ラムダ計算と計算可能性

Jacques Garrigue, 2011 年 1 月 17 日

4 ラムダ計算

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。

4.1 項書換系

元々はそうではなかったが、λ 計算は項書換系によって定義される。計算を項の一部の書き換えとして見る、ということだ。例えば、数式の書換系を次のように定義できる。

項 (term) $E ::= \mathcal{R} \mid (E + E) \mid (E - E) \mid (E \times E) \mid (E/E)$

書換規則 x, y は $\mathcal{R}$ の実数ならば、

$$\begin{array}{lcl} (x+y) & \rightarrow & x+y \\ (x-y) & \rightarrow & x-y \\ (x\times y) & \rightarrow & x\times y \\ (x/y) & \rightarrow & x/y \end{array}$$

($(x+y)$ は数式だが、 $x+y$ は実数である)

上の規則を簡約規則ともいう。

簡約の例

$$(15 + (1/3)) \times (5-2) \rightarrow (15 + 0.3333) \times (5-2) \rightarrow (15.3333) \times 3 \rightarrow 46$$

4.2 λ 計算の構文 (syntax)

定義 4.1 λ 項 (λ-term) は次の三つの構文からできている。

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

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。

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

構文では括弧を書かなかったが、曖昧性のを解決するに適宜

自由変数と代入　$\lambda x.M$ 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 $FV(M)$ は再起的に次の三条で定義される。

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

代入はそういう自由変数の値を定義する。$([N/x]M)$ は項 M の中に出現するすべての自由な x を N と置き換える。

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

4 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。

4.3 簡約規則

定義 4.2 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 4.1 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 4.1 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。

4.4 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)。

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

練習問題 4.1 *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。

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

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。

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

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。

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

4.5 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。

λ 機械　標準形と戦略の組み合わせを選ぶと、任意の λ 項をある λ 機械によって決定的に実行されるプログラムとしてみることができる。

5 チューリング機械とラムダ計算の同値性

5.1 チューリング機械を λ 項に

組とリスト構造 既に 2 組の作り方を見ている。同様に n 組が作れる。$(a_1,\dots a_n) = \lambda f.(f\ a_1\ \dots\ a_n)$.

長さの分からないリストは 2 組で構成する。$[a_1,a_2,\dots,a_n]$ は
$(a_1,[a_2,\dots,a_n]) = \lambda f.(f\ a_1\ [a_2,\dots,a_n])$ になる。

規則性のある無限なデータ構造も無限リストとして表現できる。例えば、同じ値の繰り返し $[a,a,\dots]$ を $\mathsf{Y}\ (\lambda x.(\mathsf{pair}\ a\ x))$ と書ける。

状態と記号 $M = (K,\Sigma,q_0,H,\delta)$, $|K| = k$, $|\Sigma| = l$ とする。各状態と記号に番号をふっておき、$K = \{q_0,\dots,q_{k-1}\}$, $\Sigma = \{\sigma_0 = \mathrm{B},\dots,\sigma_{l-1}\}$。次の翻訳を定義する。

$$
\begin{aligned}
\overline{q_i} &= \lambda x_0\dots x_{k-1}.x_i\\
\overline{\sigma_i} &= \lambda x_0\dots x_{l-1}.x_i
\end{aligned}
$$

時点 時点 (T,n,q) を

$$(\overline{q},\overline{T(n)},[\overline{T(n-1)},\overline{T(n-2)},\dots],[\overline{T(n+1)},\overline{T(n+2)},\dots])$$

の 4 つ組で表す。無限リストを使うことになるが、B でしか構成されていない末端の部分を Y を使って表現する。$[\overline{\mathrm{B}},\dots] = Y(\lambda y.(\overline{\mathrm{B}},y)) \to^* (\overline{\mathrm{B}},Y(\lambda y.(\overline{\mathrm{B}},y))) \to^* \dots$

