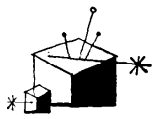
講　座



プログラム理論とその応用（2）†

伊　藤　貴　康††

3. プログラムの関数的意味と関数型論理

プログラムの意味記述を数学的に与えることは，プログラムの性質およびプログラムによって表現される計算過程を数学的に論じることを目的とするプログラム理論の基礎となる重要な事柄である．第2章で述べたプログラム図式の理論が対象としたのは図式的解釈の下でのプログラムの構造的な性質に関したものであった．ここではプログラムの関数的な意味記述を与える試みについて説明する．

プログラムの意味記述には，いくつかの方法が知られているが，プログラムを入力変数空間から出力変数空間への写像を行う関数であると考えて意味記述を行うのが関数的意味記述と呼ばれるものである．プログラムの関数的意味記述を明確な形で提唱したのが，McCarthy [1963] と Strachey [1965] である．このような立場の研究を数学的に発展させ，関数型論理に対する1つの無矛盾な計算モデルを与えたのが Scott によるプログラム束の理論である．この章では，プログラムの状態ベクトル・セマンティクス，帰納的証明法，プログラム代数 K とその性質および関数型論理——λ-計算と結合子論理——について簡単に説明する．これらは，次章で説明するプログラム束の理論が誕生する基礎を与えた Pre-Scott theory とも呼ばれるものである．

3.1 プログラムの状態ベクトル・セマンティクス

フローチャートおよびそれに対応する反復的プログラムの意味記述が，プログラム状態ベクトルを用いて行えることを第2章で述べた．たとえば，自然数 n の階乗 ($n!$) を計算する次の反復的プログラムを考えよう．

factorial: **begin** $x \leftarrow 1$;
　　$t \leftarrow n$;
　　fact: **while** $\neg(t=0)$ **do**
　　begin $x \leftarrow x \times t$;
　　$t \leftarrow t-1$
　　end
end　　(3.1)

このプログラムの意味は，プログラム状態ベクトルと内容関数 $c[x](\xi)$ および代入関数 $a[x, \alpha](\xi)$ を用いると，状態ベクトル上で定義される関数 **factorial**(ξ) として次のように与えられる．

$$\begin{aligned}
&\textbf{factorial}(\xi) = \textbf{fact}(e_2(e_1(\xi)))\\
&\quad \textbf{fact}(\xi) = \textbf{if } p(\xi) \textbf{ then } \xi \textbf{ else fact}(g(f(\xi)))\\
&\quad e_1(\xi) = a[x, 1](\xi)\\
&\quad e_2(\xi) = a[t, n](\xi)\\
&\quad p(\xi) = (c[t](\xi) = 0)\\
&\quad f(\xi) = a[x, c[x](\xi) \times c[t](\xi)](\xi)\\
&\quad g(\xi) = a[t, c[t](\xi) - 1](\xi)
\end{aligned} \qquad (3.2)$$

ここに，定数 $\{0, 1, n\}$ は解釈領域での値，演算 $\{\times, -\}$ とテスト $\{=\}$ も解釈領域での対応する演算およびテストであるとしている．また内容関数と代入関数については次のような性質が成り立つ．

① $c[x](a[y, \alpha](\xi)) = \textbf{if } x=y \textbf{ then } \alpha$
　　else $c[x](\xi)$

② $a[x, c[x](\xi)](\xi) = \xi$

③ $a[x, \alpha](a[y, \beta](\xi))$
$= \textbf{if } x=y \textbf{ then } a[x, \alpha](\xi)$
　　else $a[y, \beta](a[x, \alpha](\xi))$

ただし，この関係が成り立つのは α と β が定数の場合である．$\alpha = f(x)$，$\beta = g(y)$ の場合には

$a[x, f(x)](a[y, g(y)](\xi))$
$= \textbf{if } x=y \textbf{ then } a[x, g(f(x))](\xi)$
　　else $a[y, g(y)](a[x, f(x)](\xi))$　　(3.3)

また，x と t は異なるとすると，

$a[x, h(f(k(t)), g(t, x))](\xi)$
$= a[x, h(f(k(t)), x)](a[x, g(t, x)](\xi))$
$= a[x, h(f(t), x)](a[t, k(t)](a[x, g(t, x)](\xi)))$
　　(3.3 a)

† Mathematical Theory of Programs and its Applications by Takayasu ITO (Department of Electrical Communications, Faculty of Engineering, Tohoku University).

†† 東北大学工学部通信工学科

式 (3.2) の場合には，状態ベクトルは $\xi=(x,t)$ で与えられると考えてよい．**fact** に入るまえには e_1 および e_2 が施されているから状態ベクトルは $\xi=(1,n)$ となっており，**fact** の実行後には状態ベクトルは $\xi=(n!,0)$ となっているべきである．式 (3.2) が $n!$ を計算することを示すには，**fact** が $\xi=(1,n)$ で計算を開始し，

$$x\leftarrow t!\times x\,;\ t\leftarrow 0 \tag{3.4}$$

と同じ計算を行うプログラム（すなわち，同じ意味を持つプログラム）であることを示せばよいことが知られる．式(3.4)の状態ベクトル表現 $\varphi(\xi)=a[t,0](a[x,t!\times x](\xi))$ は次のように変形される*.

$$\begin{aligned}\varphi(\xi)&=\textbf{if } t=0 \textbf{ then } \varphi(\xi) \textbf{ else } \varphi(\xi)\\&=\textbf{if } t=0 \textbf{ then } a[t,0](a[x,0!](\xi))\\&\quad\textbf{else } a[t,0](a[x,(t-1)!\times(t\times x)](\xi))\\&=\textbf{if } t=0 \textbf{ then } \xi\\&\quad\textbf{else } a[t,0](a[x,t!\times x](a[t,t-1](a[x,t\times x](\xi))))\\&=\textbf{if } t=0 \textbf{ then } \xi\\&\quad\textbf{else } a[t,0](a[x,t!\times x](g(f(\xi))))\\&=\textbf{if } p(\xi) \textbf{ then } \xi \textbf{ else } \varphi(g(f(\xi)))\end{aligned} \tag{3.5}$$

すなわち

$$\varphi(\xi)=\textbf{if } p(\xi) \textbf{ then } \xi \textbf{ else } \varphi(g(f(\xi))) \tag{3.5a}$$

が得られる．一方，式(3.2)の **fact** の式は

$$\textbf{fact}(\xi)=\textbf{if } p(\xi) \textbf{ then } \xi \textbf{ else } \textbf{fact}(g(f(\xi))) \tag{3.5b}$$

となり，φ と **fact** は同じ再帰的関数定義を満足することが知られる．McCarthy [1963] は再帰的帰納法 (recursion induction) と呼ばれる次のような証明法を提案した**.

『関数 F を定義する次のような式が成り立つとする：

$$F(\xi)=\mathcal{E}(F\,;\,\xi) \tag{3.6}$$

ここに $\mathcal{E}(F\,;\,\xi)$ は F および ξ (およびその要素) を含む **if-then-else** から作られる式であるとする．このとき，$F(\xi)$ の計算が停止し，2つの関数 G と H が式 (3.6) の F にそれらを置き換えたときに成り立つならば，$G(\xi)=H(\xi)$ である．』

式(3.5a)および式(3.5b)から知られるように，φ と **fact** は同じ再帰的関数的定義を満たし，この再帰的関数定義は任意の自然数 n に対して $(t=0)$ のテストによってその計算が必ず停止することが知られる．したがって McCarthy の再帰的帰納法によって

$$\varphi(\xi)=\textbf{fact}(\xi) \tag{3.7}$$

が主張できることになる．$\varphi(\xi)$ および **fact**(ξ) の前に $e_1\,;\,e_2$ を実行してもそれらは同じになる．すなわち，

$$\begin{aligned}\varphi(e_2(e_1(\xi)))&=\textbf{fact}(e_2(e_1(\xi)))\\&=\textbf{factorial}(\xi)\end{aligned} \tag{3.8}$$

となり，**factorial**(ξ) は $n!$ を計算し，その値を x に与えることが知られる*.

