型無しラムダ計算とモデル

星野直彦

1　導入：関数抽象と関数適用

この講義では自然数全体の集合 $\mathbb{N} = \{0, 1, 2, \cdots\}$ から $\mathbb{N}$ への関数などの他に関数を受け取って実数を返す汎関数など「何か」を受け取って「何か」を返すものは全て関数と呼ぶ。受け取る値や返す値が関数である関数は特に**高階関数**と呼ばれる。例えば

$$\mathtt{twice}(f) = f \circ f$$

は関数 $f : \mathbb{N} \to \mathbb{N}$ を受け取って関数 $f \circ f : \mathbb{N} \to \mathbb{N}$ を返す高階関数である。他には関数 $f : \mathbb{N} \to \mathbb{N}$ と自然数 x を受け取って自然数 $f(f(x))$ を返す高階関数

$$\mathtt{twice}'(f, x) = f(f(x))$$

が考えられる。各関数 $f : \mathbb{N} \to \mathbb{N}$ に対して関数 $\mathtt{twice}'(f, x)$ は x についての関数と見なせるが、この関数をラムダ計算では

$$\lambda x.\, \mathtt{twice}'(f, x)$$

と表現する。（この λ は形式的な記号である。）このように式をある変数に関する関数と見なす操作を**関数抽象**と呼び、逆に関数に値を渡して結果を得る操作を**関数適用**と呼ぶ。例えば自然数 x に関数 $\mathtt{twice}(f)$ を適用することで自然数 $f(f(x))$ を得るこの操作が関数適用である。

ラムダ計算は関数抽象と関数適用を形式化した体系であり、ラムダ計算の研究は「高階関数を用いた計算」の代数的な研究と言える。ラムダ計算は数学の基礎づけを与える研究の中で生まれたが、現在では関数型プログラミング言語の基礎理論としても研究されている。この講義の前半では Church の提唱の根拠の一つである型無しラムダ計算の表現力が再帰的関数と一致することを解説し、後半では型無しラムダ計算の数学的性質を調べる手法の一つである表示的意味論の研究について型付きラムダ計算との比較をしつつ解説する。最後に高階関数の計算可能性について触れる。

型無しラムダ計算と計算可能性の関係や型無しラムダ計算の表示的意味論については [1, 3] に詳しい解説がある。型付きラムダ計算の表示的意味論についての参考文献は [9] が挙げられる。

2　型無しラムダ計算

型無しラムダ計算は関数適用と関数抽象を形式的に表現するラムダ式とラムダ式に関する書き換え規則（もしくは等式理論）から成り立つ。2.1 章ではラムダ式の定義を行い、2.3 章でラムダ式に関する書き換え規則である β 変換の定義をする。

2.1 ラムダ式の定義

ラムダ式を定義するため変数記号の集合 Var を一つ選ぶ。Var は無限集合とする[1]。自然数や加減乗除等を行う関数、非決定的計算、継続計算などは考えず、関数適用と関数抽象のみからなる最もシンプルな型無しのラムダ式を与える。

定義 2.1. ラムダ式を以下で帰納的に定義する。

- 変数記号 $x, y, z, \cdots \in \mathrm{Var}$ はラムダ式。
- （関数適用）t と s がラムダ式のとき $(t \cdot s)$ はラムダ式。
- （関数抽象）t がラムダ式で x が変数記号のとき $(\lambda x.\, t)$ はラムダ式。

ラムダ式全体の集合を Λ と書く。

例えば x と y が変数記号の時、

$$(\lambda x.\, x) \qquad ((\lambda x.\, x) \cdot t) \qquad (\lambda x.\, (\lambda y.\, x)) \tag{1}$$

等はラムダ式である。直感的には左からそれぞれ恒等関数、「恒等関数にラムダ式 t を関数適用した値」、「ラムダ式 t を受け取ると、t を常に返す定数関数を返す関数」を意味している。恒等関数にラムダ式 t を関数適用すれば t が得られるはずだが、ラムダ式自体はただの記号列であり $((\lambda x.\, x) \cdot t)$ と t は区別される。ラムダ式の等しさは 2.3 章の β 変換により与えられる。

ラムダ式を定義に忠実にしたがって書いていると多くの括弧が現れ煩雑になるため次のような略記法を用いる。

$$\lambda x_1 x_2 \cdots x_n.\, t_1\, t_2\, t_3 \cdots t_m = (\lambda x_1.\, (\lambda x_2.\, \cdots (\lambda x_n.\, ((\cdots ((t_1 \cdot t_2) \cdot t_3) \cdots) \cdot t_m)) \cdots)) \tag{2}$$

この略記法を使うと (1) のラムダ式はそれぞれ

$$\lambda x.\, x \qquad (\lambda x.\, x)\, t \qquad \lambda x y.\, x$$

と書くことができる。ただしこの略記法を用いると $(\lambda x.\, (t_1 \cdot t_2))$ と $((\lambda x.\, t_1) \cdot t_2)$ はともに $\lambda x.\, t_1\, t_2$ と省略され曖昧になるので、$\lambda x_1 x_2 \cdots x_n.\, t_1\, t_2\, t_3 \cdots t_m$ と書いた場合は常に (2) の略記とし、それ以外の場合は $(\lambda x_1 \cdots x_n.\, t_1 \cdots t_i)\, t_{i+1} \cdots t_m$ の様に適宜括弧を補い曖昧性を回避する。

閉ラムダ式　変数記号の有限列とラムダ式の間の二項関係 $\Gamma \vdash t$ をラムダ式の構成に関する帰納法で定義する。

- $x_1, \cdots, x_n \vdash x_i$ である。
- $\Gamma \vdash t$ と $\Gamma \vdash s$ なら $\Gamma \vdash t\, s$ である。
- $\Gamma, x \vdash t$ なら $\Gamma \vdash \lambda x.\, t$ である。

この二項関係を**判定**と呼ぶ。変数記号の有限列 Γ が空のときは単に $\vdash t$ と書き、このようなラムダ式 t を**閉ラムダ式**と呼ぶ。

[1] Var は無限集合であればどのようなものを選んでも議論に本質的な影響はない。

2.2 α 同値関係と代入

積分 $\int_0^1 (x+1)\,dx$ の中の変数 x に特別な意味はなく $\int_0^1 (y+1)\,dy$ と書いても意味は変わらない。このことと同じようにラムダ式 $\lambda x.\,t$ において変数記号 x は入力をどう扱うかを記述しているだけなので、t の中の変数記号 x を t に現れない別の変数記号に置き換えてもラムダ式 $\lambda x.\,t$ の意味は変わらないと考えられる。そこで変数名のつけ替えに関してラムダ式を同一視するための同値関係を与える。

