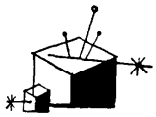
講　座



プログラム理論とその応用（4）†

伊　藤　貴　康††

5. プログラム束の理論の応用

プログラム束の理論（D_∞ モデル）の基本公理といくつかの基礎的な性質について第4章で述べ，特に，B_∞ が "universal" な性質を持つ空間であることを説明した．この章では，先ず，B_∞ 空間において定義される関数族を記述するのに適した言語 **LAMBDA** を与え，B_∞ 空間において計算可能な関数族と言語 **LAMBDA** によって定義可能な関数族が一致するという Scott の結果について説明する．この結果は，単純な言語 **LAMBDA** がプログラムの意味記述用言語として十分な表現能力を持つことを示している．

プログラム束の理論に基づくプログラミング言語の意味記述とプログラムの性質の証明についても説明を行う予定である．

5.1 言語 LAMBDA とその性質

B_∞ 空間が universal な空間であるとみなせることを述べたが，この事をより実際的に示すために，B_∞ 空間において計算可能な関数族を記述できる言語 **LAMBDA** について説明する．この言語は，Scott と Milner によって考案された type-free LCF とも呼ばれるものであり，λ-計算と pure LISP を合成した単純な言語である*．以下では，この言語 **LAMBDA** のシンタックス，セマンティクス，公理系および計算可能性問題について述べる**．

5.1.1 LAMBDA のシンタックスとセマンティクス

言語 **LAMBDA** のシンタックスとセマンティクスは次のように与えられる．

（1） **LAMBDA** のシンタックス

言語 **LAMBDA** で記述される式 (expression) E は，BNF 記法を用いると次のように定義される：

$$\xi ::= \langle \text{identifier} \rangle$$
$$E ::= \xi \mid \bot \mid \top \mid \boldsymbol{t} \mid \boldsymbol{f} \mid \supset \mid \lambda\xi.\, E \mid EE \mid (E) \qquad (5.1)$$

（2） **LAMBDA** のセマンティクス

identifier の集合を **Id**，expression の集合を **Exp** としたとき，expression に対して B_∞ 空間での意味を与える意味関数 $\tilde{\boldsymbol{E}}$ は次のように定義される：

$$\tilde{\boldsymbol{E}} : \mathbf{Exp} \to [[\mathbf{Id} \to \boldsymbol{B}_\infty] \to \boldsymbol{B}_\infty] \qquad (5.2)$$

なお

$$\sigma : \mathbf{Id} \to \boldsymbol{B}_\infty \qquad (5.3)$$

を environment と呼ぶ．environment は identifier に値を結合させるメカニズムをモデル化したものであるが，これを用いると $\tilde{\boldsymbol{E}}$ は再帰的に次のように与えられる：

$$\tilde{\boldsymbol{E}}[\xi](\sigma) = \sigma(\xi)$$
$$\tilde{\boldsymbol{E}}[\bot](\sigma) = \bot$$
$$\tilde{\boldsymbol{E}}[\top](\sigma) = \top$$
$$\tilde{\boldsymbol{E}}[\boldsymbol{t}](\sigma) = \boldsymbol{t}$$
$$\tilde{\boldsymbol{E}}[\boldsymbol{f}](\sigma) = \boldsymbol{f}$$
$$\tilde{\boldsymbol{E}}[\supset](\sigma) = \lambda p.\, \lambda x.\, \lambda y.\, (p \supset x, y)$$
$$\tilde{\boldsymbol{E}}[\lambda\xi.\, E](\sigma) = \lambda x.\, \tilde{\boldsymbol{E}}[E](\sigma[x/\xi])$$
$$\tilde{\boldsymbol{E}}[E_1 E_2](\sigma) = (\tilde{\boldsymbol{E}}[E_1](\sigma))(\tilde{\boldsymbol{E}}[E_2](\sigma))$$
$$\tilde{\boldsymbol{E}}[(E)](\sigma) = \tilde{\boldsymbol{E}}[E](\sigma) \qquad (5.4)$$

ここに，① $\sigma(\xi)$ は identifier ξ の B_∞ における値；$\sigma[x/\xi]$ は $\sigma(\xi)$ の ξ を x に置き換えて得られる environment.

② $(p \supset x, y)$ は条件式 (conditional expression) で

† 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

* ここに紹介する言語 **LAMBDA** は，Scott-Milner による LCF (Logic for Computable Functions) の type-free version である．

** プログラム束の理論と計算可能性問題を密接に関連付けるために導入されたモデルとして P_∞ モデルがある．Scott [1976] では P_∞ モデルにおける言語 **LAMBDA** を次のように定義している：

$$0 = \{0\}$$
$$x+1 = \{n+1 \mid n \in x\}$$
$$x-1 = \{n \mid n+1 \in x\}$$
$$x \supset x, y = \{n \in x \mid 0 \in z\} \cup \{m \in y \mid \exists k.\, k+1 \in z\}$$
$$u(x) = \{m \mid \exists e_n \subseteq x.\, (n, m) \in u\}$$
$$\lambda x.\, \tau = \{(n, m) \mid m \in \tau[e_n/x]\}$$

ここで述べる D_∞ モデルにおける言語 **LAMBDA** は **integer-free** の形式であることにも注意されたい．（このために，計算可能性問題を論じるのが若干困難となる．）

あり，次のように定義されるとする：

$$(p\supset x, x)=\begin{cases} x\sqcup y & (p=\top \text{ のとき}) \\ x & (\boldsymbol{t}\sqsubseteq p\neq\top \text{ のとき}) \\ y & (\boldsymbol{f}\sqsubseteq p\neq\top \text{ のとき}) \\ \bot & (\text{上記以外のとき}) \end{cases} \quad (5.5)$$

[注] 言語 **LAMBDA** における条件式 $\supset(p)(x)(y)$ のかわりに，式 (5.5) の記法 $(p\supset x, y)$ を用いた．

（3） **LAMBDA** の論理式の意味

LAMBDA で書かれた expression E_1 と E_2 に対して，次の2つの論理式を考える：

$$E_1\sqsubseteq E_2$$
$$E_1=E_2 \quad (5.6)$$

これらの論理式が成り立つ——真となる——のは次のときである：

① $E_1\sqsubseteq E_2$ が environment σ において真

$$\Updownarrow$$
$$\tilde{\boldsymbol{E}}[E_1](\sigma)\sqsubseteq\tilde{\boldsymbol{E}}[E_2](\sigma) \quad (5.7\,a)$$

② $E_1=E_2$ が environment σ において真

$$\Updownarrow$$
$$\tilde{\boldsymbol{E}}[E_1](\sigma)=\tilde{\boldsymbol{E}}[E_2](\sigma) \quad (5.7\,b)$$

5.1.2 LAMBDA の公理系——Type-free LCF

言語 **LAMBDA** の性質を式 (5.6) に与えた **LAMBDA** における論理式の公理系を与えることによって説明しよう．以下では，この公理系を LAMBDA 論理と呼ぶことにする．