遷移関数 ある時点から次の時点への遷移関数を関数の l 組の k 組で表す。

$$\Delta = \lambda t.t((\overline{\delta}(q_0,\sigma_0),\overline{\delta}(q_0,\sigma_1),\dots,\overline{\delta}(q_0,\sigma_{l-1})),\dots,(\overline{\delta}(q_{k-1},\sigma_0),\dots))$$

$$
\overline{\delta}(q,\sigma) = \begin{cases}
\lambda t_l.\lambda t_r.f(\overline{q'},\mathsf{fst}\ t_l,\mathsf{snd}\ t_l,(\overline{\sigma'},t_r)) & \text{when } \delta(q,\sigma) = (\sigma',\text{左},q')\\
\lambda t_l.\lambda t_r.f(\overline{q'},\mathsf{fst}\ t_r,(\overline{\sigma'},t_l),\mathsf{snd}\ t_r) & \text{when } \delta(q,\sigma) = (\sigma',\text{右},q')\\
\lambda t_l.\lambda t_r.(\overline{q},\overline{\sigma},t_l,t_r) & \text{when } q \in H
\end{cases}
$$

この中の f を恒等関数 $\lambda x.x$ と定めると、抽象的に見る Δ の型は $(K,\Sigma,\Sigma\ list,\Sigma\ list) \to (K,\Sigma,\Sigma\ list,\Sigma\ list)$ であり、時点から時点への遷移関数に対応している。

実行 Δ は 1 回分の動作しか行わないので、その不動点を取ればよい。そのために f を抽象化する。

$$
\begin{aligned}
\lambda f.\Delta :\ & ((K,\Sigma,\Sigma\ list,\Sigma\ list) \to (K,\Sigma,\Sigma\ list,\Sigma\ list)) \to\\
& ((K,\Sigma,\Sigma\ list,\Sigma\ list) \to (K,\Sigma,\Sigma\ list,\Sigma\ list))
\end{aligned}
$$

$$(Y\ (\lambda f.\Delta)) : (K,\Sigma,\Sigma\ list,\Sigma\ list) \to (K,\Sigma,\Sigma\ list,\Sigma\ list)$$

それを初期状態に適用すると、$T \rhd (T',n,q')$ が算出される。しかし無限リストを使っているので、既約標準形まで計算すると止らない。弱冠頭標準形では正しい結果が得られる。

$$
\begin{gathered}
(Y\ (\lambda f.\Delta))\ (\overline{q_0},\overline{T(0)},[\overline{T(-1)},\dots],[\overline{T(1)},\dots]) \to^*\\
(\overline{q'},\overline{T'(n)},[\overline{T'(n-1)},\dots],[\overline{T'(n+1)},\dots])
\end{gathered}
$$

5.2　λ 項を実行できるチューリング機械

この節の目的は λ 項をテープに書き、実行することである。

テープを実行するチューリング機械を考える前に、λ 項を実行しやすい形でテープに書き込まなければならない。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 5.1 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。

$$\begin{array}{llll}
\Uparrow_n k & = & k+1 & k \geq n \\
\Uparrow_n k & = & k & k < n \\
\Uparrow_n \lambda M & = & \lambda(\Uparrow_{n+1} M) & \\
 & & & \\
\beta\text{簡約} & \lambda M\ N \to [N/1]M & &
\end{array}
\qquad
\begin{array}{llll}
[N/n]n & = & N & \\
[N/n]k & = & k-1 & k > n \\
[N/n]k & = & k & k < n \\
[N/n]\lambda M & = & \lambda([\Uparrow_n N/n+1]M) & \\
[N/n](M\ M') & = & ([N/n]M\ [N/n]M') &
\end{array}$$

例題 5.1 対から 1 つ目の値を出す。

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

テープの形式　5 つの記号 $\{0, 1, \mathrm{E}, \lambda, @\}$ を使い、テープへの変換 $\overline{M}$ を定義する。

$$\begin{array}{rcl}
\overline{n} & = & (n \text{ の 2 進法記述})\ \mathrm{E} \\
\overline{\lambda M} & = & \lambda\ \overline{M} \\
\overline{(M\ M')} & = & @\ \overline{M}\ \overline{M'}
\end{array}$$

チューリング機械 Λ　上のテープを実行する機械 Λ はここで定義しないが、加算等の簡単な動作でできるので、そういうチューリング機械の存在は明かであろう。ここで弱冠頭標準形を最左戦略で計算する。

5.3　万能チューリング機械

任意のチューリング機械を λ 項で表現できる、任意の λ 項はある特定のチューリング機械が実行するテープに変換できる。そういう二つの定理に面白い系がある。

系 5.1 任意のチューリング機械を模擬できる万能チューリング機械が存在する。その機械は模擬する機械の定義をテープで与えられ、それを実行する。

証明　模擬したいチューリング機械を λ 項に変換し、それを Λ に与えればよい。