定義 2.2. ラムダ式の間の二項関係 R で**生成**される同値関係とは R を含むラムダ式の間の同値関係 $\overline{R}$ で

$$\begin{aligned}
(t,s) \in \overline{R} &\implies \forall u : \Lambda.\, (t\,u, s\,u) \in \overline{R} \\
(t,s) \in \overline{R} &\implies \forall u : \Lambda.\, (u\,t, u\,s) \in \overline{R} \\
(t,s) \in \overline{R} &\implies \forall x : \mathrm{Var}.\, (\lambda x.\,t, \lambda x.\,s) \in \overline{R}
\end{aligned}$$

を満たす最小のものである。

定義 2.3 (α 同値関係)**.** ラムダ式 t の中の変数記号 x を変数記号 y に置き換えたラムダ式を $t[x \mapsto y]$ と書く。α **同値関係**とは二項関係

$$\{(\lambda x.\,t, \lambda y.\,t[x \mapsto y]) \mid \text{変数記号 } y \text{ は } t \text{ に現れない}\}$$

で生成される同値関係である。ラムダ式 t と s が α 同値であるとき $t =_\alpha s$ と書く。

例えば以下のような α 同値関係が成り立つ。

$$\lambda x.\,x =_\alpha \lambda y.\,y \qquad x\,(\lambda x.\,y\,x) =_\alpha x\,(\lambda z.\,y\,z) \qquad (\lambda x.\,x)\,y =_\alpha (\lambda y.\,y)\,y$$

一方、$\lambda x.\,x\,y$ と $\lambda y.\,y\,y\,(= \lambda y.\,(x\,y)[x \mapsto y])$ は α 同値ではない。

ラムダ式の代入 $\int_0^1 (x+y)\,dx = \frac{1}{2} + y$ は正しいが、変数 y を文字通り x に置き換えると $\int_0^1 (x+x)\,dx = \frac{1}{2} + x$ となり正しくない。正しい結果を得るには先に $\int_0^1 (z+y)\,dz$ などと変数名をつけ替えてから変数 y を文字通り x に置き換える必要がある。ラムダ式の変数記号への代入でも同様の工夫が必要である。まずラムダ式 t、s、u が与えられたときこれらのラムダ式に現れない変数記号を選ぶ関数 $\upsilon(t,s,u) \in \mathrm{Var}$ を一つ取り、ラムダ式 t, s と変数記号 x に対してラムダ式 $t[s/x]$ をラムダ式の構成の帰納法で定義する。

$$\begin{aligned}
y[s/x] &= \begin{cases} s & (x = y) \\ y & (x \neq y) \end{cases} \\
(t_1\,t_2)[s/x] &= t_1[s/x]\,t_2[s/x] \\
(\lambda y.\,t)[s/x] &= \begin{cases} \lambda y.\,t & (x = y) \\ \lambda z.\,(t[y \mapsto z])[s/x] & (y \neq x \text{ かつ } z = \upsilon(t,s,x)) \end{cases}
\end{aligned}$$

命題 2.1. $s =_\alpha s'$ かつ $t =_\alpha t'$ なら $t[s/x] =_\alpha t'[s'/x]$ である。また $t[s/x]$ は α 同値を除いて υ の取り方に依存しない。

以降は α 同値なラムダ式を同一視してラムダ式 t の α 同値類を単に t と書く。命題 2.1 からラムダ式（の α 同値類）t と s に対して単に $t[s/x]$ と書いても問題はない。例えば

$$(x\,y)[z/x] = z\,y \qquad (x\,(\lambda x.\,x))[z/x] = z\,(\lambda x.\,x) \qquad (\lambda x.\,x\,y)[x\,y\,z/y] = \lambda w.\,w\,(x\,y\,z)$$

となる。一方、$(\lambda x.\,y)[x/y] \neq \lambda x.\,x$ である。

2.3 β 変換と β 同値関係

ラムダ式の中に現れる $(\lambda x.\,t)\,s$ の形の部分式を β **基**と呼び、ラムダ式の中の β 基 $(\lambda x.\,t)\,s$ を $t[s/x]$ に書き換える操作を β **変換**と言う。正確には β 変換は以下を満たす最小のラムダ式の間の二項関係 $\to_\beta$ である。

$$\begin{aligned}
&(\lambda x.\,t)\,s \to_\beta t[s/x]\\
&t \to_\beta s \implies u\,t \to_\beta u\,s\\
&t \to_\beta s \implies t\,u \to_\beta s\,u\\
&t \to_\beta s \implies \lambda x.\,t \to_\beta \lambda x.\,s
\end{aligned}$$

β 変換の反射推移閉包を $\twoheadrightarrow_\beta$ と書く。

ラムダ式を β 変換していくことでそのラムダ式の意味を求めることが出来る。例えば

$$(\lambda x.\,x)\,y \to_\beta y \qquad\qquad (\lambda xy.\,x)\,z\,w \to_\beta (\lambda y.\,z)\,w \to_\beta z \tag{3}$$

がなりたち、β 変換のもとで $\lambda x.\,x$ が恒等関数、$\lambda xy.\,x$ が z を受け取ると「常に z を返す定数関数 $\lambda y.\,z$」を返す関数を意味していることが分かる。

定義 2.4. β **同値関係**とは $\to_\beta$ を含む最小の同値関係である。ラムダ式 t と s が β 同値であるとき $t =_\beta s$ と書く。またラムダ式 t がラムダ式 s と β 同値であり、判定 $\Gamma \vdash t$ と $\Gamma \vdash s$ が成り立つとき $\Gamma \vdash t =_\beta s$ と書く。

ラムダ式 t が $t \to_\beta s$ となるラムダ式 s を持たないとき t は β **正規形**であると言う。例えば

$$\lambda x.\,x\,y\,z \qquad\qquad \lambda xyz.\,x\,(\lambda w.\,z)\,(\lambda w'.\,x\,y)$$

は β 正規形である。必ずしも β 変換によって β 正規形が得られるわけではない。以下のような例がある。

$$(\lambda x.\,x\,x)\,(\lambda x.\,x\,x) \to_\beta (\lambda x.\,x\,x)\,(\lambda x.\,x\,x) \to_\beta (\lambda x.\,x\,x)\,(\lambda x.\,x\,x) \to_\beta \cdots$$
$$(\lambda xy.\,x\,x)\,(\lambda xy.\,x\,x) \to_\beta \lambda y_1.\,(\lambda xy.\,x\,x)\,(\lambda xy.\,x\,x) \to_\beta \lambda y_1 y_2.\,(\lambda xy.\,x\,x)\,(\lambda xy.\,x\,x) \to_\beta \cdots$$

また β 変換を施す β 基の選び方によって β 正規形が得られる場合と得られない場合がある。

$$(\lambda yz.\,z)\,((\lambda x.\,x\,x)\,(\lambda x.\,x\,x)) \xrightarrow[\beta]{\beta} \lambda z.\,z$$