（1） LAMBDA 論理における論理式と図式

LAMBDA 論理における**論理式**は，上述のように，

$$E_1\sqsubseteq E_2$$
$$E_1=E_2 \quad (5.8\,a)$$

のいずれかである．$\mathcal{A}$ および $\mathcal{B}$ を，それぞれ，論理式の有限集合としたときに，

$$\mathcal{A}\vdash\mathcal{B} \quad (5.8\,b)$$

は図式 (sequent) と呼ばれる．

論理式の意味は式 (5.7) に与えたが，図式の恒真性 (validity) は次のように定義される．図式 $\mathcal{A}\vdash\mathcal{B}$ が**恒真**であるとは，あらゆる environment σ に対して，$\mathcal{A}$ のあらゆる論理式が environment σ で真ならば，この environment σ において $\mathcal{B}$ の論理式が少なくとも1つは真となることである．

Scott-Milner の公理系は Gentzen の論理体系に基づいて形式的に与えられているが，ここでは，プログラム束の理論に対する公理系としての LAMBDA 論理の直感的な説明を与えるに留める．

（2） LAMBDA 論理の公理と推論規則

等号に関する公理と変数への代入規則（および Gentzen の証明図に関する計算規則）が使えるという前提の下に，次のような公理が成り立つ：

（I） 半順序と束の公理

① $\boldsymbol{t}\sqsubseteq\bot\vdash$

② $\boldsymbol{f}\sqsubseteq\bot\vdash$

③ $\vdash\bot\sqsubseteq x$

④ $\vdash x\sqsubseteq\top$

⑤ $\vdash x\sqsubseteq x$

⑥ $x\sqsubseteq y,\ y\sqsubseteq x\vdash x=y$

⑦ $x\sqsubseteq y,\ y\sqsubseteq z\vdash x\sqsubseteq z$

⑧ $\boldsymbol{t}\sqsubseteq x,\ \boldsymbol{f}\sqsubseteq x\vdash\top\sqsubseteq x$

⑨ $x\sqsubseteq\boldsymbol{t},\ x\sqsubseteq\boldsymbol{f}\vdash x\sqsubseteq\bot$

⑩ $\vdash x\sqsubseteq x\sqcup y$

⑪ $\vdash y\sqsubseteq x\sqcup y$

⑫ $x\sqsubseteq z,\ y\sqsubseteq z\vdash x\sqcup y\sqsubseteq z$

⑬ $\vdash x\sqcap y\sqsubseteq x$

⑭ $\vdash x\sqcap y\sqsubseteq y$

⑮ $z\sqsubseteq x,\ z\sqsubseteq y\vdash z\sqsubseteq x\sqcap y$

[注] "$\boldsymbol{t}\sqsubseteq\bot\vdash$" は "$\boldsymbol{t}\sqsubseteq\bot$" が矛盾であることを示している．"$\boldsymbol{f}\sqsubseteq\bot\vdash$" についても同様である．

（II） 条件式の公理

① $\boldsymbol{t}\sqsubseteq p\vdash(p\supset x, y)=x,\ p=\top$

② $\boldsymbol{f}\sqsubseteq p\vdash(p\supset x, y)=y,\ p=\top$

③ $\vdash(p\supset x, y)=\bot,\ \boldsymbol{t}\sqsubseteq p, \boldsymbol{f}\sqsubseteq p$

④ $\vdash(\top\supset x, x)=x$

⑤ $\vdash(\top\supset x, y)=x\sqcup y$

（III） 関数適用と抽象化の公理

① $\vdash\boldsymbol{t}(x)=\boldsymbol{t}$

② $\vdash\boldsymbol{f}(x)=\boldsymbol{f}$

③ $\vdash(\lambda x.\,E)(x)=E$

④ $x\sqsubseteq y\vdash f(x)\sqsubseteq f(y)$

⑤ $\dfrac{\mathcal{A}\vdash f(x)\sqsubseteq g(x),\ \mathcal{B}}{\mathcal{A}\vdash f\sqsubseteq g,\ \mathcal{B}}$

ここに x は $\mathcal{A}$ および $\mathcal{B}$ に自由変数としては現われないとする．

（IV） 不動点演算子の公理

① $\vdash f(\mathbf{Y}(f))=\mathbf{Y}(f)$

② $\mathbf{I}=\mathbf{Y}(\lambda x.\,\mathbf{T}\sqcup(x\rightarrow x))$

すなわち，

$\mathbf{T}\sqsubseteq x, (x\rightarrow x)\sqsubseteq x\vdash\mathbf{I}\sqsubseteq x$

③* $\mathcal{A}\vdash f(\bot)=g(\bot),\ \mathcal{B}$

---

* この推論規則は $\dfrac{\mathcal{A}\vdash\mathcal{B}[\bot/x]\quad \mathcal{A}, \mathcal{B}[x]\vdash\mathcal{B}[F(x)]}{\mathcal{A}\vdash\mathcal{B}(\mathbf{Y}(F))}$ と記されることもある．

$$\frac{\mathcal{A}, f(x)=g(x)\vdash f(F(x))=g(F(x)),\ \mathcal{B}}{\mathcal{A}\vdash f(\mathbf{Y}(F))=g(\mathbf{Y}(F))}$$

ここに x は $\mathcal{A}$ および $\mathcal{B}$ に自由変数として現われないとする．

ここに **Y**, **T**, ➡ および **I** は次のように定義される演算子である．

$$\mathbf{Y}=\lambda f.\ (\lambda x.\ f(x(x)))(\lambda x.\ f(x(x)))$$
$$\mathbf{T}=\lambda p.\ (p\supset \boldsymbol{t}, \boldsymbol{f})$$
$$\mathbf{➡}=\lambda a.\ \lambda b.\ \lambda f.\ \lambda x.\ b(f(a(x)))$$
$$\mathbf{I}=\lambda x.\ x$$

5.1.3 LAMBDA 定義可能な関数と計算可能性

LAMBDA によって $\boldsymbol{B}_\omega$ 空間のほとんどすべての性質を言語的に表現することができると考えられるが，実際，Scott は $\boldsymbol{B}_\omega$ 空間において計算可能な関数の族が **LAMBDA** によって定義される関数の族と一致することを示した．

定義 [LAMBDA 定義可能な関数]

$\boldsymbol{B}_\omega$ 空間の要素 z が LAMBDA 定義可能であると言われるのは，あらゆる environment σ の下で，

$$z=\tilde{\boldsymbol{E}}[E](\sigma) \tag{5.9}$$

となる **LAMBDA** の expression E が存在することである．ここに E は自由変数は含まないとする．Scott の仮説 (Scott's thesis) を次のように述べることができる：

[Scott の仮説]

$\boldsymbol{B}_\omega$ 空間において計算可能な関数の族は，LAMBDA 定義可能な関数の族と一致する．

この仮説が成り立つことを示すには，自然数を **LAMBDA** によって表現し，あらゆる部分帰納的関数 (partial recursive function) が LAMBDA 定義可能であることを示す必要がある．この概要を説明するために，まず，いくつかの定義を与える．

$$(x, y)=\lambda p.\ (p\supset x, y)$$
$$\mathbf{first}=\lambda u.\ u(\boldsymbol{t})$$
$$\mathbf{second}=\lambda u.\ u(\boldsymbol{f}) \tag{5.10}$$