状態ベクトルを用いてプログラムを表現し，この表現をもとにして2つのプログラムが同じ意味を持つことを示したが，上述の証明の過程ではいろいろな性質が用いられている．たとえば，式 (3.5) の変形においても次のような性質が使われている：

① 条件文 **if** p **then** F **else** $F=F$ の性質

② $t=0$ であれば $c[x](\xi)=1$ であるから $a[x,0!\times x](\xi)=a[x,0!](\xi)$ となること

③ 式 (3.3a) の性質によって2番目の式から3番目の式への変形が行えること

④ 式(3.5a)および式(3.5b)によって定義される計算が停止し，McCarthy の再帰的帰納法が適用できること

⑤ 状態ベクトルが定義される解釈領域上で演算やテストに関するいろいろな性質が利用できること

議論の対象となるプログラムとしてほかのものを考えたときには，ここで用いた性質だけでは不十分であり，対象に応じたいろいろな性質を利用する必要がある．どのような性質をどれだけ用意しておけばよいのか，再帰的に定義される関数の停止性はどのように証

---

* $a[x,t!\times x](\xi)$ は $a[x,(c[t](\xi))!\times(c[x](\xi))](\xi)$，$t=0$ は $c[t](\xi)=0$ などのように書くべきであるが繁雑になるから混同が起こらない限り略記する．

** 再帰的帰納法の正当性は後述のプログラム代数 K において証明されるが，概念的には原始帰納的関数定義の一意性を言っているのに過ぎない．

* 再帰的帰納法は再帰的定義関数の場合にも適用できる．たとえば，$f(n)=$**if** $n=0$ **then** 1 **else** $n\times f(n-1)$ が $n!$ を計算することの証明は次のように行える．

$$\begin{aligned}n!&=\textbf{if } n=0 \textbf{ then } n! \textbf{ else } n!\\&=\textbf{if } n=0 \textbf{ then } 0! \textbf{ else } n\times(n-1)!\\&=\textbf{if } n=0 \textbf{ then } 1 \textbf{ else } n\times(n-1)!\end{aligned}$$

したがって再帰的帰納法により $f(n)=n!$ が言える．

$$\begin{aligned}f(n)&=g(n,1)\\g(n,t)&=\textbf{if } n=0 \textbf{ then } t \textbf{ else } g(n-1,t\times n)\end{aligned}$$

に対しては

$$\begin{aligned}n!&=n!\times 1\\n!\times t&=\textbf{if } n=0 \textbf{ then } n!\times t \textbf{ else } n!\times t\\&=\textbf{if } n=0 \textbf{ then } t \textbf{ else } (n-1)!\times(t\times n)\end{aligned}$$

したがって再帰的帰納法により $g(n,t)=n!\times t$ が言え，$f(n)=n!$ が示せる．

明できるのか，McCarthy の再帰的帰納法は正しい証明法なのかといった問題が考えられた．プログラム理論の研究の主要な目的の1つはこれらの問題に答えることであった．

以上の議論では式(3.2)や $\varphi(\xi)$ のように状態ベクトル表現が与えられたものについて考えてきた．しかし式(3.1)に対して式(3.2)，式(3.4)に対して $\varphi(\xi)=a[t, 0](a[x, t!\times x](\xi))$ のような状態ベクトル表現を対応させる問題を一般的に扱うにはどのようにすればよいかという問題も考えられる．この問題に答えるには，プログラミング言語のセマンティクスを状態ベクトル法によって与える必要がある．

●言語の状態ベクトル・セマンティクス

簡単な例によって言語の状態ベクトル・セマンティクスがどのように与えられるかを示そう．

e を定数および変数から加算と乗算によって作られる算術式，ξ をプログラム変数が作る状態ベクトルとしたとき，状態ベクトル ξ の下での算術式 e の意味は次のように定義される **value**$[e](\xi)$ である．

$$\begin{aligned}&\textbf{value}[e](\xi)=\textbf{if isconst}[e]\ \textbf{then val}[e]\\&\quad\textbf{else if isvar}[e]\ \textbf{then } c[e](\xi)\\&\quad\textbf{else if issum}[e]\\&\qquad\textbf{then value}[S_L(e)](\xi)+\textbf{value}[S_R(e)](\xi)\\&\quad\textbf{else if isproduct}[e]\\&\qquad\textbf{then value}[M_L(e)](\xi)\times\textbf{value}[M_R(e)](\xi)\\&\quad\textbf{else undifined}\end{aligned}\qquad(3.9)$$

ここに val$[e]$ は e の定数としての値であり，$c[\cdot](\xi)$ は内容関数，＋と×は解釈領域上での値に対して定義された加算と乗算，$S_L(e)$ と $S_R(e)$ は e が加算の式のときに左側の項と右側の項を取り出す関数，$M_L(e)$ と $M_R(e)$ は e が乗算の式のときに左側の項と右側の項を取り出す関数とする．isconst(e)，isvar(e)，issum(e)，isproduct(e) は，それぞれ，e が定数，変数，和形式，積形式であれば真となる述語である*．

論理式は真か，偽か，等号・不等号で表わされる関係であるとしたとき論理式 p の状態ベクトル ξ の下での意味 **bool**$[p](\xi)$ は次のように与えられる：

$$\begin{aligned}&\textbf{bool}[p](\xi)=\textbf{if istrue}[p]\ \textbf{then true}\\&\quad\textbf{else if isfalse}[p]\ \textbf{then false}\\&\quad\textbf{else if iseq}[p]\\&\qquad\textbf{then value}[E_L(p)](\xi)=\textbf{value}[E_R(p)](\xi)\\&\quad\textbf{else if isless}[p]\\&\qquad\textbf{then value}[L_L(p)](\xi)<\textbf{value}[L_R(p)](\xi)\\&\quad\textbf{else if isgreat}[p]\\&\qquad\textbf{then value}[G_L(p)](\xi)>\textbf{value}[G_R(p)](\xi)\\&\quad\textbf{else undifined}\end{aligned}\qquad(3.10)$$

代入文，複合文，条件文，繰返し文の意味は意味関数 **sem**$[S](\xi)$ を用いて次のように与えられる．

① 代入文：$x\leftarrow e$ （e は算術式とする）

$$\textbf{sem}[x\leftarrow e](\xi)=a[x, \textbf{value}[e](\xi)](\xi)\qquad(3.11)$$

② 複合文：S_1；S_2

$$\textbf{sem}[S_1;\ S_2](\xi)=\textbf{sem}[S_2](\textbf{sem}[S_1](\xi))\qquad(3.12)$$

③ 条件文：IF p THEN S_1 ELSE S_2

$$\begin{aligned}&\textbf{sem}[\text{IF } p \text{ THEN } S_1 \text{ ELSE } S_2](\xi)\\&\quad=\textbf{if bool}[p](\xi)\ \textbf{then sem}[S_1](\xi)\\&\qquad\textbf{else sem}[S_2](\xi)\end{aligned}\qquad(3.13)$$

④ 繰返し文：WHILE p DO S

$$\begin{aligned}&\textbf{sem}[\text{WHILE } p \text{ DO } S](\xi)\\&\quad=\text{solution}[\gamma(\xi)\,|\,\gamma(\xi)=\textbf{if bool}[p](\xi)\ \textbf{then}\\&\qquad\gamma(\textbf{sem}[S](\xi))\ \textbf{else}\ \xi]\end{aligned}\qquad(3.14)$$

ここに solution$[\gamma(\xi)\,|\,\gamma(\xi)=\mathcal{E}(\gamma;\xi)]$ は $\gamma(\xi)=\mathcal{E}(\gamma;\xi)$ なる再帰的関数定義の（最小）解であるとする*．

以上からフローチャートと同程度の機能を持つプログラムの状態ベクトル・セマンティクスは

- 文の意味関数 **sem**$[S]$：$\boldsymbol{S\to S}$
- 算術式の意味関数 **value**$[e]$：$\boldsymbol{S\to N}$
- 論理式の意味関数 **bool**$[p]$：$\boldsymbol{S}\to$ {**true**, **false**}

なる3つの意味関数を用いて与えられることが分かる．（ここに $\boldsymbol{S}$ は状態ベクトルの集合，$\boldsymbol{N}$ は数値集合，**true** と **false** は真理値とする．）

プログラミング言語のセマンティクスを与える場合にも，**go-to** 文や再帰的手続きおよびデータ構造などの意味記述がどのように定義されるのか，繰返し文の意味記述に用いた再帰的関数定義の解は存在するのか，一意的に求まるのかといった問題が考えられる．

プログラムの意味や言語のセマンティクスが状態ベクトルを用いて論じられることを説明してきた．その際に提起された問題に答えるためのいろいろな試みが1960年代に行われたが，プログラム代数 $\boldsymbol{K}$ という形にまとめた1つの試みについて説明する**．

---

* 構文をこのような述語によって解析・記述する方法はアブストラクト・シンタックス法と呼ばれる．この方法の場合，たとえば，和を $x+y$，$xy+$，$+xy$，(PLUS x y) のどれで表わすかは具体的な形では問題としていないことに注意せよ．

* solution $[\cdot]$ は，後述するプログラム代数 $\boldsymbol{K}$ の最小化演算 μ_x $[\cdot]$ あるいは関数型論理の最小不動点結合子 $\boldsymbol{Y}$ と考えればよい．

** プログラム代数 $\boldsymbol{K}$ は，言わば Pre-Scott theory の1つの簡約化された体系である．

3.2 プログラム代数 K と帰納的証明法

プログラム代数 K は，プログラム状態ベクトル上で定義される基本演算 $(\phi, e, f_1, f_2, \cdots)$ と基本述語 $(\mathbf{T}, \mathbf{F}, p_1, p_2, \cdots)$ を要素として以下のように構成される．

