λ計算

Jacques Garrigue, 2019 年 1 月 16 日

ドイツの論理学者 Schönfinkel が 1920 年代に作った論理コンビネータは元々論理の証明論のための道具だった。1930 年代に Curry と Church がそれをλ計算に発展させると、論理的な意味と同時に計算的な意味を持つようになった。その後、計算機科学との関係がだんだん深くなり、1950 年代からリスプから始まる関数型言語の重要な基礎となった。簡単な定義で表現力が強いので、一般的にプログラミング言語のモデルとして使われる。

1 λ計算の構文 (syntax)

定義 1.1 λ項 (λ-term) は次の三つの構文からできている。

$$\begin{array}{rcll} M & ::= & x & \text{変数 (variable)} \\ & | & \lambda x.M & \text{抽象 (abstraction)} \\ & | & M\ M & \text{適用 (application)} \end{array}$$

x は文脈 (環境) で定義されている値 (別のλ項) を指す。

$\lambda x.M$ は項 M の中に使われている変数 x を束縛する。関数として見れば $f = \lambda x.M$ という定義は、普段の $f(x) = M$。ただし、$\lambda x.M$ と書くとき、名前をつける必要がないという便利さがある。

$M_1\ M_2$ は関数の適用である。普段の $M_1(M_2)$ にあたるが、M_1 は名前 (変数) だけでなく、どのλ項でもよい。

上の文法を普段の数式と混ぜると、

$$f(x) = x + 1 \text{ のとき } f(2)$$

が 1 回で書ける。

$$(\lambda x.x + 1)\ 2$$

括弧の使い方　構文では括弧を書かなかったが、曖昧性を解決するのに適宜括弧を使う。

$$\begin{array}{rcl} \lambda x.M\ N & = & \lambda x.(M\ N) \\ \lambda x.\lambda y.M & = & \lambda x.(\lambda y.M) \\ M\ N\ P & = & ((M\ N)\ P) \end{array}$$

自由変数　$\lambda x.M$ では、M の中の全ての x の出現は「束縛されている」という。ある項 M の中で、x が使われながら、束縛されていないのであれば、x は M の自由変数である。M の自由変数の集合 $FV(M)$ は帰納的に次の 3 条で定義される。

$$\begin{array}{rcl} FV(x) & = & \{x\} \\ FV(\lambda x.M) & = & FV(M) \setminus \{x\} \\ FV(M\ N) & = & FV(M) \cup FV(N) \end{array}$$

代入　代入は自由変数の値を変える。$([N/x]M)$ は項 M の中に出現するすべての自由な x を N で置き換える。

$$\begin{array}{lcll} ([N/x]x) & = & N & \\ ([N/x]y) & = & y & \\ ([N/x]\lambda y.M) & = & \lambda y.([N/x]M) & y \notin FV(N) \\ ([N/x](M\ M')) & = & (([N/x]M)\ ([N/x]M')) & \end{array}$$

3 条目では、y は N の自由変数であってはならない。そのためには、次の α 変換が許される。z は N の自由変数でなければ、

$$\lambda y.N \leftrightarrow \lambda z.([z/y]N)$$

こういう束縛変数の改称は常に許される。

簡約規則

定義 1.2　λ 項、α 変換と次の β 簡約で構成される項書き換え系は λ 計算という。

$$((\lambda x.M)\ N) \to ([N/x]M)$$

例題 1.1 簡約の例

$$\begin{array}{ll} & (\lambda f.\lambda g.\lambda x.f\ x\ (g\ x))\ (\lambda x.\lambda y.x)\ (\lambda x.\lambda y.x) \\ \to & \lambda x.((\lambda x.\lambda y.x)\ x\ ((\lambda x.\lambda y.x)\ x)) \\ \to & \lambda x.((\lambda y.x)\ (\lambda y.x)) \\ \to & \lambda x.x \end{array}$$

$$\begin{array}{ll} & (\lambda x.(x\ x))\ (\lambda x.(x\ x)) \\ \to & (\lambda x.(x\ x))\ (\lambda x.(x\ x)) \\ \to & \ldots \end{array}$$

定理 1.1　λ 計算は**合流性**を持つ。$M \to \ldots \to N$ と $M \to \ldots \to P$ という二つの簡約列があれば、$N \to \ldots \to T$, $P \to \ldots \to T$ となるような T が存在する。