自然数と基本的な演算を次のように定義する：

$$\bar{0}=(\boldsymbol{t}, \bot)$$
$$\overline{n+1}=(\boldsymbol{f}, \bar{n})$$
$$\mathbf{test}=\lambda n.\ (\mathbf{first}\ (n)\supset \boldsymbol{t},\ \mathbf{test}\ (\mathbf{second}\ (n)))$$
$$\boldsymbol{v}=\lambda n.\ (\mathbf{test}\ (n)\supset n, \bot)$$
$$\mathbf{suc}=\lambda n.\ \boldsymbol{v}\ (\boldsymbol{f}, \boldsymbol{v}\ (n))$$
$$\mathbf{pred}=\lambda n.\ \mathbf{second}\ (\boldsymbol{v}\ (n))$$
$$\mathbf{zero}=\lambda n.\ \mathbf{first}\ (\boldsymbol{v}\ (n))$$
$$\mathbf{cond}=\lambda x.\ \lambda y.\ \lambda p.\ (p\supset x, y)$$
$$\mathbf{I}=\lambda x.\ x$$
$$\mathbf{K}=\lambda x.\ \lambda y.\ x$$
$$\mathbf{S}=\lambda u.\ \lambda v.\ \lambda x.\ u(x)(v(x)) \tag{5.11}$$

これらの定義を用いることにより，λ-計算の理論におけると同様の手法を用いて，自然数に関するあらゆる部分帰納的関数が LAMBDA 定義可能であることが知られる．LAMBDA 定義可能な関数が計算可能であることを示すのはより困難であるが*，その考え方を簡単に説明しておく．z が LAMBDA 定義可能ならば

$$z=\bigsqcup_{n=0}^{\infty} e^{(n)} \tag{5.12}$$

と書ける．$e^{(n)}$ は有限空間 $\boldsymbol{B}_n$ の計算可能な要素であると考えてよい．したがって，有限的な要素——すなわち，孤立点——の極限が計算可能であることを示せばよい．$\boldsymbol{B}_\omega$ の孤立点 (isolated point) は $\{\bot, \boldsymbol{f}, \boldsymbol{t}, \top\}$ から2つの演算 $e\sqcup e^1$ と $\vec{e}(e, e^1)$ を反復的に用いることにより enumeration できることから示せる．ここに，

$$\vec{e}(e, e^1)(x)=\begin{cases} e^1 & (e\sqsubseteq y\ \text{のとき}) \\ \bot & (\text{上記の以外とき}) \end{cases} \tag{5.13}$$

実際，$f\in \boldsymbol{B}_{n+1}$ は

$$f=\bigsqcup\{\vec{e}(e, f(x))\mid e\in \boldsymbol{B}_n\} \tag{5.14}$$

と表現される．

［コメント］

言語 **LAMBDA** のシンタックス，セマンティクスおよび LAMBDA 論理の公理系について説明してきたが，プログラム束の理論の応用という観点からは，**第4章**で述べたモデル論的な性質に立ち入ることなく LAMBDA 論理を用いてプログラムの意味を論じることができると考えてよい．

LAMBDA 論理の公理系の恒真性 (validity) は，**第4章**の議論から明らかであろう．LAMBDA 論理の演習問題として，次の命題が成り立つことを示せ．

$$f=\mathbf{Y}(F), g=F(g)\vdash f\sqsubseteq g$$
$$F\sqsubseteq G, x\sqsubseteq y\vdash F(x)\sqsubseteq G(y) \tag{5.15}$$

プログラム束の理論の応用に当っては，$\boldsymbol{D}_\omega$ 空間の特定の部分空間について論じることが要求される．これはレトラクションを用いて行われる．いまデータタイプ A と B に対するレトラクションをそれぞれ a と b とする．このとき，$A\times B$, $A+B$ および $A\to B$ に対応するレトラクション $\otimes$, $\oplus$ および $\Rightarrow$ はそれぞれ次のよ

---

* $\boldsymbol{P}_\omega$ モデルにおいては容易であり，その詳細については Scott [1976] を参照せよ．

うに与えられる.

$$a\otimes b=\lambda u.\,(a(\mathbf{first}(u)),\,b(\mathbf{second}(u)))$$
$$a\oplus b=\lambda u.\,(\mathbf{first}(u)\supset(\boldsymbol{t},\,a(\mathbf{second}(u))),\ (\boldsymbol{f},\,b(\mathbf{second}(u))))$$
$$a\circ\!\!\rightarrow b=\lambda u.\,b\circ u\circ a \qquad (5.16)$$

言語 **LAMBDA** に対するセマンティクスを式 (5.4) に与えたが, 以下では, プログラミング言語のセマンティクスをプログラム束の理論を基にして与えるいくつかの例を説明する.

5.2 プログラミング言語の意味論への応用

——言語の Denotational Semantics 入門——

プログラム束の理論は, 再帰的関数 (すなわち, 自己適用可能な関数) の意味とプログラミング言語の意味論に対する数学的な基礎を与えることを目的として始められたと考えてよい. 自己適用可能な関数の意味は, プログラム束の理論における不動点演算子 (**Y**) によって論じられることが明らかとなった. ここでは, プログラミング言語の意味論がプログラム束の理論を基にしてどのように論じられるかを例を用いて説明する*.

ここに説明する Denotational Semantics は Strachey [1966] によって始められた考え方に基づいており, Strachey の試みに対する数学的な基礎を与えたのが Scott によるプログラム束の理論である.

5.2.1 2進数のセマンティクス

2進数のシンタックスとセマンティクスを次のように与えることができる.

[シンタックス]

$N\in\boldsymbol{B}_{\mathrm{num}}$ ($\boldsymbol{B}_{\mathrm{num}}$: binary numeral の集合)

$N::=0\,|\,1\,|\,N0\,|\,N1$

[意味領域]

$\mathbf{N}=\{\mathrm{zero}\}+\mathbf{N}$

$=\{0,1,2,\cdots\}$

[意味関数]

$\mathcal{N}:\ \boldsymbol{B}_{\mathrm{num}}\to\mathbf{N}$

$\mathcal{N}[\![0]\!]=0$

$\mathcal{N}[\![1]\!]=1$

$\mathcal{N}[\![N0]\!]=2\times\mathcal{N}[\![N]\!]$

$\mathcal{N}[\![N1]\!]=2\times\mathcal{N}[\![N]\!]+1$

[コメント]

データタイプの和 $\boldsymbol{A}+\boldsymbol{B}$ は次のように定義される:

$$\boldsymbol{A}+\boldsymbol{B}=\{(\boldsymbol{t},a)\,|\,a\in\boldsymbol{A}\}\cup\{(\boldsymbol{f},b)\,|\,b\in\boldsymbol{B}\}$$

したがって, $\mathbf{N}=\{\mathrm{zero}\}+\mathbf{N}$ によって定義される要素は

0: $(\boldsymbol{t},\mathrm{zero})$

1: $(\boldsymbol{f},(\boldsymbol{t},\mathrm{zero}))$

2: $(\boldsymbol{f},(\boldsymbol{f},(\boldsymbol{t},\mathrm{zero})))$

3: $(\boldsymbol{f},(\boldsymbol{f},(\boldsymbol{f},(\boldsymbol{t},\mathrm{zero}))))$

⋮

ここで zero=⊥ と置くと, この自然数の定義は式 (5.11) において与えた定義と一致するから, **N** は自然数の領域と考えてよいことが知られる. (なお, 対 (x,y) の定義は式 (5.10) に与えられたものを用いるとする.) 意味領域 **N** においては加算 (+) と乗算 (×) が通常のように定義されるとしている.