2.4 ラムダ式と再帰的関数

この章ではラムダ式による幾つかのデータ表現を紹介し、型無しラムダ計算が再帰的関数と同等の表現力を持っていることを示す。

2.4.1 組

ラムダ式 t と s に対して t と s に現れない変数記号 x を選び、ラムダ式 $\langle t, s\rangle$ を

$$\lambda x.\, x\, t\, s$$

で定義する。ラムダ式 $\mathtt{fst}$、$\mathtt{snd}$ と $\mathtt{mkpair}$ をそれぞれ

$$\mathtt{fst} = \lambda z.\, z\, (\lambda xy.\, x) \qquad \mathtt{snd} = \lambda z.\, z\, (\lambda xy.\, y) \qquad \mathtt{mkpair} = \lambda xyz.\, z\, x\, y$$

と定義すると、

$$\mathtt{fst}\, \langle t, s\rangle =_\beta t \qquad \mathtt{snd}\, \langle t, s\rangle =_\beta s \qquad \mathtt{mkpair}\, t\, s =_\beta \langle t, s\rangle$$

が成り立つ。

2.4.2 真偽値

ラムダ式 $\mathtt{true}$ と $\mathtt{false}$ を

$$\mathtt{true} = \lambda xy.\, x \qquad \mathtt{false} = \lambda xy.\, y$$

で定義する。ラムダ式 t と s が与えられたとき

$$\mathtt{true}\, t\, s =_\beta t \qquad \mathtt{false}\, t\, s =_\beta s$$

が成り立つ。以下では読みやすさのためにラムダ式 $t\, s\, u$ を $\mathtt{if}\ t\ \mathtt{then}\ s\ \mathtt{else}\ u$ と書く。

2.4.3 自然数

自然数 n に対して、ラムダ式 $\mathtt{c}_n$ を

$$\lambda f x.\, \overbrace{f(f(\cdots f(f}^{n}\ x)\cdots))$$

で定義する。この自然数の表現は Church によるもので **Church 数**と呼ばれる。Church 数に関して幾つかの計算がラムダ式で表現可能である。ラムダ式 $\mathtt{suc}$ と $\mathtt{iszero}$ を

$$\mathtt{suc} = \lambda n f x.\, f\, (n\, f\, x) \qquad \mathtt{iszero} = \lambda n.\, n\, (\lambda x.\, \mathtt{false})\, \mathtt{true}$$

と定義する。これらのラムダ式は β 同値のもとでそれぞれ 1 を足す関数と、0 には $\mathtt{true}$ を返し 1 以上の値には $\mathtt{false}$ を返す関数である。

$$\mathtt{suc}\, \mathtt{c}_n =_\beta \mathtt{c}_{n+1} \qquad \mathtt{iszero}\, \mathtt{c}_n =_\beta \begin{cases} \mathtt{true} & (n = 0) \\ \mathtt{false} & (n > 0) \end{cases}$$

また自然数を受け取る関数の帰納的定義が出来る。ラムダ式 $\mathtt{rec}$ を

$$\mathtt{rec} = \lambda xyn.\, n\, x\, y$$

とすると

$$\mathtt{rec}\, t\, s\, \mathtt{c}_n =_\beta \begin{cases} s & (n = 0) \\ t\,(\mathtt{rec}\, t\, s\, \mathtt{c}_m) & (n = m+1) \end{cases}$$

が成り立つ。例えばラムダ式 $\mathtt{add}$ を

$$\lambda nm.\, \mathtt{rec}\, \mathtt{suc}\, n\, m$$

で定義すると、

$$\mathtt{add}\, \mathtt{c}_n\, \mathtt{c}_m =_\beta \mathtt{c}_{n+m}$$

が成り立つ。

2.4.4 不動点演算子

ラムダ式 $\mathtt{Y}$ を

$$\mathtt{Y} = (\lambda xy.\, y\,(x\, x\, y))\,(\lambda xy.\, y\,(x\, x\, y))$$

と定義する。このラムダ式 $\mathtt{Y}$ は **Turing の不動点演算子**と呼ばれるもので

$$\mathtt{Y}\, t =_\beta t\,(\mathtt{Y}\, t)$$

を満たす。つまりラムダ式 t を関数と見たとき、$\mathtt{Y}\, t$ は t の不動点になっている。Turing の不動点演算子以外にも不動点演算子は存在する。

$$\mathtt{Y}_{\mathrm{Curry}} = \lambda y.\, (\lambda x.\, y\,(x\, x))\,(\lambda x.\, y\,(x\, x))$$

は **Curry の不動点演算子**と呼ばれ、$\mathtt{Y}_{\mathrm{Curry}}\, t =_\beta t\,(\mathtt{Y}_{\mathrm{Curry}}\, t)$ を満たす[2]。

不動点演算子を用いたプログラミング　ラムダ式 Ω を $\lambda xy.\, y\,(x\, x\, y)$ で定義すると、不動点演算子 $\mathtt{Y}$ は $\Omega\,\Omega$ という形に書ける。通常の集合と関数の感覚では関数を自分自身に関数適用することは無意味に感じられるかも知れないが、不動点演算子を使うと関数の再帰的定義が可能になる。関数の再帰的定義は関数型言語のプログラミング手法として基本的である。例として階乗を計算するラムダ式 $\mathtt{fact}$ を与える。全てを具体的に与えると長くなるので掛け算を計算するラムダ式 $\mathtt{mult}$ と一つ前の自然数を計算するラムダ式 $\mathtt{pred}$ は既に与えたとする。

$$\mathtt{mult}\, \mathtt{c}_n\, \mathtt{c}_m =_\beta \mathtt{c}_{n\times m} \qquad\qquad \mathtt{pred}\, \mathtt{c}_n =_\beta \begin{cases} \mathtt{c}_m & (n = m+1) \\ \mathtt{c}_0 & (n = 0) \end{cases}$$

[2]Klop による \$ も不動点演算子である。

$$\pounds = \lambda abcdefghijklmnopqstuvwxyzr.\, r\,(thisisafixedpointcombinator)$$
$$\$ = \pounds$$

(関数の帰納的定義を用いれば $\mathtt{mult}$ と $\mathtt{pred}$ は構成できる。) ラムダ式 t を

$$t = \lambda f n.\, \mathtt{if}\ (\mathtt{iszero}\, n)\ \mathtt{then}\ \mathtt{c}_1\ \mathtt{else}\ (\mathtt{mult}\, n\, (f\, (\mathtt{pred}\, n)))$$