2　λ 計算は万能である

全てのプログラムは λ 計算で書ける。

自然数　チャーチによるエンコーディングがある (Church numeral)。

$$\begin{array}{lll} \mathsf{c}_n = \lambda f.\lambda x.(f\ \ldots(f\ x)\ldots) & & f \text{ を } n \text{ 回適用する} \\ \mathsf{c}_+ = \lambda m.\lambda n.\lambda f.\lambda x.(m\ f\ (n\ f\ x)) & & \text{加算} \\ \mathsf{c}_\times = \lambda m.\lambda n.\lambda f.(m\ (n\ f)) & & \text{積算} \end{array}$$

練習問題 2.1　*1.* $(\mathsf{c}_3\ (\lambda x.x\ x)\ y)$ の標準形を計算せよ。

2. $(\mathsf{c}_2\ (\lambda x.x\ x)\ \mathsf{s})$ の標準形を計算せよ。

3. 指数の λ 項を与えよ。

ブール環　次のエンコーディングを使う。

$$\mathsf{t} = \lambda x.\lambda y.x \qquad \mathsf{f} = \lambda x.\lambda y.y \qquad \mathsf{not} = \lambda b.\lambda x.\lambda y.(b\ y\ x)$$

数の区別も可能で、例えば次の項が引き数に数をもらい、真偽値を返す。

$$\mathsf{if0} = \lambda n.(n\ (\lambda x.\mathsf{f})\ \mathsf{t})$$

組　λ 計算では組が簡単に表現できる。

$$\mathsf{pair} = \lambda x.\lambda y.\lambda f.(f\ x\ y) \qquad \mathsf{fst} = \lambda p.(p\ \mathsf{t}) \qquad \mathsf{snd} = \lambda p.(p\ \mathsf{f})$$

計算して見ると、

$$\mathsf{fst}\ (\mathsf{pair}\ a\ b) \to \mathsf{pair}\ a\ b\ \mathsf{t} \to (\mathsf{t}\ a\ b) \to a$$

引き算　チャーチエンコーディングによる引き算は比較的に難しい。ここに一つの定義を与える。

$$\begin{array}{lcl} \mathsf{c}_- & = & \lambda m.\lambda n.(n\ \mathsf{p}\ m) \\ \mathsf{s} & = & \lambda n.\lambda f.\lambda x.(f\ (n\ f\ x)) \\ \mathsf{s'} & = & \lambda x.(\mathsf{pair}\ (\mathsf{snd}\ x)\ (\mathsf{s}\ (\mathsf{snd}\ x))) \\ \mathsf{p} & = & \lambda n.(\mathsf{fst}\ (n\ \mathsf{s'}\ (\mathsf{pair}\ \mathsf{c}_0\ \mathsf{c}_0))) \end{array}$$

s は足す 1 を計算し、p は引く 1 を計算する。足し算は既に定義したので、s はすぐ定義できるが、p のためには補助関数 s' を使う。

s' は自然数の組 (m, n) に対して $(n, m+1)$ を返す関数である。それを $(0, 0)$ に繰り返し適用すると、順番に $(0, 1)$, $(1, 2)$, $(2, 3)\ldots$ が得られる。第一要素がちょうど適用回数引く 1 という値になるので、それを返すと前者関数になる。

最後に、$m - n$ を計算するために、m に n 回 p を適用すればいい。よって、$m \geq n$ ならば、

$$\mathsf{c}_-\ \mathsf{c}_m\ \mathsf{c}_n \xrightarrow{*} \mathsf{c}_{m-n}$$

不動点演算子　再起的な関数を定義するのに、不動点演算子 Y が必要になる。その基本的な属性は、$(Y\ M)$ が $(M\ (Y\ M))$ に簡約できることである。

$$Y = (\lambda f.\lambda x.(x\ (f\ f\ x)))\ (\lambda f.\lambda x.(x\ (f\ f\ x)))$$

Y はループを回す回数が分からないとき必要になる。例えば、階数の再帰的な定義は次の通りである。

$$\begin{array}{lcl} 0! & = & 1 \\ n! & = & n \times (n-1)! \quad \text{if } n > 0 \end{array}$$

それを λ 計算に翻訳するとこうなる。

$$\mathsf{c}_! = \lambda n.\mathsf{if0}\ n\ \mathsf{c}_1\ (\mathsf{c}_\times\ n\ (\mathsf{c}_!\ (\mathsf{p}\ n)))$$

こういう再帰的な定義は ($\mathsf{c}_!$ は右にも出ている) λ 計算では直接にできないので、Y を使う。

$$\mathsf{c}_! = Y(\lambda f.\lambda n.\mathsf{if0}\ n\ \mathsf{c}_1\ (\mathsf{c}_\times\ n\ (f\ (\mathsf{p}\ n))))$$