5.2.2 λ-計算のセマンティクス

λ-計算の1つの言語モデル **LAMBDA** のシンタックスとセマンティクスを **5.1** に与えたが, ここでは, 若干異なる形で与えるとともに, LISP のリストを意味領域として持つ λ-計算のセマンティクスも与える.

(I) 単純 λ-計算 **SLAMBDA** のセマンティクス

[シンタックス]

$I\in\mathbf{Id}$ (**Id**: identifier の集合)

$E\in\mathbf{Exp}$ (**Exp**: expression の集合)

$E::=I\,|\,\lambda_I.\,E\,|\,EE\,|\,(E)$

[意味領域]

$\mathbf{E}=[\mathbf{E}\to\mathbf{E}]$ (式の値域)

$\boldsymbol{\Sigma}=[\mathbf{Id}\to\mathbf{E}]$ (environment)

[意味関数]

$\tilde{\mathbf{E}}:\ [\mathbf{Exp}\to[\boldsymbol{\Sigma}\to\mathbf{E}]]$

$\tilde{\mathbf{E}}[\![I]\!](\sigma)=\sigma(I)$ $(\sigma\in\boldsymbol{\Sigma})$

$\tilde{\mathbf{E}}[\![\lambda_I.\,E]\!](\sigma)=\lambda x.\,\tilde{\mathbf{E}}[\![E]\!](\sigma[x/I])$

$\tilde{\mathbf{E}}[\![E_1E_2]\!](\sigma)=(\tilde{\mathbf{E}}[\![E_1]\!](\sigma))(\tilde{\mathbf{E}}[\![E_2]\!](\sigma))$

$\tilde{\mathbf{E}}[\![(E)]\!](\sigma)=\tilde{\mathbf{E}}[\![E]\!](\sigma)$

ここに $\sigma(I)$ は environment σ における I の値. $\sigma[x/I]$ は σ において I を x に置き換えて得られる environment.

(II) アトム記号を持つ λ-計算 **A-LAMBDA** のセマンティクス

[シンタックス]

$I\in\mathbf{Id}$ (**Id**: identifier の集合)

$K\in\mathbf{Const}$ (**Const**: constant の集合)

$E\in\mathbf{Exp}$ (**Exp**: expression の集合)

$E::=I\,|\,K\,|\,\lambda_I.\,E\,|\,EE\,|\,(E)$

[意味領域]

$\mathbf{A}$ (アトムの集合)

* プログラミング言語の意味論の役割については, たとえば, 伊藤 [1980] を参照せよ.

$\mathbf{E}=\mathbf{A}+[\mathbf{E}\to\mathbf{E}]$　（式の値域）

$\Sigma=[\mathbf{Id}\to\mathbf{E}]$　(environment)

[意味関数]

$\mathcal{K}$: $[\mathbf{Const}\to\mathbf{E}]$

$\tilde{\mathbf{E}}$: $[\mathbf{Exp}\to[\Sigma\to\mathbf{E}]]$

$\tilde{\mathbf{E}}[\![I]\!](\sigma)=\sigma(I)$

$\tilde{\mathbf{E}}[\![K]\!](\sigma)=\mathcal{K}[\![K]\!]$

$\tilde{\mathbf{E}}[\![\lambda_I.\ E]\!](\sigma)=(\lambda x.\ \tilde{\mathbf{E}}[\![E]\!](\sigma[x/I]))$

$\tilde{\mathbf{E}}[\![E_1E_2]\!](\sigma)=(\tilde{\mathbf{E}}[\![E_1]\!](\sigma)|[\mathbf{E}\to\mathbf{E}])(\mathbf{E}[\![\tilde{E}_2]\!](\sigma))$

$\tilde{\mathbf{E}}[\![(E)]\!](\sigma)=\tilde{\mathbf{E}}[\![E]\!](\sigma)$

ここに $(\tilde{\mathbf{E}}[\![E_1]\!](\sigma)|[\mathbf{E}\to\mathbf{E}])$は$\mathbf{E}$の部分領域 $[\mathbf{E}\to\mathbf{E}]$ における $(\tilde{\mathbf{E}}[\![E_1]\!](\sigma))$ の値を意味している．

[コメント]

単純 λ-計算 **SLAMBDA** に対するセマンティクスが λ-計算のモデルとなることは，**5.1** の言語 **LAMBDA** のセマンティクスと LAMBDA 論理から明らかであろう．なお，ある λ-計算の言語モデルがその意味定義（セマンティクス）の下で λ-計算の基本的な性質（α-規則，β-規則，η-規則，Church-Rosser の定理など）を満足するときに λ-計算モデルになっていると言われる．

アトム記号を持つ λ-計算 **A-LAMBDA** のセマンティクスが λ-計算モデルとなることは別途示す必要がある．***P*∞*** モデルにおける詳細な議論は Stoy [1977] に与えられている．***D*∞*** モデルにおいては，自然数を ***B*∞*** モデルにおいて表現したのと同様の手法によってアトム記号に対応する対象を ***B*∞*** モデルで表現してやれば ***B*∞*** モデルの中に **A-LAMBDA** の意味領域を埋め込めることになり，ここで与えたセマンティクスが λ-計算モデルとなることが直感的に知られる．

（Ⅲ）　フローチャート言語 **FLOW** のセマンティクス*

[シンタックス]

$I\in\mathbf{Id}$　（**Id**: identifier の集合）

$E\in\mathbf{Exp}$　（**Exp**: expression の集合）

$C\in\mathbf{Cmd}$　（**Cmd**: Command の集合）

E::=0|1|true|false|I|$-E$|$E+E$|**not** E|$E=E$|**procedure** C|(E)

C::=**dummy**|$I\leftarrow E$|**call** E|C; C|**if** E **then** C **else** C|**while** E **do** C|**begin** C **end**

[意味領域]

Num　（数値領域）

Bool　（真理値領域）

Val=**Num**+**Bool**　（値域）

Mv=**Val**+[**S**→**Sr**]　（メモリの値）

S=**Id**→(**Mv**+{unbound})　（メモリの状態）

Er=**Mv**+{error}　（式の結果）

Sr=**S**+{error}　（文の結果）

[意味関数]

ρ: $\mathbf{Exp}\to[\mathbf{S}\to\mathbf{Er}]$

π: $\mathbf{Cmd}\to[\mathbf{S}\to\mathbf{Sr}]$

$\rho[\![0]\!](s)=0$

$\rho[\![1]\!](s)=1$

$\rho[\![\text{true}]\!](s)=\boldsymbol{t}$

$\rho[\![\text{false}]\!](s)=\boldsymbol{f}$

$\rho[\![I]\!](s)=(\text{isvar}[I]\supset s[\![I]\!],\text{error})$

ここに $s[\![I]\!]$ は状態 s における変数 I の値とする．

$\rho[\![-E]\!](s)=(\text{isexp}[E]\supset-\rho[\![E]\!](s),\text{error})$

$\rho[\![E_1+E_2]\!](s)$
$=(\text{isexp}[E_1]\wedge\text{isexp}[E_2]\supset\rho[\![E_1]\!](s)+\rho[\![E_2]\!](s),\text{error})$

$\rho[\![\mathbf{not}\ E]\!](s)=(\text{isbool}[E]\supset\neg\rho[\![E]\!](s),\text{error})$