と定義し、ラムダ式 $\mathtt{fact}$ を t の不動点 $\mathtt{Y}\, t$ と定義すると

$$\mathtt{fact}\, \mathtt{c}_n =_\beta \begin{cases} \mathtt{mult}\, \mathtt{c}_n\, (\mathtt{fact}\, \mathtt{c}_{n'}) & (n = n' + 1) \\ \mathtt{c}_1 & (n = 0) \end{cases}$$

となり、$\mathtt{fact}$ が階乗を計算するラムダ式であることが分かる。

最小化関数　ラムダ式 $\mathtt{min}$ を

$$\mathtt{min} = \lambda x.\, \mathtt{Y}\, (\lambda y n.\, \mathtt{if}\ (\mathtt{iszero}\, (x\, n))\ \mathtt{then}\ n\ \mathtt{else}\ (y\, (\mathtt{suc}\, n)))\, \mathtt{c}_0$$

定義する。ラムダ式 t が

$$(\forall n \geq 0.\, \exists m \geq 0.\, t\, \mathtt{c}_n =_\beta \mathtt{c}_m) \wedge (\exists n \geq 0.\, t\, \mathtt{c}_n =_\beta \mathtt{c}_0)$$

を満たすなら

$$\mathtt{min}\, t =_\beta \mathtt{c}_n \qquad (n \text{ は } t\, \mathtt{c}_n =_\beta \mathtt{c}_0 \text{ を満たす最小の自然数})$$

となる。

2.4.5　再帰的関数

定義 2.5. 関数 $f : \mathbb{N}^k \to \mathbb{N}$ について、ラムダ式 t が性質

$$\forall n_1, n_2, \cdots, n_k.\, t\, \mathtt{c}_{n_1} \cdots \mathtt{c}_{n_k} =_\beta \mathtt{c}_{f(n_1, \cdots, n_k)}$$

を満たすとき t は関数 f を**表現する**という。

定義 2.6. 再帰的関数を以下で帰納的に定義する。

- (零関数) $z_n(x_1, \cdots, x_n) = 0$ で与えられる零関数 $z_n : \mathbb{N}^n \to \mathbb{N}$ は再帰的関数。
- (後者関数) $s(n) = n + 1$ で与えられる後者関数 $s : \mathbb{N} \to \mathbb{N}$ は再帰的関数。
- (射影関数) $p_{n,i}(x_1, \cdots, x_n) = x_i$ で与えられる射影関数 $p_{n,i} : \mathbb{N}^n \to \mathbb{N}$ は再帰的関数。
- (関数合成) $\{f_i : \mathbb{N}^m \to \mathbb{N}\}_{i=1,2,\cdots,n}$ と $g : \mathbb{N}^n \to \mathbb{N}$ が再帰的関数なら

  $$h(x_1, \cdots, x_m) = g(f_1(x_1, \cdots, x_m), \cdots, f_n(x_1, \cdots, x_m))$$

  で与えられる関数 $h : \mathbb{N}^m \to \mathbb{N}$ は再帰的関数。
- (帰納的定義) $g : \mathbb{N}^n \to \mathbb{N}$ と $f : \mathbb{N}^{n+2} \to \mathbb{N}$ が再帰的関数の時

  $$h(x_1, \cdots, x_n, y) = \begin{cases} g(x_1, \cdots, x_n) & (y = 0) \\ f(x_1, \cdots, x_n, y', h(x_1, \cdots, x_n, y')) & (y = y' + 1) \end{cases}$$

  で与えられる関数 $h : \mathbb{N}^{n+1} \to \mathbb{N}$ は再帰的関数。

- （最小化関数）再帰的関数 $f : \mathbb{N}^{n+1} \to \mathbb{N}$ が以下を満たすとする。

$$\forall (x_1, \cdots, x_n) \in \mathbb{N}^n. \exists m \geq 0. f(x_1, \cdots, x_n, m) = 0$$

この時、

$$g(x_1, \cdots, x_n) = 「f(x_1, \cdots, x_n, m) = 0 \text{ を満たす最小の } m」$$

で与えられる $g : \mathbb{N}^n \to \mathbb{N}$ は再帰的関数。

定理 2.1. 任意の再帰的関数は表現可能。逆に表現可能な関数は再帰的関数である。

証明. 零関数、後者関数と最小化関数はそれぞれ $\mathtt{c}_0$、$\mathtt{suc}$ と $\mathtt{min}$ を使えば表現できる。射影関数 $p_{n,i}$ は $\lambda x_1 \cdots x_n. x_i$ により表現される。再帰的関数 $f_i : \mathbb{N}^m \to \mathbb{N}$ と $g : \mathbb{N}^n \to \mathbb{N}$ の表現をそれぞれ t_i と s とすると $\{f_i\}_{i=1,2,\cdots,n}$ と g から関数合成により得られる再帰的関数は

$$\lambda x_1 \cdots x_m. s\,(t_1\, x_1 \cdots x_m) \cdots (t_n\, x_1 \cdots x_m)$$

で表現される。再帰的関数 $g : \mathbb{N}^n \to \mathbb{N}$ と $f : \mathbb{N}^{n+2} \to \mathbb{N}$ による帰納的定義は f と g を表現する閉ラムダ式 t と s を用いると

$$\lambda x_1 \cdots x_n k. \mathtt{snd}\,(\mathtt{rec}\,(\lambda w. w\,(\lambda y z. \langle \mathtt{suc}\, y, t\, x_1\ \cdots\ x_n\, y\, z\rangle))\,\langle \mathtt{c}_0, s\, x_1\ \cdots\ x_n\rangle\, k)$$

により表現できる。ラムダ式 t で表現可能な関数 $f : \mathbb{N}^k \to \mathbb{N}$ が与えられたとき、$f(n_1, \cdots, n_k) = m$ となるのは $\mathtt{c}_m =_\beta t\, \mathtt{c}_{n_1}\ \cdots\ \mathtt{c}_{n_k}$ のときである。集合 $\{(s, s') \mid s =_\beta s'\}$ は再帰的に数え上げることが出来るので、f は再帰的関数。

型無しラムダ計算と部分再帰的関数　部分関数についてのラムダ式による表現可能性の定義を与える。一般にラムダ式 t は

1. $\lambda x_1 \cdots x_n. (\lambda x. s)\, s_1 \cdots s_m$
2. $\lambda x_1 \cdots x_n. x\, s_1 \cdots s_m$

のいずれかの形をしている。(1) の形をしているとき β 基 $(\lambda x. s)\, s_1$ を t の**頭基**と呼び、頭基に対する β 変換を**頭変換**と呼ぶ。頭変換の反射推移閉包を $\twoheadrightarrow_{\mathrm{h}}$ と書く。

定義 2.7. 頭基を持たないラムダ式を**頭正規形**と呼ぶ。ラムダ式 t に対して $t \twoheadrightarrow_{\mathrm{h}} s$ を満たす頭正規形のラムダ式 s が存在しないとき t を**不定式**と呼ぶ。