プログラム代数 K の演算は次のように定義される：

① 基本演算 $(\phi, e, f_1, f_2, \cdots)$ は演算である

② x, y, z を演算，p を述語としたとき

- 合成　$(x \cdot y)$
- 条件式　$\lfloor_{p} x \vee y \rfloor$ または $\langle p \rangle \lfloor x \vee y \rfloor$
- 反復　$[_{p} x]$　または $\langle p \rangle [x]$
- 最小解　$\mu_z[z]$

は演算である

述語は次のように定義される：

① 基本述語 $(\mathbf{T}, \mathbf{F}, \omega, p_1, p_2, \cdots)$ は述語である

② x を演算，p を述語としたとき，xp は述語である

③ p, q を述語としたとき，$p \wedge q, p \vee q, p \to q, \rceil p$ は述語である

x, y を K の演算としたとき，K の論理式は

$x \leqslant y$

$x \simeq y$

のどちらかである．

プログラム代数 K の演算，述語，論理式は状態ベクトル上でその意味を与えられる．状態ベクトルの集合として与えられる解釈領域を $\mathbf{D}$ とし，演算と述語の意味を与える意味関数は，それぞれ，$s.[x](\xi)$ および $\pi.[p](\xi)$ によって与えられるとする．このとき

$s.[x]: \mathbf{D} \to \mathbf{D}$

$\pi.[p]: \mathbf{D} \to \{\omega, \mathbf{true}, \mathbf{false}\}$

ここに ω は **undefined** とし，$\mathbf{D}$ も Ω (undefined) をその要素として含むものとする（すなわち $\mathbf{D} = \{\Omega\} \cup \{\xi\}$）．

演算と述語の意味は次のように与えられる：

$s.[x](\Omega) = \Omega$；$s.[\phi](\xi) = \Omega$

$s.[e](\xi) = \xi$

$s.[f_i](\xi) = \hat{f}_i(\xi)$　($\hat{f}_i$ は $\mathbf{D}$ 上での関数)

$s.[x \cdot y](\xi) = s.[y](s.[x](\xi))$

$s.[\lfloor_{p} x \vee y \rfloor](\xi) = \mathbf{if}\ \pi.[p](\xi)\ \mathbf{then}\ s.[x](\xi)$
$\mathbf{else}\ s.[y](\xi)$

$s.[[_{p}[x]](\xi) = \mathbf{while}\ \pi.[p](\xi)\ \mathbf{do}\ s.[x](\xi)$

$s.[\mu_z[z]](\xi) = \mathbf{min}[x(\xi) \mid x(\xi) = z(x(\xi))]$

ここに $\mathbf{min}[x(\xi) \mid x(\xi) = z(x(\xi))]$ は $x(\xi) = z(x(\xi))$ を満たす（$\leqslant$に関する）最小解 $x(\xi)$ を意味している．

$\pi.[p](\Omega) = \omega$

$\pi.[\omega](\xi) = \omega$

$\pi.[\mathbf{T}](\xi) = \mathbf{true}$；$\pi.[\mathbf{F}](\xi) = \mathbf{false}$

$\pi.[xp](\xi) = \pi.[p](s.[x](\xi))$

$\pi.[p \vee q](\xi) = \mathbf{if}\ \pi.[p](\xi)\ \mathbf{then\ true}$
$\mathbf{else}\ \pi.[q](\xi)$

$\pi.[p \wedge q](\xi) = \mathbf{if}\ \pi.[p](\xi)\ \mathbf{then}\ \pi.[q](\xi)$
$\mathbf{else\ false}$

$\pi.[p \to q](\xi) = \mathbf{if}\ \pi[p](\xi)\ \mathbf{then}\ \pi.[q](\xi)$
$\mathbf{else\ true}$

$\pi.[\rceil p](\xi) = \mathbf{if}\ \pi[p](\xi)\ \mathbf{then\ false\ else\ true}$

ここに **if** P **then** Q **else** R は，P が **true** なら Q，P が **false** なら R，P が ω (undefined) なら ω とする．

次に K の論理式の意味は次のように与えられる*：

- $x \leqslant y \Leftrightarrow \forall \xi \in \mathbf{D}.\ [\mathbf{if}\ s.[x](\xi) = \eta$
$\mathbf{then}\ s.[y](\xi) = \eta]$
（ここに η は undefined でないとする）
したがって，$\phi \leqslant x \Leftrightarrow \Omega \leqslant s.[x](\xi)$
- $x \simeq y \Leftrightarrow (x \leqslant y)\ \mathbf{and}\ (y \leqslant x)$
$\Leftrightarrow \forall \xi \in \mathbf{D}.\ [s.[x](\xi) = s.[y](\xi)]$

●プログラム代数 K の公理

K において次のような公理が成り立つ．なお，論理式 Φ から Ψ が主張できることを $\Phi \vdash \Psi$ と書くとする．

(1) 結合則の公理

$\phi \cdot x \simeq x \cdot \phi \simeq \phi$，$e \cdot x \simeq x \cdot e \simeq x$

$x \cdot (y \cdot z) \simeq (x \cdot y) \cdot z$

(2) 半順序の公理

$\phi \leqslant x$

$x \leqslant x$

$x \leqslant y, y \leqslant x \vdash x \simeq y$

$x \leqslant y, y \leqslant z \vdash x \leqslant z$

(3) 条件式の公理

$\lfloor_{\omega} x \vee y \rfloor \simeq \phi$

$\lfloor_{\mathbf{T}} x \vee y \rfloor \simeq x$

$\lfloor_{\mathbf{F}} x \vee y \rfloor \simeq y$

---

* $x \leqslant y$ は，『x is compact in y』，『x is an approximation to y』あるいは『y is an extension of x』のように呼ばれる．

$\lfloor_p x \vee y \rfloor \simeq \lfloor_p y \vee x \rfloor$

$\lfloor_p x \vee x \rfloor \leqslant x$

$\lfloor_p x \vee y \rfloor \simeq \lfloor_p \lfloor_p x \vee z \rfloor \vee y \rfloor$

$\lfloor_p x \vee y \rfloor \simeq \lfloor_p x \vee \lfloor_p z \vee y \rfloor \rfloor$

$\lfloor_{p,q} \lfloor x_1 \vee y_1 \rfloor \vee \lfloor_q x_2 \vee y_2 \rfloor \rfloor \simeq \lfloor_{q,p} \lfloor x_1 \vee x_2 \rfloor \vee \lfloor_p x_2 \vee y_2 \rfloor \rfloor$

$\lfloor_p x \vee \lfloor_q y \vee z \rfloor \rfloor \simeq \lfloor_p x \vee \lfloor_{q,p} \lfloor y \vee y \rfloor \vee \lfloor_p z \vee z \rfloor \rfloor \rfloor$

$\lfloor_{p,q} \lfloor x \vee y \rfloor \vee z \rfloor \simeq \lfloor_{p,q,p} \lfloor \lfloor x \vee x \rfloor \vee \lfloor_p y \vee y \rfloor \rfloor \vee z \rfloor$

$\lfloor_p x \vee y \rfloor \cdot z \simeq \lfloor_p x \cdot z \vee y \cdot z \rfloor$

$x \cdot \lfloor_p y \vee z \rfloor \simeq \lfloor_{x,p} x \cdot y \vee x \cdot z \rfloor$

$x \leqslant y, u \leqslant v \vdash \lfloor_p x \vee u \rfloor \leqslant \lfloor_p y \vee v \rfloor$

(4) 反復の公理

$[x] \simeq \lfloor_p x \cdot [x]_p \vee e \rfloor$

$[x]_p \simeq [[x]_p]_p$

$[x]_p \simeq [x \cdot [x]_p]_p$

$[x]_p \simeq [\lfloor [x]_p \vee z \rfloor]_p$

$[x]_{\mathbf{T}} \simeq \phi$

$[e]_p \simeq \lfloor \phi \vee e \rfloor_p$

$[[e]_q]_p \simeq [e]_p$

$[x]_p \cdot [y \vee z]_p \simeq [x]_p \cdot z$

(5) 最小化演算の公理

$\mu_z[z(x)] \simeq z(\mu_z[z(x)]/x)$

ここに $z(x)$ は z が x を含む演算であることを意味しており，$z(\mu_z[z(x)]/x)$ は $z(x)$ のあらゆる x を $\mu_z[z(x)]$ に置き換えて得られる式であることを意味している．

(6) プログラム代数 $\boldsymbol{K}$ の推論規則

R1
$$\frac{x \leqslant y}{z(x) \leqslant z(y/x)}$$

ここに $z(x)$ は z が x を含む演算であり，$z(y/x)$ は $z(x)$ のあらゆる x を y に置き換えて得られる式である．

R2
$$\frac{y \simeq \lfloor_p a \cdot y \vee b \rfloor}{y \simeq [a]_p \cdot b}$$