$\rho[\![E_1=E_2]\!](s)$
$=(\text{isexp}[E_1]\wedge\text{isexp}[E_2]\supset\rho[\![E_1]\!](s)=\rho[\![E_2]\!](s),\text{error})$

$\rho[\![\mathbf{procedure}\ C]\!](s)=\pi[\![C]\!]$

$\rho[\![(E)]\!](s)=\rho[\![E]\!](s)$

$\pi[\![\mathbf{dummy}]\!](s)=s$

$\pi[\![I\leftarrow E]\!](s)=(\text{isexp}[E]\supset s(\rho[\![E]\!](s)/I),\text{error})$

$\pi[\![\mathbf{call}\ E]\!](s)=(\text{isprocedure}[E]\supset\rho[\![E]\!](s),\text{error})$

$\pi[\![C_1;\ C_2]\!](s)$
$=(\pi[\![C_1]\!](s)\in\mathbf{S}\supset\pi[\![C_2]\!](\pi[\![C_1]\!](s)),\text{error})$

$\pi[\![\mathbf{if}\ E\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2]\!](s)$
$=(\rho[\![E]\!](s)\supset\pi[\![C_1]\!](s),\pi[\![C_2]\!](s))$

$\pi[\![\mathbf{while}\ E\ \mathbf{do}\ C]\!](s)$
$=\mathbf{Y}(\lambda f.\ (\rho[\![E]\!](s)\supset(\pi[\![C]\!](s)\in\mathbf{S}\supset f(\pi[\![C]\!](s)),\text{error}),s)$

$\pi[\![\mathbf{begin}\ C\ \mathbf{end}]\!](s)=\pi[\![C]\!](s)$

ここに isvar[I], isexp[E], isbool[E] などは，それぞれ，I, E, E が変数 (variable)，算術式 (arithmetic expression)，論理式 (boolean expression) であれば真 (true)，そうでなければ偽 (false) であるとする．なお，意味領域における定数 (0, 1, $\boldsymbol{t}$, $\boldsymbol{f}$) や演算（−，+，¬(論理否定)，=）などは通常のごとく定義されているとする．なお意味定義の仕方は，ここに与えたものに一意的に定まる訳ではなく，たとえば，

---

* ここで与えるセマンティクスを状態ベクトル・セマンティクスと呼ぶことがある．

$\pi[\![C_1\,;\,C_2]\!](s)$

$=((\pi[\![C_1]\!](s)=\text{error})\supset\text{error},\ \pi[\![C_2]\!](\pi[\![C_1]\!](s))$

のように与えることもできる.

[シーケンス演算⊛と直接的意味論について]

プログラムは文の系列からなる. 文の系列を形成する基本的な演算は『$C_1\,;\,C_2$』である. 『$C_1\,;\,C_2$』の意味を状態ベクトルsに関する関数的な演算(あるいは, 関数の合成)として表現することをこれまでに考えてきた. しかし, プログラムを構成する文の系列の実行という観点からは, 与えられた状態の下で,

① C_1を実行し, 新しい状態が得られたら, その状態の下でC_2を実行し,

② C_1を実行し, "error"が生じたら, "C_1 : C_2"の結果も"error"とする

というのが自然と考えられる. このようなプログラムの実行系列を反映した演算をシーケンス演算と呼ぶことにする. このような立場から提案されている演算として次のようなシーケンス演算⊛がある:

f : $\boldsymbol{D}_1\to[\boldsymbol{D}_2+\{\text{error}\}]$

g : $\boldsymbol{D}_2\to[\boldsymbol{D}_3+\{\text{error}\}]$

としたとき

$$f\circledast g=\lambda x.\,((f(x)=\text{error})\supset\text{error},\ g(f(x))) \tag{5.17}$$

このシーケンス演算⊛を用いると次のように書ける.

$$\pi[\![C_1\,;\,C_2]\!]=\pi[\![C_1]\!]\circledast\pi[\![C_2]\!] \tag{5.18}$$

また, 式の評価に関しても次のように書ける.

$$\rho[\![E_1=E_2]\!]=\rho[\![E_1]\!]\circledast\lambda u.\,\rho[\![E_2]\!]\circledast\lambda v.\,(u=v) \tag{5.19}$$

シーケンス演算⊛を用いて状態の変換を直接的に表現し意味記述を行う方法が直接的意味論(Direct Semantics)と呼ばれる. フローチャート言語 **FLOW** 全体に対する直接的意味論を与えることも式 (5.18) および式 (5.19) と同様の仕方で行えるがその詳細は読者への演習問題としておく.

ここで考えているシーケンス演算⊛では **goto** 文を扱うのには限界があることを述べておこう. たとえば, "**go to** $L\,;\,C$" の意味を⊛を用いて

$$\pi[\![\textbf{go to } L\,;\,C]\!]=\pi[\![\textbf{go to } L]\!]\circledast\pi[\![C]\!] \tag{5.20}$$

としたとき, "**go to** L" は文 "C" に渡すべき状態を生成しないから意味をなさないと考えられる. "**go to** L" に対しては, ラベルLの付した文 "L : C_0" を探し, 実行をこの文に移す必要がある. このような状態の連結関係を continuation という領域を導入して正しく追跡し意味記述を与える仕方が連続性を用いた意味論 (Continuation Semantics) である. 次に簡単な例によって Continuation Semantics について説明する.

(Ⅳ) Continuation Semantics の例

プログラミング言語に現われる定義や宣言および飛び越し文 (Jump および **goto**) などの意味記述を与えるためには,

- **environment**
- **continuation**

を用いる必要がある. environment は **LAMBDA** の意味記述にも用いたように, identifier をそれが意味する値に結合する. したがって, environment $\mathbf{Ev}=\mathbf{Id}\to\mathbf{Dv}$ と与えられる. ここに $\mathbf{Dv}=\mathbf{L}+\mathbf{Mv}+\{\text{undefined}\}$ が denotable value の集合となる.

continuation は与えられたプログラムの実行結果の関数として表現される. 文あるいは命令の continuation は次のように与えられる.

$$\mathbf{C}_c=\mathbf{S}\to\mathbf{A} \tag{5.21}$$

ここに$\mathbf{S}$はメモリ状態の集合, $\mathbf{A}$は命令実行結果としての答 (Answer) の集合である. このとき, 命令の意味は次のように与えられる:

$$\hat{\pi}:\ \mathbf{Cmd}\to[\mathbf{Ev}\to[\mathbf{C}_c\to[\mathbf{S}\to\mathbf{A}]]] \tag{5.22}$$

たとえば*,

$$\hat{\pi}[\![C_1\,;\,C_2]\!](\beta)(\theta)(s)=\hat{\pi}[\![C_1]\!](\beta)(\hat{\theta})(s) \tag{5.23}$$

ここに, $\forall u.\,[\hat{\theta}(u)=\hat{\pi}[\![C_2]\!](\beta)(\theta)(u)]$.