部分関数についてのラムダ式による表現可能性は以下のように定義される。

定義 2.8. 部分関数 $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$ について、ラムダ式 t が以下の条件

- $f(n_1, \cdots, n_k) = n$ なら $t\, \mathtt{c}_{n_1} \cdots \mathtt{c}_{n_k} =_\beta \mathtt{c}_n$
- $f(n_1, \cdots, n_k)$ が定義されていないなら $t\, \mathtt{c}_{n_1} \cdots \mathtt{c}_{n_k}$ は不定式。

を満たすとき t は部分関数 f を**表現する**という。

部分再帰的関数の定義は省くが、表現可能な部分関数の集合と部分再帰的関数の集合は一致する。チューリング機械や算術の概念とは無縁な型無しラムダ計算がチューリング機械で計算可能な関数である部分再帰的関数を特徴づけていることが「計算可能な関数」を部分再帰的関数として定義することとした **Church の提唱**の根拠の一つである。

3　ラムダ計算と型

変数を用いた数式を書くときはその変数が実数全体を動くのか、整数全体を動くのかなど変数がどの集合に属しているかを指定する。例えば変数 x が実数全体を動き、変数 y が整数全体を動くとき式 $y + x^2$ は実数の値を取ることを

$$x : \text{実数}, y : \text{整数} \vdash y + x^3 : \text{実数} \tag{4}$$

のように書くとする。さらに $x : \text{実数}, y : \text{整数} \vdash t : \text{正整数}$ と $x : \text{実数}, y : \text{整数} \vdash s : \text{正整数}$ という形の式にのみ最小公倍数を求める式

$$x : \text{実数}, y : \text{整数} \vdash \mathrm{lcm}(t, s) : \text{正整数}$$

を認めることで実数の間の最小公倍数を求めるといった無意味な式を排除できる。型とは (4) の「実数」や「整数」のように変数の動く範囲を意味するものである。型付きラムダ計算では型の整合性が取れないラムダ式を禁止することで無意味なラムダ式を排除している。型はプログラムの安全性の保証に応用されている [8]。

型とラムダ式　型付きラムダ計算はラムダ式と型システムと書き換え規則からなる。ここでは基底型が β のみの型付きラムダ計算を与える。型は以下で帰納的に定義される。

- 基底型 β は型。
- τ と σ が型なら $(\tau \Rightarrow \sigma)$ は型。

例えば

$$\beta \qquad (\beta \Rightarrow \beta) \qquad ((\beta \Rightarrow \beta) \Rightarrow \beta)$$

などが型である。直感な意味は β は集合で、$(\beta \Rightarrow \beta)$ は β から β への関数全体の集合、$((\beta \Rightarrow \beta) \Rightarrow \beta)$ は β から β への関数を受け取って β の値を返す関数全体の集合である。

型付きラムダ計算のラムダ式は以下で帰納的に与えられる。

1. 変数記号 $x, y, z, \cdots \in \mathrm{Var}$ はラムダ式。
2. （関数適用）t と s がラムダ式のとき $(t \cdot s)$ はラムダ式。
3. （関数抽象）t がラムダ式で x が変数記号、τ が型のとき $(\lambda x : \tau. t)$ はラムダ式。

関数抽象で変数記号に型がついている点が型無しラムダ計算と異なっている。ラムダ式 $\lambda x : \tau. t$ に現れる型 τ は関数の定義域を意味している。型付きラムダ計算についても型無しラムダ計算と同様の略記法を用いる。型についても曖昧にならない場合は括弧を省略する。

型システム　型付きラムダ計算では**型環境**によって変数記号がどの型に属しているかを指定する。例えば

$$x : \beta, y : \beta \Rightarrow \beta, z : \beta$$

が型環境である。この型環境は x が型 β、y が型 $\beta \Rightarrow \beta$、z が型 β の変数記号であることを意味している。

定義 3.1. **型環境**は変数記号と型の組の有限列で全ての変数記号は高々1 回だけ現れるものである。

ラムダ式の**型判定**とは型環境、ラムダ式と型の間の三項関係 $\Gamma \vdash t : \tau$ でラムダ式の構成に関して帰納的に定義される。

- $x_1 : \tau_1, \cdots, x_n : \tau_n \vdash x_i : \tau_i$
- $\Gamma \vdash t : \tau \Rightarrow \sigma$ かつ $\Gamma \vdash s : \tau$ なら $\Gamma \vdash t\, s : \sigma$
- $\Gamma, x : \tau \vdash t : \sigma$ なら $\Gamma \vdash \lambda x : \tau. t : \tau \Rightarrow \sigma$

例えば

$$\vdash \lambda x : \tau. x : \tau \Rightarrow \tau \qquad\qquad x : \sigma,\, z : \sigma \Rightarrow \tau \vdash z\, x : \tau$$

は型判定である。それぞれ、τ 上の恒等関数と σ から τ への関数に σ の値を関数適用する式を意味している。一方で

$$x : \beta,\, y : \tau \vdash x\, y : ?$$

のように関数型でないラムダ式を関数適用に用いることは許されていない。また

$$\Gamma \vdash \lambda x : \tau. x\, x : \sigma$$

はどのような型環境 Γ と型 τ, σ に対しても型判定とはならない。実際、変数 x は関数適用に使われているので τ は $\rho \Rightarrow \sigma$ という型でなくてはならないが、x は x に適用されているので $\rho \Rightarrow \sigma = \tau$ となる。このような型は存在せず $\Gamma \vdash \lambda x : \tau. x\, x : \sigma$ は型判定ではない。

型無しラムダ計算と同様にして、型付きラムダ計算に対しても α 同値関係、代入操作、β 変換、β 同値関係が定義できる。型付きラムダ計算のラムダ式についても α 同値関係に関する同一視を暗黙のうちに行う。また $\Gamma \vdash t : \tau$ と $\Gamma \vdash s : \tau$ が型判定であり、ラムダ式 t と s が β 同値であるとき

$$\Gamma \vdash t =_\beta s : \tau$$

と書く。例えば

$$\Gamma \vdash (\lambda x : \tau. t)\, s =_\beta t[s/x] : \sigma \qquad \Gamma \vdash \lambda y : \sigma. t\, ((\lambda x : \tau. x)\, s) =_\beta \lambda y : \sigma. t\, s : \sigma \Rightarrow \rho$$

等が成り立つ。正確な型付きラムダ計算の定義については [4, 2] を参照して頂きたい。

型無しラムダ計算との大きな違いは以下の性質である [5]。

定理 3.1. 任意の型付きラムダ計算のラムダ式 t はただひとつの β 正規形を持つ。

もう一つの大きな違いは次の章で触れるが、型付きラムダ計算のラムダ式 $\lambda x : \tau. t$ と $t\, s$ を集合論的な意味での関数抽象と関数適用として容易に解釈出来る点である。