ここに a, b は演算であるとする．

R3 (μ-帰納法)
$$\frac{\Phi \vdash \Psi[\phi/x] \qquad \Phi, \Psi \vdash \Psi[z/x]}{\Phi \vdash \Psi[\mu_z[z]/x]}$$

ここに Φ, Ψ は論理式であり，$\Psi[z/x]$ の記法は Ψ にあらわれるあらゆる x を z に置き換えて得られる論理式であることを示している．また Φ には x は（自由変数としては）現われないとする．

●プログラム代数 $\boldsymbol{K}$ の基本的性質

プログラム代数 $\boldsymbol{K}$ の上述の公理の多くは意味関数 $s.[x](\xi)$ と $\pi.[p](\xi)$ および半順序 $\leqslant$ の定義から容易に知られる．しかし，推論規則 (**R1**, **R2**, **R3**) および最小化演算の公理は自明ではないので，プログラム代数 $\boldsymbol{K}$ の性質について説明してから，それらの証明の概要を与える．

プログラム代数 $\boldsymbol{K}$ は $\leqslant$ に関する半順序集合であり，ϕ は最小要素であることが知られる．半順序集合の部分集合 X に対して，$\forall x \in X.[a \leqslant x]$ を満たすとき a は下界 (lower bound) と呼ばれ，任意の下界 a に対して $a \leqslant c$ となる下界 c は**下限** (g. l. b.: greatest lower bound) と呼ばれ，X の下限を $\vee\{x \mid x \in X\}$，2つの要素 x と y の下限は $x \wedge y$，$x_1 \wedge x_2 \wedge \cdots \wedge x_n$ は $\bigwedge_{i=1}^{n} x_i$ と記される．同様に上界 (upper bound) と**上限** (l. u. b.: least upper bound) が定義され，X の上限は $\vee\{x \mid x \in X\}$，2つの要素 x と y の上限は $x \vee y$，$x_1 \vee x_2 \vee \cdots \vee x_n$ は $\bigvee_{i=1}^{n} x_i$ と記される．$\boldsymbol{K}$ の任意の部分集合に上限および下限が存在すれば $\mu_z[z]$ は z の最小不動点であり，$x \simeq z(x)$ となる x の上限 $\vee\{x \mid x \simeq z(x)\}$ と一致することが示される．

$\tau(x)$ を x を含む任意の演算としたとき，演算 τ に関する**単調性** $x \leqslant y \vdash \tau(x) \leqslant \tau(y)$ が成り立つとすると，次の最小不動点定理が成り立つことが知られる．

〔最小不動点定理〕

$f(x) \leqslant x$ となる最小要素が一意的に存在し，それは $f(x) \simeq x$ となる最小要素でもある．

(証明) $X = \{y \mid f(y) \leqslant y\}$ および $x \simeq \wedge X$ とする．

① $y \in X$ に対して $\wedge$ の定義から $x \leqslant y$．単調性から $f(x) \leqslant f(y)$．$y \in X$ であるから $f(x) \leqslant y$．あらゆる $y \in X$ に対してこれが言えるから $f(x) \leqslant \wedge X \simeq x$．

② 単調性から $f(x) \leqslant x \vdash f(f(x)) \leqslant f(x)$，したがって
$f(x) \in X$; $x \simeq \wedge X \leq f(x)$，すなわち

$f(x)\simeq x$

③ $f(y)\simeq y$ なら $y\in X$, したがって $x\simeq\bigwedge X\leqslant y$.

したがって, x は最小不動点である.

次に, $f(x)\simeq x$ の解が $x_0\simeq\overset{\infty}{\underset{i=0}{\forall}}f^i(\phi)$ で与えられ, それが最小不動点となることが示される.

$$\phi,\ f(\phi),\ f^2(\phi),\cdots \tag{3.15}$$

を考えると, f の単調性から

$$\phi\leqslant f(\phi)\leqslant f^2(\phi)\leqslant\cdots \tag{3.16}$$

式 (3.15) の演算の系列には上限 (l. u. b.)$x_0\simeq\overset{\infty}{\underset{i=0}{\forall}}f^i(\phi)$ が存在し, f の連続性からこの x_0 が $f(x)\simeq x$ の最小不動点であることが知られる. ここに演算 f が**連続**であるというのは, 演算の系列

$$a_0\leqslant a_1\leqslant a_2\leqslant\cdots \tag{3.17}$$

に対して

$$f(\overset{\infty}{\underset{i=0}{\forall}}a_i)\simeq\overset{\infty}{\underset{i=0}{\forall}}f(a_i) \tag{3.18}$$

が成り立つことである. 以上の事柄および最小化演算の公理から, 次の定理が成り立つことが知られる.

〔最小解の定理〕

$x\simeq f(x)$ には, 半順序 $\leqslant$ に関する最小解が存在し, その解 x_0 は

$$x_0\simeq\mu_x[f(x)]\simeq\overset{\infty}{\underset{i=0}{\forall}}f^i(\phi) \tag{3.19}$$

ここに $f^0(\phi)=\phi$, $f^1(\phi)=f(\phi)$, $f^2(\phi)=f(f(\phi))$, ….

以上の結果は, 演算の単調性と連続性を用いて証明されたことである. プログラム代数 $\boldsymbol{K}$ に関しては次の定理が成り立つことから, 以上の議論が正当化される.

〔単調性・連続性の定理〕

プログラム代数 $\boldsymbol{K}$ の演算はすべて単調・連続である.

この定理の証明は式に関する帰納法により行われる. その概要を説明しよう. 演算の合成と条件式に関しては, 単調性と連続性が共に保存されることが容易に知られる. また演算の反復に関しては $\underset{p}{[a]}\simeq\mu_x[\underset{p}{[a\cdot x}$ Ve$]]$ となるから, 最小化演算 $\mu_x[\cdot]$ が単調性を保存することを示せば十分である. $f(x)(y)$ を x と y に関して単調・連続な演算とする. このとき $\mu_y[f(x)(y)]$ が x に関して単調・連続であることが以下のようにして示せる. $f(x)(y)$ の連続性から $\mu_y[f(x)(y)]\simeq\overset{\infty}{\underset{i=0}{\forall}}(f(x))^i(\phi)$. したがって $x\leqslant x'$ なら $\mu_y[f(x)(y)]\leqslant\mu_y[f(x')(y)]$ を示すには, (あらゆる i に対して) $x\leqslant x'$ なら $(f(x))^i(\phi)\leqslant(f(x'))^i(\phi)$ を示せばよい. この証明は帰納法により行える. $i=0$ のときは明らか. $(f(x))^{i+1}(\phi)\simeq f(x)((f(x))^i(\phi))\leqslant f(x')((f(x))^i(\phi))\leqslant f(x')((f(x'))^i(\phi))\leqslant(f(x'))^{i+1}(\phi)$. したがって最小化演算が単調性を保存することが言える. 次に最小化演算の連続性の証明を考える. $x_0\leqslant x_1\leqslant x_2\leqslant\cdots$ としたとき, $\mu_y[f(\overset{\infty}{\underset{i=0}{\forall}}x_i)(y)]\simeq\overset{\infty}{\underset{i=0}{\forall}}\mu_y[f(x_i)(y)]$ を示すわけであるが, f が y に関して連続であることから, $\overset{\infty}{\underset{j=0}{\forall}}(f(\overset{\infty}{\underset{i=0}{\forall}}x_i))^j(\phi)\simeq\overset{\infty}{\underset{i=0}{\forall}}\overset{\infty}{\underset{j=0}{\forall}}(f(x_i))^j(\phi)\simeq\overset{\infty}{\underset{j=0}{\forall}}\overset{\infty}{\underset{i=0}{\forall}}(f(x_i))^j(\phi)$ と書ける. したがって, $(f(\overset{\infty}{\underset{i=0}{\forall}}x_i))^j(\phi)\simeq\overset{\infty}{\underset{i=0}{\forall}}(f(x_i))^j(\phi)$ を示せばよい. これは j に関する帰納法で証明できる. $j=0$ のときは明らか. $(f(\overset{\infty}{\underset{i=0}{\forall}}x_i))^{j+1}(\phi)\simeq f(\overset{\infty}{\underset{i=0}{\forall}}x_i)(f(\overset{\infty}{\underset{i=0}{\forall}}x_i))^j(\phi)\simeq f(\overset{\infty}{\underset{i=0}{\forall}}x_i)\overset{\infty}{\underset{i=0}{\forall}}(f(x_i))^j(\phi)\simeq\overset{\infty}{\underset{i=0}{\forall}}\overset{\infty}{\underset{k=0}{\forall}}f(x_i)((f(x_k))^j(\phi))\simeq\overset{\infty}{\underset{n=0}{\forall}}f(x_n)((f(x_n))^j(\phi))\simeq\overset{\infty}{\underset{n=0}{\forall}}(f(x_n))^{j+1}(\phi)$. したがって最小化演算が連続性を保存することも言えたことになる.