なお, $(\beta)(\theta)(s)\triangleq\beta\theta s$ と記し, command continuation $\hat{\theta}$ を次のように記す.

$$\hat{\theta}=\lambda u.\,\hat{\pi}[\![C_2]\!]\beta\theta u \tag{5.24}$$

このとき

$$\hat{\pi}[\![C_1\,;\,C_2]\!]\beta\theta s=\hat{\pi}[\![C_1]\!]\beta(\lambda u.\,\hat{\pi}[\![C_2]\!]\beta\theta u)s \tag{5.25}$$

たとえば

$$\hat{\pi}[\![\textbf{dummy}]\!]\beta\theta s=\theta(s) \tag{5.26}$$

次にフローチャート言語 **FLOW** に **goto** 文を追加した言語 **GFLOW** の continuation semantics を与えることを考えよう. **GFLOW** のシンタックスは次のように与えられる:

E ::= $0\,|\,1\,|\,\textbf{true}\,|\,\textbf{false}\,|\,I\,|\,-E\,|\,E+E\,|\,\textbf{not } E\,|\,E=E\,|\,\textbf{procedure } C\,|\,(E)$

C ::= $\textbf{dummy}\,|\,I\leftarrow E\,|\,\textbf{call } E\,|\,C\,;\,C\,|\,\textbf{if } E \textbf{ then}$

* この例から知られるように, β は identifier に対する binding を行い, θ が実行のシーケンスを決定し, S の下で実行されるというように考えるのが continuation semantics の考え方である.

C **else** C|**while** E **do** C|**begin** C **end**
|I : C|G
G=**go to** I

[意味領域]

Num	(numbers)
Bool	(boolean values)
L	(locations)
Val=**Num**+**Bool**	
Mv=**Val**+[**S**→**Sr**]	(memory values)
A=**Val**+{error}	(answers)
S=**L**→(**Mv**+{unbound})	(memory states)
Ev=**Id**→**Dv** ($\ni\beta$)	(environments)
Dv=**L**+**Mv**+{undefined}	(denotable values)
Er=**Mv**+{error}	(expression results)
Sr=**S**+{error}	(command results)
P=$\mathbf{C_c}$→[**S**→**A**]	(procedures)
$\mathbf{C_c}$=**S**→**A** ($\ni\theta$)	(command continuation)
$\mathbf{E_c}$=**Mv**→**A** ($\ni\varphi$)	(expression continuation)

[意味関数]

ρ : **Exp**→[**Ev**→[$\mathbf{E_c}$→[**S**→**A**]]]
π : **Cmd**→[**Ev**→[$\mathbf{C_c}$→[**S**→**A**]]]
$\hat{\tau}$: **G**→[**Ev**→[**S**→**A**]]
$\rho[\![0]\!]\beta\varphi s=\varphi(0)$
$\rho[\![1]\!]\beta\varphi s=\varphi(1)$
$\rho[\![\text{true}]\!]\beta\varphi s=\varphi(\boldsymbol{t})$
$\rho[\![\text{false}]\!]\beta\varphi s=\varphi(\boldsymbol{f})$
$\rho[\![I]\!]\beta\varphi s=(\beta[\![I]\!]\in\mathbf{L}\supset\varphi(s(\beta[\![I]\!])),$
$\quad(\beta[\![I]\!]\in\mathbf{Mv}\supset s(\beta[\![I]\!]),\text{error})$
$\rho[\![-E]\!]\beta\varphi s$
$=\rho[\![E]\!]\beta(\lambda x.(x\in\mathbf{Num}\supset\varphi(-x),\text{error}))s$
$\rho[\![E_1+E_2]\!]\beta\varphi s$
$=\rho[\![E_1]\!]\beta(\lambda x.(x\in\mathbf{Num}$
$\supset\rho[\![E_2]\!]\beta(\lambda y.(y\in\mathbf{Num}\supset\varphi(x+y),$
$\text{error}))s,\text{error}))s$
$\rho[\![\textbf{not } E]\!]\beta\varphi s$
$=\rho[\![E]\!]\beta(\lambda x.(x\in\mathbf{Bool}\supset\varphi(\neg x),\text{error}))$
$\rho[\![E_1=E_2]\!]\beta\varphi s$
$=\rho[\![E_1]\!]\beta(\lambda x.(x\in\mathbf{Bool}\supset\rho[\![E_2]\!]\rho(\lambda y.(y\in\mathbf{Bool}$
$\supset\varphi(x=y),\text{error}))s,\text{error}))s$
$\rho[\![\textbf{procedure } C]\!]\beta\varphi s=\varphi(\pi[\![C]\!]\beta)$
$\rho[\![(E)]\!]\beta\varphi s=\rho[\![E]\!]\beta\varphi s$
$\pi[\![\textbf{dummy}]\!]\beta\theta s=\varphi(s)$
$\pi[\![I\leftarrow E]\!]\beta\theta s$
$=\rho[\![E]\!]\beta(\lambda x.(x\in\mathbf{L}\supset\theta([\beta[\![I]\!]/x]),\text{error}))s$
$\pi[\![\textbf{call } E]\!]\beta\theta s$
$=\rho[\![E]\!]\beta(\lambda x.(x\in\mathbf{P}\supset x(\theta)(s),\text{error}))s$
$\pi[\![C_1\,;\,C_2]\!]\beta\theta s=\pi[\![C_1]\!]\beta(\lambda u.\pi[\![C_2]\!]\beta\theta u)s$
$\pi[\![\textbf{if } E \textbf{ then } C_1 \textbf{ else } C_2]\!]\beta\theta s$
$=\rho[\![E]\!]\beta(\lambda x.(x\supset\pi[\![C_1]\!]\beta\theta s,\pi[\![C_2]\!]\beta\theta s))s$
$\pi[\![\textbf{while } E \textbf{ do } C]\!]\beta\theta s=\hat{\theta}(s)$

ここに

$\hat{\theta}=\mathbf{Y}(\lambda t.\lambda x.\rho[\![E]\!]\beta(\lambda y.(y\supset\pi[\![C]\!]\beta t x,\theta(x))))$
$\pi[\![\textbf{begin } C \textbf{ end}]\!]\beta\theta s=\pi[\![C]\!]\beta\theta s$
$\pi[\![I\,:\,C]\!]\beta\theta s=\mathbf{Y}(\lambda\hat{\theta}.\pi[\![C]\!]([\hat{\theta}/I])\theta)s$
$\pi[\![G]\!]\beta\theta s=\hat{\tau}[\![G]\!]\beta s$
$\hat{\tau}[\![\textbf{go to } I]\!]\beta s=(\beta[\![I]\!]\in\mathbf{C_c}\supset\beta[\![I]\!](s),\text{error})$

[コメント]

言語 **GFLOW** から **goto** 文とラベル文 (I : C) を除去すると continuation を用いることなくセマンティクスが与えられる．このとき意味関数は

ρ' : **Exp**→[**Ev**→[**S**→**Er**]]
π' : **Cmd**→[**Ev**→[**S**→**Sr**]]　　(5.24)

を考えれば十分である．このとき environment を用いた意味記述を行っているから，次のような意味関数 δ を用いるとデータ定義や宣言が扱えることになる．

δ : **Dfl**→[**Ev**→[**S**→[**Ev**×**Sr**]]]　　(5.25)

ここに **Dfl** はデータ定義および宣言のシンタックス定義式の集合とする．

なお，Jump や **goto** 文がなければ，continuation semantics を用いなくても良いことが知られており，

$\hat{\rho}[\![E]\!]\beta\varphi=\rho'[\![E]\!]\beta\circledast\varphi$
$\hat{\pi}[\![C]\!]\beta\theta=\pi'[\![C]\!]\beta\circledast\theta$　　(5.26)