3.1 型付きラムダ計算の表示的意味論

ここまで $\lambda x:\tau.t$ を関数抽象、$t\,s$ を関数適用と説明してきたが、この章では実際にこれらに関数抽象と関数適用の「意味」をつける**表示的意味論**[3]を与える。

集合 B を一つ選び、型 τ に対して集合 $[\![\tau]\!]$ を型の構成に関して帰納的に定義する。

- 基底型 β に対して $[\![\beta]\!]$ を B で定義する。
- $[\![\tau\Rightarrow\sigma]\!]$ を $[\![\tau]\!]$ から $[\![\sigma]\!]$ への関数全体の集合で定義する。

型 τ に対して集合 $[\![\tau]\!]$ を型 τ の**解釈**と呼ぶ。型の解釈を型環境の解釈に以下のように拡張する。

$$[\![\Gamma]\!]=\begin{cases}\{*\} & (\Gamma\text{ の長さ }|\Gamma|\text{ が }0)\\ [\![\tau_1]\!]\times\cdots\times[\![\tau_n]\!] & (\Gamma=(x_1:\tau_1,\cdots,x_n:\tau_n))\end{cases}$$

次に型判定 $\Gamma\vdash t:\tau$ に対し関数

$$[\![\Gamma\vdash t:\tau]\!]:[\![\Gamma]\!]\to[\![\tau]\!]$$

を型判定の導出に関して帰納的に定義する。

- $[\![x_1:\tau_1,\cdots,x_n:\tau_n\vdash t:\tau_i]\!](a_1,\cdots,a_n)=a_i$
- $[\![\Gamma\vdash t\,s:\tau]\!](\vec{a})=([\![\Gamma\vdash t:\sigma\Rightarrow\tau]\!](\vec{a}))([\![\Gamma\vdash s:\sigma]\!](\vec{a}))$
- $([\![\Gamma\vdash\lambda x:\tau.t:\tau\Rightarrow\sigma]\!](\vec{a}))(a)=[\![\Gamma,x:\tau\vdash t:\sigma]\!](\vec{a},a)$

関数 $[\![\Gamma\vdash t:\tau]\!]$ を型判定 $\Gamma\vdash t:\tau$ の**解釈**と呼ぶ。定義から分かる通り、$\Gamma\vdash\lambda x:\tau.t:\tau\Rightarrow\sigma$ の解釈は集合の世界での関数抽象で与えられ、$\Gamma\vdash t\,s:\tau$ の解釈は集合の世界での関数適用で与えられている。

型判定の解釈は β 同値性を保っている。

命題 3.1. $\Gamma\vdash t=_\beta s:\tau$ なら $[\![\Gamma\vdash t:\tau]\!]=[\![\Gamma\vdash s:\tau]\!]$。

例えば型判定

$$x:\beta\vdash\lambda f:\beta\Rightarrow\beta.\,f\,x:(\beta\Rightarrow\beta)\Rightarrow\beta\qquad x:\beta\vdash\lambda f:\beta\Rightarrow\beta.\,f\,(f\,x):(\beta\Rightarrow\beta)\Rightarrow\beta$$

の解釈はそれぞれ $a\in[\![\beta]\!]$ に対して以下の関数 $F_1:[\![\beta\Rightarrow\beta]\!]\to[\![\beta]\!]$ と $F_2:[\![\beta\Rightarrow\beta]\!]\to[\![\beta]\!]$ を返す関数である。

$$F_1(h)=h(a)\qquad\qquad F_2(h)=h(h(a))$$

集合 B が 2 つ以上の要素を持つときは $F_1\neq F_2$ であることから 命題 3.1 により

$$x:\beta\vdash\lambda f:\beta\Rightarrow\beta.\,f\,x\neq_\beta\lambda f:\beta\Rightarrow\beta.\,f\,(f\,x):(\beta\Rightarrow\beta)\Rightarrow\beta\tag{5}$$

であることが分かる。(5) は当たり前のように感じるかもしれないが定義からの直接証明が容易であるとは言えない。

[3]一般には**圏論的意味論**と呼ばれるものがある。圏論的意味論では型を圏の対象、型判定を圏の射として解釈する。型を集合、型判定を関数として解釈するのは集合と関数の圏 **Set** での型付きラムダ計算の圏論的意味論である。

4　型無しラムダ計算の表示的意味論

型付きラムダ計算の表示的意味論は型付きラムダ計算が実際の集合の世界での関数適用と関数抽象を形式化した体系であることを示している。では型無しラムダ計算も集合の世界での関数適用と関数抽象を形式化した体系であると言えるだろうか。実は、集合 X と X 上の関数全体の集合の間に関数

$$\phi : X \leftrightarrows \text{“}X \text{ 上の関数全体の集合”} : \psi$$

が存在して

$$\phi \circ \psi = \mathrm{id}$$

を満たすなら、型付きラムダ計算の表示的意味論とほぼ同じ方法で型無しラムダ計算の表示的意味論を与えることができる（4.2 章参照)。しかし型付きラムダ計算の場合とは異なり非自明な表示的意味論の構成は容易ではない。上のような状況を満たす集合は一点集合のみだからである。

命題 4.1. 集合 X について X 上の関数全体の集合から X への単射が存在するなら X は一点集合。

また不動点演算子の存在も型無しラムダ計算の集合論的解釈が困難であることを示唆している。任意のラムダ式 t は $t\,u = u$ を満たすラムダ式を持つが、集合 X 上の関数 $f : X \to X$ は必ずしも $f(x) = x$ となる $x \in X$ を持たない。

型無しラムダ計算に（非自明な）表示的意味論を与えるという問題は 1969 年に Dana Scott によって領域理論を使った解決がなされた。Scott が用いたのは集合と関数による表示的意味論ではなく完備半順序集合と連続関数によって型無しラムダ計算を解釈するという手法である。 4.1 章では領域理論に関する基本的な概念と具体例を与え、4.2 章と 4.3 章で領域理論に基づいた型無しラムダ計算の表示的意味論を与える。

4.1　領域理論

定義 4.1. ω **完備半順序集合** D は半順序集合 D で D の任意の上昇列

$$x_1 \le x_2 \le x_3 \le \cdots$$

が最小上界を持つものである。このような上昇列を ω **鎖**と呼ぶ。ω 鎖 $x_1 \le x_2 \le x_3 \le \cdots$ の最小上界を $\bigvee_{n\ge 1} x_n$ と書く。また、ω 完備半順序集合 D から ω 完備半順序集合 E への関数 $f : D \to E$ が半順序と ω 鎖の最小上界を保つとき f を**連続関数**と呼ぶ。

$$x \le y \implies f(x) \le f(y) \qquad\qquad f\left(\bigvee_{n\ge 1} x_n\right) = \bigvee_{n\ge 1} f(x_n)$$