● プログラム代数 $\boldsymbol{K}$ における帰納的証明法

推論規則について考えるまえに, 再帰的帰納法について考えてみよう. 演算 x が

$$x\simeq f(x) \tag{3.20}$$

によって定義されるとするときに,

$$y\simeq f(y)$$
$$z\simeq f(z) \tag{3.21}$$

が成り立つならば, $x(\xi)\neq\phi$ なる ξ に対して

$$y(\xi)\simeq z(\xi) \tag{3.22}$$

であるというのが再帰的帰納法である. x_0 を式 (3.20) を満たす最小解とすると, 式(3.21)から $x_0\leqslant y$ かつ $x_0\leqslant z$ となる. 半順序 $\leqslant$ の定義から, $x_0(\xi)$ が定義されるあらゆる ξ に対して, $x_0(\xi)=y(\xi)$ かつ $x_0(\xi)=z(\xi)$ が言え, そのような ξ に対して $y(\xi)=z(\xi)$ が言える. したがって再帰的帰納法が成り立つことが分かる.

推論規則 **R 3** (μ-帰納法) の証明を考えよう. 推論の仮定から

$$\Phi\vdash\Psi[\phi]$$
$$\Phi,\ \Psi[x]\vdash\Psi[f(x)] \tag{3.24}$$

が成り立つと考えてよい. したがって, 次のような系列を得る:

$$\Phi\vdash\Psi[\phi]$$
$$\Phi\vdash\Psi[f(\phi)]$$
$$\Phi\vdash\Psi[f^2(\phi)]$$
$$\vdots$$
$$\Phi\vdash\Psi[f^n(\phi)] \qquad (3.25)$$
$$\vdots$$

したがって

$$\Phi\vdash\Psi[\overset{\infty}{\underset{n=0}{\vee}}f^n(\phi)] \qquad (3.26)$$

Ψ（に現われる演算）の連続性から

$$\Phi\vdash\Psi[\overset{\infty}{\underset{n=0}{\vee}}f^n(\phi)] \qquad (3.27)$$

したがって

$$\Phi\vdash\Psi[\mu_x[f(x)]] \qquad (3.28)$$

• プログラム代数 K とプログラムの性質

プログラム代数 K の形式化の下でプログラムの諸種の性質を形式化することができるので，そのことについて説明しておこう．

① プログラムの同値性，停止性，発散

- 同値性： $x\simeq y$
- 停止性： $\neg(x\simeq\phi)$
- 発　散： $x\simeq\phi$

② プログラムの正当性

- ある問題を解く2つのプログラム x と x が与えられ，x が正しいと分かっているとき，$x\simeq x$ なら x も正しいと言われ，$x\leqslant x$ なら x は x に対する近似プログラムであると言われる．

プログラムの正当性を入力条件および出力条件に関する正当性問題として論じるために，半順序関係 $\leqslant$ を述語の場合に対しても拡張して用いることにする．

$\omega\leqslant$**true**, $\omega\leqslant$**false**, $\omega\leqslant p$

が成り立つとする．入力条件を Q，出力条件を R と書き

$Q,\ R$: **D**$\rightarrow\{\omega,$ **true, false**$\}$

であるとする．このとき，入出力条件 (Q,R) に対してプログラム x の正当性を主張するいろいろな表現を与えることができる．たとえば，

ⓐ $Q\leqslant x\cdot R$

すなわち，$\pi.[Q](\xi)\leqslant\pi.[R](s.[x](\xi))$．これは入力条件 Q が **true** であるか **false** であるときには出力条件がそれぞれ **true** か **false** の値を取ることを要求するが，入力条件が **undefined** のときには出力条件には何も要求しないものとなる．なお，$\mathbf{T}\leqslant Q\leqslant x\cdot R$ とするときわめて強い正当性条件となることが知られる．

ⓑ **T**$\leqslant$**if** Q **then** $x\cdot R$ **else T**

すなわち **true** $\leqslant$ **if** $\pi.[Q](\xi)$ **then** $\pi.[R](s.[x](\xi))$ **else true** これは入力条件 Q が **false** の場合は常に成り立ち，Q が **true** の場合には x が停止して（$x\neq\phi$ で）かつ出力条件 R が **true** となるときにのみ成り立つ．また

ⓑ′ **T**$\leqslant$**if** Q **then** $x\cdot R$ **else F**

のときには，入力条件 Q が **true** で x が停止して出力条件 R が **true** となるときにのみ成り立つ．

これらⓑおよびⓑ′は強い意味でのプログラムの正当性を主張するものと言える．

ⓒ $x\cdot R\leqslant$**if** Q **then T else** $x\cdot R$

この式が成り立つのは Q が **true** で x の実行後に停止して R が成り立つ場合，Q が **true** で x が停止しないプログラムの場合，あるいは Q が **false** の場合である．これは Hoare の検証文 $\{Q\}x\{R\}$ による正当性の主張そのものである．

プログラムの正当性の概念をプログラム代数 K の形式化を用いて与え，正当性の検証を行うことが期待されるが，この詳細はプログラム束の理論の応用について述べる後章で説明する．

3.3 関数型論理（λ-計算および結合子論理）

プログラムの意味が状態ベクトルを用いて再帰的関数定義の解として表現でき，その中に現われる対象固有の関数や演算の性質（たとえば，代入関数や内容関数の性質，四則演算の性質など）とプログラム代数 K の公理や推論規則を用いてプログラムに関する論証を行うことができることがこれまでの説明から知られる．これらの扱いでは，階乗プログラムの例からも知られるように，あらゆる演算は状態ベクトル上の演算として関数的に定義されており，これはプログラム代数 K の演算の解釈においても同様である．あらゆる対象を関数的に取り扱う論理的体系として λ-計算 (Lambda calculus) や結合子論理 (Combinatory logic) があり，これらはプログラム代数 K やプログラム束の理論と密接な関係がある．λ-計算の考え方は LISP 1.5 に用いられているものであるが，この説明から始める．

• λ-計算 (λ-calculus)

λ-計算は A. Church によって導入されたもので，関数定義に現われる変数の束縛 (binding) に λ-記法と呼ばれるものを用いるのが出発点である．たとえば，x^2+2y^2 という式は x を変数とする関数 f を定義するとも考えられ，また y を変数とする関数 g を

定義するとも考えられる．λ-記法を用いるとこれらの関数を次のように明確に表現できる：

$$f=\lambda x.\ x^2+2y^2$$
$$g=\lambda y.\ x^2+2y^2 \qquad (3.29)$$

f は y に関する関数 $\hat{f}$，g は x に関する関数 $\hat{g}$ であると考えると，

$$\hat{f}=\lambda y.\ (\lambda x.\ x^2+2y^2)$$
$$\hat{g}=\lambda x.\ (\lambda y.\ x^2+2y^2) \qquad (3.29\,a)$$

のように書け，多変数の場合も扱えることになる．このとき式の評価は次のように行われることに注意する必要がある：

$$f(a)=\lambda x.\ x^2+2a^2;\ (\hat{f}(a))(b)=b^2+2a^2$$
$$\hat{g}(a)=\lambda y.\ a^2+2y^2;\ (\hat{g}(a))(b)=a^2+2b^2 \qquad (3.30)$$

λ-計算の λ-項は次のように定義される．①定数か変数であれば λ-項である，②M が λ-項で x が変数のとき $(\lambda x.M)$ は λ-項である*，③M と N が λ-項なら (MN) はλ-項である**．変数 x が λ-項の中に $\lambda x.M$ のような形で現われれば x は束縛変数であり，そうでなければ自由変数であると言われる．

λ-記法を用いると多変数の関数を1変数関数の関数の適用の繰返しとして表現できる．たとえば，$f(x, y)$ に対して

$$g\equiv\lambda x.\ \lambda y.\ f(x, y) \qquad (3.31)$$

任意の多変数関数に対して，このような操作を行う演算 κ は Curry 演算と呼ばれ，式(3.31)は次のように書ける．

$$g\equiv\kappa f \qquad (3.32)$$

式(3.31)の場合 $\kappa=\lambda f.\ \lambda x.\ \lambda y.\ f(x, y)$ となり，$f(x, y)\equiv\kappa f xy$ となることが知られる．κ の逆演算（逆 Curry 演算）ν というものを考えることもでき，$g\equiv\kappa f$ ならば $\nu\equiv\lambda g.\ \lambda(x, y).\ gxy$ と定義でき，$f\equiv\nu g$ となる．したがって $gxy\equiv\nu g(x, y)$ となることが知られる．