が成り立つ．continuation を用いない semantics (direct semantics) と continuation semantics の関係については，たとえば，Reynolds [1974] および Jones [1978] を参照せよ．

5.3 プログラムの性質の証明

LAMBDA 論理は関数の近似関係と同値性に関する公理系であるから，これを用いてプログラムの性質を証明することができる．またプログラミング言語のセマンティクスを用いるとプログラムの正当性の証明やコンパイラ作成の自動化も行いやすくなることが予想される．ここでは，プログラムのいくつかの性質を表現するとともに，それを用いて Hoare のプログラム論理の規則の検証が行えることを説明する．

5.3.1 プログラムの性質の表現と証明

LAMBDA 論理は関数の近似関係と同値性に関する公理系であるから，これを用いてプログラムの同値性や停止性の証明を行うことができる．

LAMBDA 論理における証明の例を簡単な例によって示そう．

① $f=g,\ g=h\vdash f=h$

(∵) $f=g,\ g=h\vdash f\sqsubseteq g,\ g\sqsubseteq h$

$f\sqsubseteq g,\ g\sqsubseteq h\vdash f\sqsubseteq h$

$f=g,\ g=h\vdash f\sqsubseteq h$

$f=g,\ g=h\vdash h\sqsubseteq g,\ g\sqsubseteq f$

$h\sqsubseteq g,\ g\sqsubseteq f\vdash h\sqsubseteq f$

$f=g,\ g=h\vdash h\sqsubseteq f$

$f=g,\ g=h\vdash f=h$

[注] なお，この証明では

$x=y\Leftrightarrow x\sqsubseteq y$ and $y\sqsubseteq x$

と定義されているとしている．

② $f\sqsubseteq g,\ x\sqsubseteq y\vdash f(x)\sqsubseteq g(y)$

(∵) $x\sqsubseteq y$

$f(x)\sqsubseteq f(y)$

$f(y)\sqsubseteq g(y)$

$f(x)\sqsubseteq g(y)$

③ $f=\mathbf{Y}(F),\ g=F(g)\vdash f\sqsubseteq g$

(∵) $\perp\sqsubseteq g$

$h\sqsubseteq g\vdash F(h)\sqsubseteq F(g)$

$F(g)=g,\ F(h)\sqsubseteq F(g)\vdash F(h)\sqsubseteq g$

$\mathbf{Y}(F)\sqsubseteq g$

$f\sqsubseteq g$

これらの例によって知られるように LAMBDA 論理を用いることによって，プログラムの同値関係，近似関係が証明できる（可能性がある）*．プログラム x が停止しないという性質は

$$x=\perp \tag{5.27}$$

と表現できるから，やはりLAMBDA論理で扱える．

プログラムの停止性を扱うのには，次のような停止述語 (termination predicate) $\tau[x]$ を用いるのが便利である**．$\boldsymbol{D}$ をデータタイプとし，$\boldsymbol{D}=\hat{\boldsymbol{D}}\cup\{\perp\}$ とする．ここに $\perp\notin\hat{\boldsymbol{D}}$ とする．このとき $\tau[x]$ は次のように定義される：

$$\tau[\perp]=\perp$$

$$\tau[d]=t\qquad (d\in\hat{\boldsymbol{D}}) \tag{5.28}$$

いま $\varphi(x)$ を $\varphi:\boldsymbol{D}\to\{\perp,t\}$ としたとき，プログラム（あるいは関数）f の φ に関する停止性を次のように表わすことができる．

$$\lambda x.\,\varphi(x)\sqsubseteq\lambda x.\,\tau[f(x)] \tag{5.29}$$

プログラムの停止性証明の具体例は後述することにし，次にプログラムの正当性がどのように記述されるかを述べることにしよう．プログラムの正当性の概念の定義にはいくつかの仕方があるが，代表的なものとして次の2つがある*．

① 弱正当性：入力条件 Q が真でプログラム C の実行が停止して出力条件 R が真となるか，Q が真で C が停止しないか，あるいは Q が偽となるかのいずれかの場合に，プログラム C は入出力条件 (Q,R) に関して正しいと主張するものである．Hoare の検証文 $\{Q\}C\{R\}$ による正当性の定義はこの場合に当る．

② 強正当性：入力条件 Q が真でプログラム C の実行が停止して出力条件 R が真となるときにのみプログラム C は入出力条件 (Q,R) に関して正しいと主張するものである．

これらの概念はプログラム束の理論の形式化を用いると次のように定義できる**．

① 弱正当性（プログラム C の入出力条件 (Q,R) に関する弱正当性）

$$\lambda s.\,\rho[\![R]\!](\pi[\![C]\!](s))\sqsubseteq\lambda s.\,(\rho[\![Q]\!](s)\supset t,\ \rho[\![R]\!](\pi[\![C]\!](s))) \tag{5.30}$$

② 強正当性（プログラム C の入出力条件 (Q,R) に関する強正当性）

$$\lambda s.\,t\sqsubseteq\lambda s.\,(\rho[\![Q]\!](s)\supset\rho[\![R]\!](\pi[\![C]\!](s)),f) \tag{5.31}$$

ここに $\rho:\mathbf{Exp}\to[\mathbf{S}\to\mathbf{Val}]$

$\pi:\mathbf{Cmd}\to[\mathbf{S}\to\mathbf{S}]$ (5.31 a)

とする；また，プログラムに対する入出力条件 (Q,R) は total predicate であるとしている．

プログラムの性質をプログラム束の理論を用いて，上述のように形式化できる．すなわち，

(1) プログラムの近似関係 $f_1\sqsubseteq f_2$

(2) プログラムの同値性 $f_1=f_2$

(3) プログラムの非停止性 $f=\perp$

(4) プログラムの停止性

$\lambda x.\,\varphi(x)\sqsubseteq\lambda x.\,\tau[f(x)]$

(5) プログラムの弱正当性

$\lambda s.\,\rho[\![R]\!](\pi[\![C]\!](s))\sqsubseteq\lambda s.\,(\rho[\![Q]\!](s)$

---

* Scott の仮説が成り立つことを認めると LAMBDA 論理は，本質的に "undecidable theory"（あるいは "incomplete theory"）となるから自動的な証明手法は存在しない．

** $\tau[x]$ は "well-definedness predicate" とも呼ばれる．

* プログラムの正当性の記述に関しては 3.2 を参照せよ．

** 話を簡単にするために状態ベクトル・セマンティクスを用いて対応関係を与えている．

$$\supset t, \rho[\![R]\!](\pi[\![C]\!](s)))$$

（6） プログラムの強正当性

$$\lambda s.\, t \sqsubseteq \lambda s.\, (\rho[\![Q]\!](s) \supset \rho[\![R]\!](\pi[\![C]\!](s)), f)$$

プログラムの正当性については **5.3.2** において Hoare のプログラム論理の妥当性を論じる際に説明するから，ここではプログラムの停止性がどのように証明されるかを例によって説明する．

[プログラムの停止性の証明と termination predicate]

具体的なプログラムの停止性を証明するためには，プログラムに現われる対象についての公理を与えておくのが便利である．まず，自然数の領域で定義される関数を例にとって証明の仕方を説明するために，次のような公理 $\mathcal{T}_A$ を用いる．