$YM \to M(YM)$ ということを思い出せば、上の方程式が成り立つことが分かる。

$$\mathsf{c}_! \to (\lambda f.\lambda n.\mathsf{if0}\ n\ \mathsf{c}_1\ (\mathsf{c}_\times\ n\ (f\ (\mathsf{p}\ n))))\ \mathsf{c}_! \to \lambda n.\mathsf{if0}\ n\ \mathsf{c}_1\ (\mathsf{c}_\times\ n\ (\mathsf{c}_!\ (\mathsf{p}\ n)))$$

因みに、階乗関数では繰り返しの回数があらかじめ分かるので、Y を使わなくても定義できる。

$$\mathsf{c_!} = \lambda n.\mathsf{snd}\ (n\ (\lambda x.\mathsf{pair}\ (\mathsf{s}\ (\mathsf{fst}\ x))(\mathsf{c}_\times\ (\mathsf{fst}\ x)\ (\mathsf{snd}\ x)))\ (\mathsf{pair}\ \mathsf{c}_1\ \mathsf{c}_1))$$

しかし、不動点を使えば、止まるかどう分からない関数も定義できる。

$$Y\ (\lambda f.\lambda x.\mathsf{s}\ (f\ x))\ \mathsf{c}_1 \to \mathsf{s}\ (Y\ (\lambda f.\lambda x.\mathsf{s}\ (f\ x))\ \mathsf{c}_1) \to \mathsf{s}\ (\mathsf{s}\ (Y\ (\lambda f.\lambda x.\mathsf{s}\ (f\ x))\ \mathsf{c}_1)) \to \ldots$$

3 λ 項の実行

3.1 評価戦略

ラムダ計算は計算機械ではない。なぜかといえば、簡約という計算機構があっても、それをどの順番で行うか、あるいはいつ計算が終わるか、が定まっていない。標準形と戦略はちょうどそれを決める役割をはたしている。

標準形　簡約できる個所を全く残さない (既約) 標準形以外にも、先頭の部分だけが簡約できない弱冠頭標準形も定義できる。例えば $(x\ ((\lambda y.y)\ z))$ や $\lambda x.((\lambda y.y)\ z)$ は既約標準形でない (y がまだ簡約できる) が弱冠頭標準形である。形式化すると、$\lambda x.M$ または $(x\ M_1\ \ldots\ M_n)$ のどちらかの形をしたものが弱冠頭標準形である。

最左戦略　最も左にある簡約できる個所を先に簡約する。名前呼び出し (call-by-name) に当る。関数の引数を評価せずに、そのまま代入する。$(\lambda x.x)\ ((\lambda y.y)\ z) \to ((\lambda y.y)\ z)$。ある戦略で $M \to^* N(N$ 標準形) のとき、最左戦略でも $M \to^* N$。

最右最内戦略　最も右にある簡約できる個所の中で、最も中にあるものを選ぶ。値呼び出し (call-by-value) に当る。関数の引数を標準形に落してから代入する。$(\lambda x.x)\ ((\lambda y.y)\ z) \to ((\lambda x.x)\ z)$。ある戦略で $M \to^* \ldots$(無限に続く簡約) のとき、最右最内戦略でも無限な簡約になる。

3.2 変数名をなくす

λ 項を実行しやすい形で表現したい。特に変数名と α 変換は省きたい。そのために De Bruijn 添数を導入する。

定義 3.1 De Bruijn 添数　変数名を使わないラムダ計算を定義する。

$$M ::= n \mid \lambda M \mid (M\ M)$$

添数はこの変数が何番目の抽象 (中から数えて) で束縛されたかを表す。例えば $\lambda x.\lambda y.x = \lambda\lambda 2$, $\lambda x.(x\ x) = \lambda(1\ 1)$。代入を新しく定義する。

$\Uparrow_n k$	$= k+1$	$k \geq n$	$[N/n]n$	$= N$		
$\Uparrow_n k$	$= k$	$k < n$	$[N/n]k$	$= k-1$		$k > n$
$\Uparrow_n \lambda M$	$= \lambda(\Uparrow_{n+1} M)$		$[N/n]k$	$= k$		$k < n$
			$[N/n]\lambda M$	$= \lambda([\Uparrow_n N/n+1]M)$		
β簡約	$\lambda M\ N \to [N/1]M$		$[N/n](M\ M')$	$= ([N/n]M\ [N/n]M')$		

例題 3.1 対から 1 つ目の値を出す。

$$\mathit{fst}\ \lambda(1\ a\ b) = \lambda(1\ \lambda\lambda 2)\ \lambda(1\ a\ b) \to \lambda(1\ a\ b)\ \lambda\lambda 2 \to \lambda\lambda 2\ a\ b \to \lambda a\ b \to a$$