CDR≡ λ w(w(λ u λ vv))

(CAR((CONS A)B)) ⇒$_\beta$ A

(CDR((CONS A)B)) ⇒$_\beta$ B

問：以上の表現や変換を確かめよ

条件式、論理演算子

- 真偽値: $T \equiv \lambda a \lambda b\, ba, \quad F \equiv \lambda a \lambda b\, bb$

- 条件式: (**if** P **then** M **else** N) $\rightarrow$ ((PM)N)
- 論理演算子: $AND \equiv \lambda a \lambda b((ab)F)$

  $OR \equiv \lambda a \lambda b((aT)b)$

  $NOT \equiv \lambda a((aF)T)$
- 問: $((TM)N) \overset{\beta}{\Rightarrow} M$, $((FM)N) \overset{\beta}{\Rightarrow} N$ を確かめよ。

継続(continuation)

- 継続：λ式Mの評価が終わった後、次ぎに何を実行するかの記述（λ式）Cを継続と呼ぶ。これは、Mへの引数として渡せばよい。この引数
- 継続の起動：　Mがその継続Cに引き継がせたい対象をその継続に渡すことにより、継続Cは起動される。（Cからの継続）
- 継続は、いろいろな言語において、その実行規則に従って定まる概念。

逐次実行を継続で表現
手続き型プログラムの関数型への変換

- L,M,Nの動作をこの順で行う（但しLの前に変数x,yが存在）を想定：

  x←0;　y←x+y;　x←2・y;

L：xの値を0にする、yは不変　　x←0;

M：yの値を（xの値を）＋（yのもとの値）にする、xは不変　y←x+y;

N：xの値をyの値の2倍にする、yは不変　　x←2・y;

- Lの継続はM以降の動作、Mの継続はN以降の動作。さらにNの継続はNの後に行う動作で、これをCとしておく。

- 継続Cはxとyの最新の値を引数として受け取ると仮定。すると、
- Nは、Mからxとyの（Mによる評価直後の）値を受け取り、Nを評価した直後のxとyの値をCに送り込む。よって、Nをλ式で表現するすると: $\lambda x \lambda y((C((TIMES\ \ 2)y))y)$ （((TIMES 2)y)：xの値）
- Cも実際は引数としてx,yより先に渡されるから、

$$N = \lambda c \lambda x \lambda y((c((TIMES\ 2)y))y)$$

が一般的な形である。同様にして、

$$L = \lambda c \lambda x \lambda y((c\ \ 0)y)$$

$$M = \lambda c \lambda x \lambda y((c\ \ x)((PLUS\ \ x)y))$$

さらに、L,M,Nを順次に実行することは、λ式として（L（M（NC））） となる。

継続を用いた動作列の順次実行の表現

- いま一般に、n個の操作$A_1,A_2,\cdots,A_n$が順々に行われ、各操作においうて次の操作に移るときにm個の変数$x_1,x_2,\cdots,x_m$の値が受け渡されるような操作全体をAとしよう。このとき、動作A_i $(1\leqq i\leqq n)$の直後に変数x_j $(1\leqq j\leqq m)$の値がz_jになるとすると、各A_iはλ-式で

$$A_i = \lambda c \lambda x_1 \cdots \lambda x_m (\cdots((cz_1)z_2)\cdots z_m)$$

のように表現される。さらにこの順次的な操作の全体Aは

$$(A_1(A_2(\cdots(A_nC)\cdots)))$$

となる。CはA_nの継続でもあり同時にA全体の継続でもある。

n回の繰り返しと「階乗」

- 自然数nの表現は、関数のn回の繰り返し適用。
- すると操作Bをp回繰り返した後、継続Cに移る操作全体は: ((p B) C) のように表現される。
- 例: nの階乗の計算の表現:

1. 変数 x, y の初期値をそれぞれ 1, n として、
2. 次に操作X,Y をこの順にn回繰り返す。

X:x を(xのもとの値の)×y にする、yは不変

Y:y を(yのもとの値の)ー1 にする、xは不変

- これに、継続を考慮すると、λ式は右のようになる:

$$X = \lambda c \lambda x \lambda y((c((TIMES\ \ x)y))y)$$

$$Y = \lambda c \lambda x \lambda y((c\ \ x)(PRED\ \ y))$$

繰り返しによる「階乗」の計算

- Y の継続を d とすると、X,Y を順次実行しd に移ることをλ式で表現すると、(X(Y d)) のようになる。
- d をパラメタとしでλ抽象し, λd (X(Y d)) としてX,Yをこの順にn回繰り返すことは:

$$((n\ \lambda d(X\ (Y\ d)))\ E) \qquad (*)$$

- EはX,Yをn回繰り返したのち変数x,yを受け取り、階乗値だけを持つxを取り出すには、Eはλxλyxの形が必要。
- (n λdλxλy((d((TIMES x)y))(PRED y))) λxλyx)

  x, yの初期値1,nに指定をすると:
- ((((n λdλxλy((d((TIMES x)y))(PRED y))) λxλyx)1)n)

  更にnを抽象して:

IFACT≡ λn((((n λdλxλy((d((TIMES x)y))(PRED y))) λxλyx)1)n)

条件付き繰り返し

- 無条件にn回繰り返すのではなく、ある条件P（述語）が満たされている間はAを実行し、最大n回おこない、途中でPが満たされなくなったら、Bを行う。

  いまAが、m個の変数$x_1,x_2,\cdots,x_m$の値を受け取り、m個の値を$z_1,z_2,\cdots,z_m$を計算して、それらを継続cに送る式、すなわち $cA = (\cdots((cz_1)z_2)\cdots z_m)$の形をしていると仮定すると、このとき条件付きの繰り返しは、

  $$(n\ \lambda c \lambda x_1 \cdots \lambda x_m((P(cA))B)$$ 上限つきサーチ

  のように書ける。（各自これを確かめよ、条件式の表現((P(cA))B)でPが真のとき(cA)、偽のときBが得られることに注意。）

回数に上限のない繰り返し

- 上の二つの繰り返しには、回数に上限があった。 この制限をなくしたい。　　最小解オペレータ
- すなわち、「条件Pが満たされている間はAを実行し、満たされなくなったら直ちにBを実行する」
  -- ***while P do A; B***
- これを表現するには、直前の例の λ 式においてn回繰り返す部分を無限回の適用を表すそれに書き換えればよさそう。すなわち、$(f\ (f\ (f\ (\ldots\ ))))$ を表すものに書き換えればよい。

既約形をもたない λ 式、不動点演算子

- 既約形を持たない例： $(\lambda x(xx)\ \lambda x(xx))$をXとすると、 β -規則が限りなく適用されて
  $X \overset{\beta}{\Rightarrow} X \overset{\beta}{\Rightarrow} X \overset{\beta}{\Rightarrow} \cdots$となる。
- 上のXの替わりに、$(\lambda x(F(xx))\ \lambda x(F(xx)))$
  を Z として、これを用いると、Z に β 規則が無限回適用される。
- Z における F を λ 抽象すると不動点演算子が得られ、
  $Z \overset{\beta}{\Rightarrow} (FZ) \overset{\beta}{\Rightarrow} (F(FZ)) \overset{\beta}{\Rightarrow} (F(F(FZ))) \overset{\beta}{\Rightarrow} \cdots$
  そこで　　$Y \equiv \lambda f(\lambda x(f(xx))\ \lambda x(f(xx)))$

これをFに適用させると $(YF) \equiv (F(YF))$となるので、Fの不動点を求める

再帰的な階乗プログラムの表現

- そこで、*n*の代わりに不動点演算子***Y***をもちいると、階乗の計算は次に λ 式で表現される。

$$(\mathrm{Y}\ \lambda\,\mathrm{c}\ \lambda\,\mathrm{x}_1 \cdots \lambda\,\mathrm{x}_m((\mathrm{P}(\mathrm{cA}))\mathrm{B}))$$