M と N を λ-項とし，x を変数としたとき，M において自由変数として現われるあらゆる x に対して N を代入した結果を $[N/x]M$ と書く．

λ-計算では

$$((\lambda x.\ \lambda y.\ x^2+2y^2)(1))(2)=((\lambda x.\ x^2+2)(2))=6$$
$$((\lambda y.\ \lambda x.\ x^2+2y^2)(1))(2)=((\lambda y.\ 1+2y^2)(2))=9 \qquad (3.33)$$

のように等号の左辺を右辺のように変換して計算できるが，等号の右辺に対して左辺が求まるわけではない．左辺から右辺へのような変換は還元 (reduction) と呼ばれる．λ-計算における基本的な還元規則 (α-規則，β-規則，η-規則) と Church-Rosser の定理について説明しておく．

① 直接還元 (immediate reduction)

- α-規則：$\lambda x.M\xrightarrow{\alpha}\lambda v.[v/x]M$
  ただし x は M の中で束縛されておらず，v は M の中に自由変数としても束縛変数としても現われないものとする．
- β-規則：$(\lambda x.M)N\xrightarrow{\beta}[N/x]M$
  ただし N のいかなる自由変数も M の束縛変数として現われないとする．
- ρ-規則：$M\xrightarrow{\rho}M$
  任意の λ-項 $\mathcal{A}$ が α-規則，β-規則，ρ-規則の1つを適用して $\mathcal{B}$ に変換できるとき $\mathcal{A}$ は (α-規則，β-規則，ρ-規則によって) $\mathcal{B}$ に直接還元できると言われ，$\mathcal{A}\to\mathcal{B}$ と記される．

② 還元 (reduction)

$\mathcal{A}\equiv M_1, M_1\to M_2, M_2\to M_3, \cdots, M_{n-1}\to M_n, M_n\equiv\mathcal{B}$

となるような有限系列 $M_1, M_2, \cdots, M_n$ が存在するとき，$\mathcal{A}$ は $\mathcal{B}$ に還元できると言われ $\mathcal{A}\Rightarrow\mathcal{B}$ と記される．なお，このとき $M_i\to M_{i+1}$ または $M_{i+1}\to M_i$ ならば $\mathcal{A}$ は $\mathcal{B}$ と等しいと言われ $\mathcal{A}=\mathcal{B}$ と書かれる．

〔Church-Rosser の定理〕*

$\mathcal{A}=\mathcal{B}$ ならば，$\mathcal{A}\Rightarrow\mathcal{C}$ かつ $\mathcal{B}\Rightarrow\mathcal{C}$ となる $\mathcal{C}$ が存在する．

λ-項 $\mathcal{A}$ が $(\lambda x.M)N$ なる形の式を含まなければ，$\mathcal{A}$ は (λ-) 標準形であると言われる．Church-Rosser の定理から，$\mathcal{A}=\mathcal{B}$ で $\mathcal{B}$ が λ-標準形であれば，$\mathcal{A}\Rightarrow\mathcal{B}$ となる．$\mathcal{A}=\mathcal{B}$ であれば $\mathcal{A}$ と $\mathcal{B}$ は同じ λ-標準を持つか，共に λ-標準形を持たない．また，$\mathcal{A}=\mathcal{B}$ であれば $\mathcal{A}$ が λ-標準形を持つことが知られていればその標準形を求めるアルゴリズムが存在することも知られている．λ-項には標準形を持たないものもある．たとえば，$(\lambda x.xx)\Rightarrow(\lambda x.xx)(\lambda x.xx)\Rightarrow\cdots$，$(\lambda x.xxy)(\lambda x.xxy)\Rightarrow(\lambda x.xxy)(\lambda x.xxy)y\Rightarrow(\lambda x.xxy)(\lambda x.xxy)yy\Rightarrow\cdots$

λ-計算は関数の適用 (application) と λ-抽象化 (λ-

---

* $(\lambda x.M)$ は x に関する λ-抽象化 (λ-abstraction) によって得られるとも言われる．

** (MN) は M の N への適用 (application) によって得られるとも言われる．

* 『$\mathcal{C}\Rightarrow\mathcal{A}$ かつ $\mathcal{C}\Rightarrow\mathcal{B}$ ならば $\mathcal{A}\Rightarrow\mathcal{C}$ かつ $\mathcal{B}\Rightarrow\mathcal{C}$ となる $\mathcal{C}$ が存在する』という形で Church-Rosser の定理を述べることもある．

abstraction) をもとに,

- $(\lambda x.\,A)a=[a/x]A$ (3.34 a)
- $\forall x[fx=gx]$ なら $f=g$ (3.34 b)

を公理として持つ体系である. 還元規則が λ-計算における証明手段として有用であることが Church-Rosser の定理より言える. 式(3.34 a)の公理は完全性の公理(または β-同値性) と言われ, 式(3.34 b)は外延性の公理 (axiom of extensionality) と呼ばれる*.

λ-計算を用いてプログラムの関数的意味記述を与えることができるが, そのまえに結合子論理の説明を行うことにする.

● 結合子論理 (Combinatory logic)

関数定義の概念における変数の使用を消去することを1つの目的として Schönfinkel によって導入され, Curry-Feys によって発展せられたのが結合子論理である.

結合子論理は基本的には結合子 (combinator) のみから構成されるものである. 基本的な結合子は次のような性質を持つ **I**, **K**, **S** である**:

$$\mathbf{I}x=x$$
$$\mathbf{K}xy=x$$
$$\mathbf{S}fgx=fx(gx) \qquad (3.35)$$

結合子論理における抽象化は (λx と書くかわりに) $[x]$ と書かれる. 結合子論理の項を C-項と呼び, それは変数か定数か基本結合子 (**I**, **K**, **S**) であるか, M と N を C-項としたときに (MN) は C-項であるという形で定義されるものとする. このとき C-項の還元$\Rightarrow$を次のように定義する:

$$([x]M)N\Rightarrow[N/x]M \qquad (3.36)$$

したがって, 抽象化演算を用いると,

① $[x]x\equiv\mathbf{I}$

② $[x]M\equiv\mathbf{K}M$ $(x\notin M)$

③ $[x]Mx\equiv M$ $(x\notin M)$

④ $[x](MN)\equiv\mathbf{S}([x]M)([x]N)$ (3.37)

(ただし②も③も適用できないとき)

すなわち, 任意の C-項 X, Y, Z に対して

$$X\Rightarrow X$$
$$\mathbf{I}X\Rightarrow X$$
$$\mathbf{K}XY\Rightarrow X$$
$$\mathbf{S}XYZ\Rightarrow XZ(YZ) \qquad (3.38)$$

また, $X\Rightarrow Y$ なら $ZX\Rightarrow ZY$, $X\Rightarrow Y$ なら $XZ\Rightarrow YZ$, $X\Rightarrow Y$ かつ $Y\Rightarrow Z$ なら $X\Rightarrow Z$ も成り立つ. (なお, $X\Rightarrow Y$ がこれらの性質および式 (3.38) の4つの性質から言えるとき, $X=Y$ と記す.) 式(3.38)の$\Rightarrow$の左辺の $\mathbf{I}X$, $\mathbf{K}XY$, $\mathbf{S}XYZ$ が **redex** と呼ばれ, 右辺の X, X, $XZ(YZ)$ がそれぞれ **contractum** と呼ばれる. C-項 **X** は **redex** を含まないときに (C-) 標準形であると言われ, Y が (C-) 標準形をした X に還元されるなら X は Y の (C-) 標準形であると言われる.

後述の例で知られるように結合子として次のような **B** や **C** を用いると便利なことがある.

$$\mathbf{B}fgx=f(gx)$$
$$\mathbf{C}fgx=fxg \qquad (3.39)$$

結合子として **I**, **K**, **S**, **B**, **C** を導入したが, これらの間には次のような関係が成り立つことが知られている.

① $\mathbf{I}\equiv\mathbf{SKK}$

($\because$) $\mathbf{SKK}X\Rightarrow\mathbf{K}X(\mathbf{K}X)\Rightarrow X$

② $\mathbf{B}\equiv\mathbf{S}(\mathbf{KS})\mathbf{K}$

($\because$) $\mathbf{S}(\mathbf{KS})\mathbf{K}XYZ\Rightarrow\mathbf{KS}X(\mathbf{K}X)YZ$
$\Rightarrow\mathbf{S}(\mathbf{K}X)YZ\Rightarrow\mathbf{K}XZ(YZ)\Rightarrow X(YZ)$

③ $\mathbf{C}\equiv\mathbf{S}(\mathbf{BBS})(\mathbf{KK})$

結合子論理においてもλ-計算におけると同様の定理が成り立つ:

〔結合子論理の基本定理〕

(1) $Z\Rightarrow X_1$ かつ $Z\Rightarrow X_2$ ならば, $X_1\Rightarrow Y$ かつ $X_2\Rightarrow Y$ となるような Y が存在する.