例 4.1. 集合 X の部分集合全体 $\mathcal{P}(X)$ を考える。$\mathcal{P}(X)$ は包含関係によって半順序集合をなす。

$$p \le q \overset{\text{定義}}{\iff} \forall x : X,\, x \in p \implies x \in q$$

また $\mathcal{P}(X)$ の ω 鎖 $p_1 \le p_2 \le \cdots$ の最小上界は合併集合 $\bigcup_{n\ge 1} p_1$ である。

例 4.2. ω 完備半順序集合 D、E に対して D から E への連続関数全体の集合 $[D, E]$ は ω 完備半順序集合になる。連続関数 $f, g : D \to E$ について

$$f \leq g \overset{\text{定義}}{\iff} \forall x : D.\, f(x) \leq g(x)$$

で $[D, E]$ 上の半順序が定まる。$[D, E]$ の ω 鎖

$$f_1 \leq f_2 \leq f_3 \leq \cdots$$

の最小上界は各点毎の最小上界で与えられる。

$$\left(\bigvee_{n\geq 1} f_n\right)(x) = \bigvee_{n\geq 1} f_n(x)$$

例 4.3. 集合 X は半順序

$$x \leq x \overset{\text{定義}}{\iff} x = x$$

で ω 完備半順序集合になる。

4.2 領域理論による型無しラムダ計算の表示的意味論

ω 完備半順序集合 D と E に対して連続関数 $\phi : D \leftrightarrows E : \psi$ が存在して $\psi \circ \phi$ が恒等写像になるとき D を E の**レトラクト**といい、$\phi : D \lhd E : \psi$ または単に $D \lhd E$ と書く。$[D, D]$ をレトラクトにもつ ω 完備半順序集合 D の構成方法は幾つかあるが、ここではとりあえずレトラクト $\phi : [D, D] \lhd D : \psi$ の存在は認めて、ω 完備半順序集合 D を用いた型無しラムダ計算の表示的意味論を与える。

領域理論によるラムダ計算の表示的意味論では判定 $\Gamma \vdash t$ を以下の形の連続関数として解釈する。

$$[\![\Gamma \vdash t]\!] : \overbrace{D \times \cdots \times D}^{|\Gamma|} \to D \qquad (|\Gamma| \text{ は } \Gamma \text{ の長さ})$$

判定 $\Gamma \vdash t$ の解釈の定義を判定の導出に関する帰納法で与える。

- $[\![x_1, \cdots, x_n \vdash x_i]\!](a_1, \cdots, a_n) = a_i$
- $[\![\Gamma \vdash t\, s]\!](\vec{a}) = \psi([\![\Gamma \vdash t]\!](\vec{a}))([\![\Gamma \vdash s]\!](\vec{a}))$
- $[\![\Gamma \vdash \lambda x.\, t]\!](\vec{a}) = \phi(f)$ ただし $f(a) = [\![\Gamma, x \vdash t]\!](\vec{a}, a)$

命題 4.2. $\Gamma \vdash t =_\beta s$ なら $[\![\Gamma \vdash t]\!] = [\![\Gamma \vdash s]\!]$。

4.3 具体例の構成

$[D, D]$ をレトラクトに持つ ω 完備半順序集合 D の具体例の構成を与える。Scott が最初に与えた ω 完備半順序集合 D_∞ はある ω 完備半順序集合の射影極限を用いたものだが、ここでは Plotkin と Scott により独立に与えられたより簡単な構成を紹介する。

集合 X の有限部分集合全体を $\mathcal{P}_{\mathrm{fin}}(X)$ と書く。まず連続関数 $f : \mathcal{P}(X) \to \mathcal{P}(Y)$ に対して部分集合 $G(f) \subset \mathcal{P}_{\mathrm{fin}}(X) \times Y$ を

$$G(f) = \{(p, y) \in \mathcal{P}_{\mathrm{fin}}(X) \times Y \mid y \in f(p)\}$$

で定義し、部分集合 $g \subset \mathcal{P}_{\mathrm{fin}}(X) \times Y$ に対して関数 $F(g) : \mathcal{P}(X) \to \mathcal{P}(Y)$ を

$$(F(g))(p) = \{y \in Y \mid \exists p' \in \mathcal{P}_{\mathrm{fin}}(X),\, p' \subset p \text{ かつ } (p', y) \in g\}$$

で定義する。

命題 4.3. 任意の部分集合 $g \subset \mathcal{P}_{\mathrm{fin}}(X) \times Y$ について、$F(g) : \mathcal{P}(X) \to \mathcal{P}(X)$ は連続関数。また関数 $G : [\mathcal{P}(X), \mathcal{P}(X)] \leftrightarrows \mathcal{P}(\mathcal{P}_{\mathrm{fin}}(X) \times X) : F$ は連続で $F \circ G$ は恒等写像。

関数 $c : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ と $e : \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \to \mathbb{N}$ を

$$c(n, m) = \frac{(n+m)(n+m+1)}{2} + m \qquad\qquad e(p) = \sum_{n \in p} 2^n$$

で定義する。関数 c と e は全単射である。

関数 $f : X \to Y$ に対して連続関数 $\mathcal{P}(f) : \mathcal{P}(X) \to \mathcal{P}(Y)$ を

$$(\mathcal{P}(f))(p) = \{y \in Y \mid \exists x \in p.\, y = f(x)\}$$

で定義する。任意の $f : X \to Y$ について $\mathcal{P}(f)$ は連続関数であり、以下を満たす。

$$\mathcal{P}(f) \circ \mathcal{P}(g) = \mathcal{P}(f \circ g) \qquad\qquad \mathcal{P}(\mathrm{id}_X) = \mathrm{id}_{\mathcal{P}(X)} \tag{6}$$

定理 4.1. $[\mathcal{P}(\mathbb{N}), \mathcal{P}(\mathbb{N})]$ は $\mathcal{P}(\mathbb{N})$ のレトラクト。

証明. $c \circ (e \times \mathrm{id}) : \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ は全単射である。(6) より連続写像

$$\mathcal{P}(c \circ (e \times \mathrm{id})) : \mathcal{P}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \times \mathbb{N}) \leftrightarrows \mathcal{P}(\mathbb{N}) : \mathcal{P}((c \circ (e \times \mathrm{id}))^{-1})$$