λ式における再帰的な表現

- 上の例のおける不動点演算子の役割は: ある手続きを表現するλ式の複製(copy)をそれ自身の継続(continuation)として埋め込むことにある。 -- これは再帰呼び出し・定義の特殊な形のひとつと考えられる。
- 一般に再帰表現（手続き）の実行においては、実行の任意の時点、実行している再帰表現（手続き）の複製が必要になる。
- nの階乗を再帰的に求める式を

  $$f \equiv \lambda n(((ISZERO\ n)1)((TIMES\ n)(f(PRED\ n))))$$

  とすると、f が等号の両側に出現するので、右辺のf は未定義はず。そこで f をλ抽象して、

  $$H = \lambda f \lambda n(((ISZERO\ n)1)((TIMES\ n)(f(PRED\ n))))$$

再帰的な階乗の表現

実際 f を λ -抽象して

H = λ f λ n(((ISZERO n)1)((TIMES n)(f(PRED n))))

のように定義し、このHの無限回の適用(H(H(H･･･)))が得られるように不動点演算子Yを用いればよい。

RFACT = (YH)

すると所望の再帰的な階乗の計算の λ -式による表現RFACTが得られる。

コンビネータ

- 自由変数を持たない λ 式をコンビネータと呼ぶ。

  一般に、自由変数をもたない λ -式は**組み合せ子**(combinator)と呼ばれる。例えば、恒等関数Iは組み合せ子として

  $$I = \lambda xx$$

  と表わされ、$Ix \Leftrightarrow_{\beta} x$である。2変数の関数の引数の順序を変える操作は、組み合せ子C

  $$C = \lambda f \lambda x \lambda y((fy)x)$$

  を用いる。実際、$(((Cf)x)y) \Leftrightarrow_{\beta} ((fy)x)$である。ここで、gという2変数関数が引数の順序を交換しても常に同じ値となるという性質を表現するのに、((gx)y) = ((gy)x)と書く代わりにCを用いると

  $$Cg = g$$

  のように束縛変数x,yを用いずに簡単に表現できる。また次のような組み合わせ子

  $$W = \lambda f \lambda x((fx)x)$$

  は、2変数の関数から二つの引数を同じものにすることによって1変数の関数を作り出す。

このように、組み合せ子によって関数の適用に関するパターンを変数を介さずに表現できる。その他、代表的な組み合せ子には次のようなものがある。

$S = \lambda f \lambda g \lambda x((fx)(gx))$

$K = \lambda x \lambda y x$ (定数関数を作る)

$B = \lambda f \lambda g \lambda x(f(gx))$ (関数の合成)

これらを用いて

$(S(KI)) \Leftrightarrow (BI)$

$((S(KS))K) \Leftrightarrow B$

$((SK)K) \Leftrightarrow I$

などが簡単に示せる。自然数の λ -式による表現は次のような組み合せ子で表現される

$Z_0 = (KI)$ (零の表現)

$Z_{n+1} = ((SB)Z_n)$ (n+1の表現)

コンビネータとλ計算

- λ計算でのベータ変換等は、コンビネータの間に対応する変換規則を導入することができる。（確かめよ）
- このときベータ規則に伴う名前の捕そくを回避することが（変数条件の要請）、コンビネータでは容易である。（変数がないから）
- 一方、コンビネータはλ計算が持つ直感的な理解の容易さを、失っているという欠点はある。
- SKコンビネータ機械がCambridge大で、１９７０年代後半に実際に設計され、稼動したことあり。

宿題

- λ計算をシミュレートするプログラムを Scheme言語で作成せよ。
- 階乗の計算が正しく実行されることを確かめよ
- 締め切り 2007.12.25
- 余力があれば、コンビネータの簡約器をつくり、λ計算での計算過程と対比・観察せよ。