(2) $X=Y$ ならば, $X\Rightarrow Z$ かつ $Y\Rightarrow Z$ となる Z が存在する.

(3) $X=Y$ で Y が (C-) 標準形であれば, $X\Rightarrow Y$

(4) $X=Y$ であれば, X と Y は同じ標準形を持つかともに標準形を持たない.

これまでに述べてきた結合子以外に, 次のようなものが使われることがある:

$\mathbf{W}xy=xyy$ (すなわち $\mathbf{W}=\mathbf{SS}(\mathbf{SK})$)

$\mathbf{J}uxyz=ux(uzy)$

$\boldsymbol{\Phi}xyzu=x(yu)(zu)$

$\boldsymbol{\Psi}xyuv=x(yu)(yv)$

$\mathbf{Y}_0\equiv\mathbf{WS}(\mathbf{BWB})$

$\mathbf{Y}_1\equiv\mathbf{WI}(\mathbf{B}(\mathbf{SI})(\mathbf{WI}))$

$\mathbf{Y}_{n+1}\equiv\mathbf{Y}_n(\mathbf{SI})$

ここに $\mathbf{Y}_0, \mathbf{Y}_1, \mathbf{Y}_2, \cdots$ は

---

* 関数の外延性は公理としているが, λ-項の外延性『$Mx=Nx$ なら $M=N$』は, 一般には成立しないことに注意せよ. たとえば $M\equiv\lambda x.yx$, $N\equiv y$ としたとき任意の X に対して $MX=NX$ となるが $M\neq N$ である.

** 括弧を用いると $\mathbf{I}(x)=x$, $(\mathbf{K}(x))(y)=x$, $(\mathbf{S}(f,g))(x)=f(x,g(x))$ と書いてもよい.

$$\mathbf{Y}x=x(\mathbf{Y}x) \tag{3.40}$$

なる性質を満足する結合子であり，これらは不動点結合子 (Fixed point combinator) と呼ばれる*.

●λ-計算と結合子論理の関係

λ-計算と結合子論理は形式化は異なるが，論理的には同じ記述能力を持つことが下記のように示される.

まず，結合子論理が λ-計算によって表現できることが次のように示される．任意の結合子 a に対するλ-記法を a_λ とする．このとき，①a が **I**, **K**, **S** でない項としたとき，$a_\lambda=a$, ②$\mathbf{I}_\lambda=\lambda x.\,x$, ③$\mathbf{K}_\lambda=\lambda x\,\lambda y.\,x$, ④$\mathbf{S}_\lambda=\lambda x\lambda y\lambda z.\,xz(yz)$, ⑤$(XY)_\lambda=X_\lambda Y_\lambda$.

λ-計算が結合子論理によって表現できることは次のように示される．任意の λ-項 a に対する結合子表記を a_c とする．①a が変数か定数ならば $a_c=a$, ②$(XY)_c=X_cY_c$, ③$(\lambda x.\,X)_c=[x]X_c$, ここに $[x]$ は結合子論理における抽象化演算で結合子との対応は式(3.37)のように与えられるものとする.

λ-計算では変数への**代入** (substitution) が本質的であるが，結合子論理では変数を消去でき，代入操作の解析，関数抽象化の概念を定義できるからより基礎的であると言える.

式(3.40)を満たす不動点結合子 $\mathbf{Y}_0$ と $\mathbf{Y}_1$ を λ-記法で書くと次のように表わせる.

$$\mathbf{Y}_0:\ \lambda x.\,(\lambda y.\,x(yy))(\lambda y.\,x(yy))$$
$$\mathbf{Y}_1:\ (\lambda xy.\,y(xxy))(\lambda xy.\,y(xxy)) \tag{3.41}$$

自然数 n の階乗 **fact**(n) は λ-計算と結合子論理を用いて次のように表わせる.

$$\mathbf{fact}=\lambda n.\,[\mathbf{if}\ n=0\ \mathbf{then}\ 1\ \mathbf{else}\ n\times\mathbf{fact}(n-1)]$$
$$\mathbf{fact}=\mathbf{S}(\mathbf{C}(\mathbf{B}\ \mathbf{cond}\ (\mathbf{eq}\ 0))1)(\mathbf{S}\ \mathbf{times}\ (\mathbf{B}\ \mathbf{fact}\ (\mathbf{C}\ \mathbf{sub1}))) \tag{3.42}$$

ここに **cond**, **eq**, **times**, **sub1** は条件式，=，×，−1 に対応する Curry 関数とする**.

λ-計算および結合子論理ともに関数型システムであるが，次の2つの基本的性質が要求されていることを強調しておきたい.

[combinatory completeness]

任意の項 A に対して，

$$\exists f\,\forall x_1\cdots x_n.\ f\,x_1\cdots x_n=\mathrm{A} \tag{3.43}$$

[Extensionality]

$$\forall x.\,(fx=gx)\ \text{ならば}\ f=g \tag{3.44}$$

3.4 関数型論理とプログラムの意味

関数型論理を用いてプログラムの意味（すなわち計算）記述が行えることが式(3.42)あるいはこれまでの説明からも知られるが，ここでは具体例によって説明を行うことを考える．まず，結合子論理を用いることによって原始帰納的関数が表現できることの説明から始めよう.

●原始帰納的関数の結合子論理による表現

計算機科学に現われる関数の多くは原始帰納的関数 (primitive recursive function) であることが知られているが，ここでは原始帰納的関数が結合子論理によってどのように表現されるかを示す．原始帰納的関数は次のように定義されるものであったことを思い出そう*:

① 定数 0 は原始帰納的である

② 後続者関数 (successor function) **succ** は原始帰納的である

③ 任意の $x_1, x_2, \cdots, x_n$ に対して

$$\mathbf{u}_{m,n}(x_1,\cdots,x_n)=x_m\quad(1\leqslant m\leqslant n) \tag{3.45 a}$$

なら $\mathbf{u}_{m,n}$ は原始帰納的である

④ $g(x_1,\cdots,x_n)$, $f_1(x_1,\cdots,x_m),\cdots,f_n(x_1,\cdots,x_m)$ が原始帰納的であるとき，

$$h(x_1,\cdots,x_m)=g(f_1(x_1,\cdots,x_m),\cdots,f_n(x_1,\cdots,x_m)) \tag{3.45 b}$$

は原始帰納的である

⑤ $f(x_1,\cdots,x_n)$ と $g(y,z,x_1,\cdots,x_n)$ が原始帰納的で

$h(0,x_1,\cdots,x_n)=f(x_1,\cdots,x_n)$

$h(k+1,x_1,\cdots,x_n)=g(k,h(k,x_1,\cdots,x_n),x_1,\cdots,x_n)$ のとき，h は原始帰納的である (3.45 c)

このような原始帰納的関数を結合子論理によって表現するためには自然数を結合子によって表現する必要がある.

① $\mathbf{Z}_0=\mathbf{KI}$ によって "0"

$\mathbf{Z}_n=(\mathbf{SB})^n(\mathbf{KI})$ によって "n" (3.46)

を表現すると，後続者関数 **succ** は

② $\widetilde{\mathbf{succ}}=\mathbf{SB}$ (3.47)

と定義される**. また

③ $\widetilde{\mathbf{u}}_{m,n}=[x_1,x_2,\cdots,x_n]x_m$ (3.48)

---

* 結合子論理においては $\mathbf{Y}_0\neq\mathbf{Y}_1$ であるが，次節で説明する Scott のプログラム束の理論の特徴の1つは $\mathbf{Y}_0=\mathbf{Y}_1$ となる（λ-計算の）モデルを与えたことであるとも言える．($\mathbf{Y}_0$ と $\mathbf{Y}_1$ の λ-記法による表現に関しては式 (3.41) を参照.)

** λ-計算による **fact** の定義においても厳密にはすべての要素や演算を λ-記法で表現してやる必要があるが，ここでは省略した.

* 計算の理論においてよく知られているように，原始帰納的関数は計算可能である.

** $\widetilde{\mathbf{succ}}$ は **succ** の結合子論理表現であることを意味しており，$\widetilde{\mathbf{u}}_{m,n}$, $\widetilde{\bar{k}}$ などについても同様であるとする．したがって $\widetilde{\bar{0}}=\mathbf{KI}$, $\widetilde{\bar{1}}=\mathbf{SBKI}$ である.

④ $\tilde{h}=[x_1, x_2, \cdots, x_n](\tilde{g}(\tilde{f}_1 x_1 \cdots x_m)\cdots(\tilde{f}_n x_1 \cdots x_m))$ (3.49)

ここに $\tilde{g}, \tilde{f}_1, \cdots, \tilde{f}_n$ は $g, f_1, \cdots, f_n$ の結合子論理表現とする.