はレトラクト $\mathcal{P}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \times \mathbb{N}) \lhd \mathcal{P}(\mathbb{N})$ を与える。命題 4.3 より $\mathcal{P}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \times \mathbb{N})$ は $[\mathcal{P}(\mathbb{N}), \mathcal{P}(\mathbb{N})]$ をレトラクトに持つので $[\mathcal{P}(\mathbb{N}), \mathcal{P}(\mathbb{N})]$ は $\mathcal{P}(\mathbb{N})$ のレトラクトである。

5　型無しラムダ計算と高階関数の計算可能性

Church の提唱によれば部分関数 $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$ が計算可能であることを型無しラムダ計算による表現可能性で定義できる。では集合

$$N_1 = \{f : \mathbb{N} \rightharpoonup \mathbb{N} \mid f \text{ は部分再帰的関数 }\}$$

から $\mathbb{N}$ への「計算可能」な部分関数や

$$N_2 = \{F : N_1 \rightharpoonup \mathbb{N} \mid F \text{ は「計算可能」}\}$$

から $\mathbb{N}$ への「計算可能」な部分関数とはどのようなものであろうか。候補の一つとして以下の意味で表現可能な部分関数を計算可能とする定義が考えられる。

定義 5.1. 部分関数 $F : N_1 \rightharpoonup \mathbb{N}$ について閉ラムダ式 t が以下の条件を満たすとき t は F を**表現する**という。任意の $f \in N_1$ と f を表現する任意のラムダ式 s に対して

- $F(f) = n$ なら $t\,s =_\beta \mathsf{c}_n$。
- $F(f)$ が定義されないなら $t\,s$ は不定式。

集合 N_1 から $\mathbb{N}$ への表現可能な部分関数全体の集合を N_2 と書く。集合 N_2 から $\mathbb{N}$ への部分関数に対して表現可能性を同様に定義する。

定義 5.2. 部分関数 $\Phi : N_2 \rightharpoonup \mathbb{N}$ について閉ラムダ式 t が以下の条件を満たすとき t は Φ を**表現する**という。任意の $F \in N_2$ と F を表現する任意のラムダ式 s に対して

- $\Phi(F) = n$ なら $t\,s =_\beta \mathsf{c}_n$。
- $\Phi(F)$ が定義されないなら $t\,s$ は不定式。

自然数の上の部分関数については計算可能性と表現可能性が一致するが、高階関数の世界には表現可能な部分関数以外に「計算可能」と呼べそうな部分関数が存在する[4]。

例 5.1. 部分関数 $P : N_1 \rightharpoonup \mathbb{N}$ を以下で定義する。

$$P(f) = \begin{cases} 0 & (f(0) = 0 \text{ または } f(1) = 0) \\ 1 & (f(0) = f(1) = 1) \\ \text{未定義} & (\text{その他}) \end{cases}$$

この部分関数は **parallel-or** と呼ばれ、次のアルゴリズムで「計算可能」である。

1. 部分再帰的関数 f を表現するラムダ式 t に対して $S_0 = \{t\,\mathsf{c}_0\}$、$S_1 = \{t\,\mathsf{c}_1\}$ と定義する。
2. もし $\mathsf{c}_0 \in S_0 \cup S_1$ なら計算を終了し 0 を返す。
3. もし $\mathsf{c}_1 \in S_0 \cap S_1$ なら計算を終了し 1 を返す。
4. S_0 と S_1 をそれぞれ

$$S_0 \overset{\text{再定義}}{=} S_0 \cup \{s' \mid \exists s \in S_0.\, s \to_\beta s'\} \qquad S_1 \overset{\text{再定義}}{=} S_0 \cup \{s' \mid \exists s \in S_1.\, s \to_\beta s'\}$$

と再定義し、2. に戻る。

例 5.2. 部分関数 $\Phi : N_2 \rightharpoonup \mathbb{N}$ を以下で定義する。

$$\Phi(F) = \begin{cases} 0 & (F(\bot) = 0) \\ 1 & (F(k_0) = 0 \text{ かつ } F(\bot) \text{ は未定義}) \\ \text{未定義} & (\text{その他}) \end{cases}$$

ただし N_1 の元 $\bot$ と k_0 はそれぞれ全ての点で値が未定義の部分関数と常に 0 を返す定数関数である。この部分関数 Φ は [6] で紹介されている。$\Phi(F)$ は以下のアルゴリズムで「計算可能」である。

[4]型無しラムダ計算の表示的意味論 $\mathcal{P}(\mathbb{N})$ を用いると Φ がラムダ式では表現されないことを示すことができる。またラムダ式の逐次性 (sequentiality [3]) を用いると P がラムダ式では表現されないことを示すことができる。

1. 部分関数 F を表現するラムダ式 t に対して t に可能な限り頭変換をほどこす。
2. 頭正規形が得られ、$\lambda x y_1 \cdots y_n.\, y_i\, t_1 \cdots t_m$ という形をしていた場合
   (a) $(\lambda x y_1 \cdots y_n.\, y_i\, t_1\, t_2\, \cdots\, t_m)\,((\lambda x.\, x\, x)\,(\lambda x.\, x\, x))$ に可能な限り頭変換をほどこす。
   (b) 頭正規形が得られ、$\lambda y z.\, z$ という形をしていれば 0 を返す。
3. 頭正規形が得られ、$\lambda x y_1 \cdots y_n.\, x\, t_1\, t_2\, \cdots\, t_m$ という形をしていた場合
   (a) $(\lambda x y_1 \cdots y_n.\, x\, t_1\, t_2\, \cdots\, t_m)\,(\lambda z.\, \mathsf{c}_0)$ に可能な限り頭変換をほどこす。
   (b) 頭正規形が得られ、$\lambda y z.\, z$ という形をしていれば 1 を返す。

ラムダ計算の表現可能性による計算可能性の定義をやめ、部分関数 P と Φ が計算可能となる定義を探せばいいように思えるが、P を計算可能とすると Φ の定義域に P を含める必要があり、その場合に Φ を計算する自然なアルゴリズムが存在するかは明らかではない。自然数上の計算可能性の状況とは異なり高階関数の計算可能性については本質的に複数の計算可能性の概念があるのではないかと考えられる事実も見つかっている [7, 6]。

参考文献

[1] 高橋正子, 計算論, 近代科学社, 1991.

[2] 大堀淳, プログラミング言語の基礎理論. 共立出版, 1997.

[3] H.P. Barendregt. *The Lambda Calculus.* Elsevier, 1984.

[4] Roy L. Crole. *Categories for Types.* Cambridge University Press, 1993.

[5] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types.* Cambridge University Press, New York, NY, USA, 1989.

[6] John Longley. The sequentially realizable functionals. 117(1–3):1–93, 2002.

[7] John Longley. Notions of computability at higher types I. In *In Logic Colloquium 2000*, pages 32–142, 2005.

[8] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[9] Glynn Winskel. *The Formal Semantics of Programming Languages.* The MIT Press, 1993.