⑤ $\mathbf{D}=[x, y, z]z(\mathbf{K}y)x$

$\mathbf{Q}=[y, v]\mathbf{D}(\mathbf{SB}(v\mathbf{Z}_0))(y(v\mathbf{Z}_0)(v\mathbf{Z}_1))$

$\mathbf{R}=[x, y, u]u(\mathbf{Q}y)(\mathbf{DZ}_0 x)\mathbf{Z}_1$ (3.50)

としたとき, この $\mathbf{R}$ は

$\mathbf{R}XY\mathbf{Z}_0=X$

$\mathbf{R}XY\mathbf{Z}_{k+1}=Y\mathbf{Z}_k(\mathbf{R}XY\mathbf{Z}_k)$ (3.51)

なる関係を満たす. この $\mathbf{R}$ を用いると

$$\begin{aligned}\tilde{h}\mathbf{Z}_0X_1\cdots X_n&=\mathbf{R}(\tilde{f}X_1\cdots X_n)(Y^*X_1\cdots X_n)\mathbf{Z}_0\\&=\tilde{f}X_1\cdots X_n\\ \tilde{h}\mathbf{Z}_{k+1}X_1\cdots X_n&=\mathbf{R}(\tilde{f}X_1\cdots X_n)(Y^*X_1\cdots X_n)\mathbf{Z}_{k+1}\\&=Y^*X_1\cdots X_n\mathbf{Z}_k(\tilde{h}\mathbf{Z}_kX_1\cdots X_n)\\&=\tilde{g}\mathbf{Z}_k(\tilde{h}\mathbf{Z}_kX_1\cdots X_n)X_1\cdots X_n\end{aligned}$$ (3.52)

ここに $Y^*=[x_1, x_2, \cdots, x_n, u, v]\tilde{h}uvx_1\cdots x_n$ とする.

原始帰納的関数が結合子論理を用いて表現できることは, 上述のようにして示されるが, 部分帰納的関数 (partial recursive function) を結合子論理を用いて表現できることも知られており, あらゆる計算可能な関数が結合子論理によって表現できることになる. 結合子は λ-記法によって表現できるから, 以上の議論は λ-計算の場合にも言える. (λ-計算を用いて行った方がより直観的に分かりやすいが, これについてはプログラム束の理論の際に説明する.)

λ-計算を基礎とするプログラミング言語としてリスト処理言語 LISP があるが, 結合子論理をもとにしたプログラミング言語を考えることができる. 結合子論理の表現は直観的には分かり難い面もあるが, 束縛変数を全て消去できるから実行効率のよい処理系が実現できる可能性がある*.

● λ-計算によるプログラミング言語の意味記述

λ-計算によるプログラミング言語の意味記述がどのように行えるかを例によって示そう. プログラムの意味を状態ベクトルの関数表現から状態ベクトルの関数表現への写像を行う関数として与えることを考える. なお, 状態ベクトルはプログラム変数によって作られ, 変数は変数宣言の順に並べられるものとしておく. プログラム変数 $x_1, \cdots, x_n$ によって作られる状態ベクトルを $\boldsymbol{x}=(x_1, \cdots, x_n)$ とし, その関数表現を $\varphi x_1\cdots x_n$ とする.

状態ベクトル $\boldsymbol{x}=(x_1, \cdots, x_n)$ の下での文 S の λ-計算による意味を $\hat{s}.[S](\boldsymbol{x})$ によって表わすとする. このとき, 代入文, 複合文, 条件文, 繰返し文の意味記述は次のように与えられる:

- 代入文: $x_i \leftarrow \epsilon$

$\hat{s}.[x_i\leftarrow\epsilon](\boldsymbol{x})=[\hat{\epsilon}/x_i]\varphi x_1\cdots x_n$ (3.53 a)

- 複合文: S_1; S_2

$\hat{s}.[S_1;\ S_2](\boldsymbol{x})=\lambda\varphi x_1\cdots x_n.\ \hat{S}_1\hat{S}_2\varphi x_1\cdots x_2$ (3.53 b)

- 条件文: **if** p **then** S_1 **else** S_2

$\hat{s}.[\textbf{if } p \textbf{ then } S_1 \textbf{ else } S_2](\boldsymbol{x})$

$=\lambda\varphi x_1\cdots x_n.\ (\textbf{cond}\ [\hat{p}\varphi x_1\cdots x_n,\ \hat{S}_1\varphi x_1\cdots x_n,\ \hat{S}_2\varphi x_1\cdots x_n])$

- 繰返し文: **while** p **do** S

$\hat{s}.[\textbf{while } p \textbf{ do } S](\boldsymbol{x})$

$=\mathbf{Y}(\lambda f.\hat{s}.[\textbf{if } p \textbf{ then } f(S) \textbf{ else } e](\boldsymbol{x}))$ (3.54)

ここに $\hat{\epsilon}, \hat{p}, \hat{S}_1, \hat{S}_2$ はそれぞれ ϵ, p, S_1, S_2 に対する λ-記法による表現とし, e は $\hat{s}.[e](\boldsymbol{x})=\varphi x_1\cdots x_n$ なる単位演算子とする. ϵ と p に対する λ-記法は原始帰納的関数の結合子論理による表現を行ったと同様の仕方で与えられるものとする. なお, **cond**$[\alpha, S_1, S_2]$ は α が真なら S_1, 偽なら S_2 を取る条件文であるとする.

代入文の系列, たとえば,

$x\leftarrow a;\ y\leftarrow x+b;\ x\leftarrow y-a$ (3.55 a)

に対しては, 次のように backward に評価されるから注意せよ*.

$$\begin{aligned}&[a/x]([x+b/y]([y-a/x]\varphi xy))\\&=[a/x]([x+b/y]\varphi(y-a)y)\\&=[a/x](\varphi(x+b-a)(x+b))\\&=\varphi(a+b-a)(a+b)\\&=\varphi b(a+b)\end{aligned}$$ (3.55 b)

上述の意味記述では, (入力) 状態ベクトル $(x_1, \cdots, x_n)$ に対する関数表現 $\varphi x_1\cdots x_n$ を求め, 文 S の意味を $\varphi x_1\cdots x_n$ を $\varphi a_1\cdots a_n$ に変換する関数として与えるのである. λ-計算を用いこのような仕方によってプログラミング言語のほかの構成要素の意味記述も与えることができるが, プログラム束の理論の応用として同様の問題を論じるから, ここではこれ以上言及し

* このような提案を昭和48年度電気四学連合大会で (筆者が) 行ったことがある.

* 応用として, この章の冒頭で考察した n の階乗のプログラムの意味が計算によってどのように与えられるかを調べよ.

ない.

〔コメント〕

プログラムの状態ベクトルによる意味記述は，第2章でも述べたように，Kaluzhnin [1959] に始まるが，代入関数および内容関数による本論文での記述はMcCarthy [1963] に負うものである．再帰的帰納法による証明やプログラミング言語の状態ベクトル・セマンティクスも McCarthy [1963; 1965] に負う．プログラム代数 K は，Yanov や McCarthy の研究に影響を受けながら1960年代に行われた Glushkov [1965] や Ito [1970] の研究を de Bakker [1971] を参考にしてまとめたものである．プログラムの正当性に関する事柄については，その後，Scott のプログラム束の理論の応用として同様な議論が行われ，これについては後述する．λ-計算と結合子論理に関する事柄は Strachey [1965] と Hindley 他 [1972] を参考にして説明を行い，例を与えた．

なお，この章の内容は九州大学での講義資料と昭和54年1月に行われた人工知能の研究集会での筆者の講演資料をもとにまとめたものである．

参考文献

1) Kaluzhnin, L. A.: Algorithmization of mathematical problems, Problems of Cybernetics 2, pp. 371-391 (1959).
2) McCarthy, J.: Towards a mathematical science of computation, IFIP '62, pp. 21-28 (1963).
3) McCarthy, J.: A formal description of a subset of ALGOL, Formal Language Description Languages. for Computer Programming, pp. 1-12., ed. T. B. Steel, North-Holland (1966).
4) Strachey, C.: Towards a formal semantics, Formal Language Description Languages for Computer Programming, pp. 198-220, ed. T. B. Steel, North-Holland (1966).
5) Glushkov, V. M.: Automata theory and formal microprogram transformations, Cybernetics 1, pp. 1-8 (1965).
6) Ito, T.: A theory of formal microprograms, International Advanced Study Institute on Microprogramming, ed. G. Boulaye and J. Mermet, Herman (1970), Also, in ACM SIGMICRO Newsletter (1971). (the reproduction of the same paper by invitation)
7) de Bakker, J. W.: Recursive Procedures, Mathematical Centre, Amsterdam (1971).
8) Hindley, J. R., et al.: Introduction to Combinatory Logic, Cambridge University Press (1972).

(昭和56年7月14日受付)