$\mathcal{T}_A$：（1） $\lambda x.\,((x+1)-1)=\lambda x.\, x$

（2） $Z(\perp)=\perp$

（3） $0-1=\perp$

（4） $\mathbf{Y}(\lambda f.\,\lambda x.\,(Z(x) \supset 0, f(x-1)+1))=\lambda x.\, x$

ここに "+1" および "−1" は，それぞれ，後者関数および前者関数を表わし，0は定数，$Z(x)$ は x が零 (zero) であることを示す述語であるとする*．

このとき termination predicate τ は次のように定義される．

$$\tau=\mathbf{Y}(\lambda f.\,\lambda x.\,(Z(x) \supset t, f(x-1))) \qquad (5.32)$$

このとき，次のように定義される加算⊞の停止性を考える．

$$\boxplus=\mathbf{Y}(\lambda f.\,\lambda xy.\,(Z(y) \supset x, f(x, y-1)+1)) \qquad (5.33)$$

証明のゴールは

$$\mathcal{T}_A \vdash \tau[x] \,\&\, \tau[y] \sqsubseteq \tau[x \boxplus y] \qquad (5.34)$$

ここに

$$\&=\lambda xy.\,(x \supset y, f) \qquad (5.35)$$

この & 演算に関しては次のような性質が成り立つ．

$$x \sqsubseteq t, y \sqsubseteq t \vdash x \,\&\, y \sqsubseteq y, x \,\&\, y \sqsubseteq x$$
$$x \sqsubseteq y, x \sqsubseteq z \vdash x \sqsubseteq y \,\&\, z \qquad (5.36)$$

（ここに $x, y, z \in \{\perp, t, f\}$ とする）

LAMBDA 論理の公理と推論規則および公理 $\mathcal{T}_A$ の性質を用いて式(5.34)が成り立つことを示せばよい．（証明の詳細は読者への演習問題としておく．）同様の仕方で，適当に定義された減算，乗算，除算，不等号などの停止性の証明が行える．また，これらの演算をもとにして定義される原始帰納的関数の停止性の証明も行うことができる．

自然数に関する関数の停止性は，以上のように公理 $\mathcal{T}_A$ と termination predicate τ を用いて証明できるが，LISP 関数の場合には LISP 基本関数に関する公理とその領域での termination predicate を考える必要がある．たとえば

$\mathcal{L}_A$：（1） $\lambda xy.\,\mathbf{car}[\mathbf{cons}[x, y]]=\lambda xy.\, x$

（2） $\lambda xy.\,\mathbf{cdr}[\mathbf{cons}[x, y]]=\lambda xy.\, y$

（3） $(\mathbf{atom}[x] \supset \mathbf{car}[x], \perp)=\perp$

（4） $(\mathbf{atom}[x] \supset \mathbf{cdr}[x], \perp)=\perp$

（5） $\mathbf{atom}[\mathrm{NIL}]=t$

（6） $\mathbf{Y}(\lambda f.\,\lambda x.\,(\mathbf{null}[x] \supset \mathrm{NIL}, (\mathbf{atom}[x] \supset x, \mathbf{cons}[\mathbf{car}[x], \mathrm{cdr}[x]])))=\lambda x.\, x$

この場合には termination predicate τ は

$$\tau=\mathbf{Y}(\lambda f.\,\lambda x.\,(\mathbf{atom}[x] \supset t, \tau[\mathbf{car}[x]] \,\&\, \tau[\mathbf{cdr}[x]])) \qquad (5.37)$$

によって与えられる．

これまでの議論では termination predicate τ は D の要素のみに対して定義されてきたが，たとえば，$[D \to D]$ に対しても拡張することができる．

$$\tau[\perp]=\perp$$
$$\tau[x]=t \quad (x \neq \perp) \qquad (5.38)$$

このときには，次式が成り立つ：

$$\tau[f(x, y)]=\tau[f] \,\&\, \tau[x] \,\&\, \tau[y] \qquad (5.39)$$

また strict equality ($\cong$) を次のように定義できる．

$$x \cong y \Leftrightarrow (x=y) \,\&\, \tau[x] \,\&\, \tau[y] \qquad (5.40)$$

この strict equality を用いると τ は次のようにも書ける．

$$\tau=\lambda x.\,(x \cong x) \qquad (5.41)$$

5.3.2 プログラム論理の妥当性の検証への応用

プログラムの正当性を検証するのに使われる体系として，Hoare によるプログラム論理がある．プログラム論理においては，Hoare の検証文

$$\{Q\}C\{R\} \qquad (5.42)$$

を用いて書かれる公理系から誘導される文によって正しいプログラムの族を特性付けている．プログラム論理の公理系や推論規則は計算モデルの下で妥当なものであること——"soundness of program logic"——が言えなければ，理論上は問題がある．

Hoare の検証文はプログラム C の入出力条件 (Q, R) に関する弱正当性を主張するものであり，式(5.42)の Hoare の検証文はプログラム束の理論の形式化を用いると式 (5.30) のように表現できる．すなわち，次のような対応がある．(ρ, π については式 (5.31 a) を

* $Z(\perp)=\perp$, $Z(0)=t$, $Z(x)=f$ (for $x=1, 2, \cdots$).

参照.)

$$\{Q\}C\{R\}$$
$$\Updownarrow$$
$$\lambda s.\ \rho[\![R]\!](\pi[\![C]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![Q]\!](s) \supset t,\ \rho[\![R]\!](\pi[\![C]\!](s))) \quad (5.43)$$

この対応関係を用いることによって，プログラム論理の公理がプログラム束の理論の中ですべて妥当であることが示せ，プログラム論理の推論規則はすべて妥当性を保存することが示せる．プログラム論理が"sound"であることがプログラム束の理論を用いて主張できることになる．Hoare のプログラム論理には多数の公理と推論規則が含まれているが，その中から次のような最も基本的なものだけを説明しておく：

H1: $\{[e/x]Q\}\ x \leftarrow e\ \{Q\}$

H2: $\dfrac{\{Q\}A\{P\},\ \{P\}B\{R\}}{\{Q\}A;\ B\{R\}}$

H3: $\dfrac{\{Q \wedge p\}A\{R\},\ \{Q \wedge \neg p\}B\{R\}}{\{Q\}\textbf{if } p \textbf{ then } A \textbf{ else } B\{R\}}$

H4: $\dfrac{\{Q \wedge p\}C\{Q\}}{\{Q\}\textbf{while } p \textbf{ do } C\{Q \wedge \neg p\}}$

① H1 が妥当であることを示すには，次式を言えばよい：

$$\lambda s.\ \rho[\![Q]\!](\pi[\![x \leftarrow e]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![[e/x]]\!]Q(s) \supset t,\ \rho[\![Q]\!](\pi[\![x \leftarrow e]\!](s)))$$

② H2 が妥当性を保存する規則であることを示すには

$$\lambda s.\ \rho[\![P]\!](\pi[\![A]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![Q]\!](s) \supset t,\ \rho[\![P]\!](\pi[\![A]\!](s)))$$
$$\lambda s.\ \rho[\![R]\!](\pi[\![B]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![P]\!](s) \supset t,\ \rho[\![R]\!](\pi[\![B]\!](s)))$$

から

$$\lambda s.\ \rho[\![R]\!](\pi[\![A;\ B]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![Q]\!](s) \supset t,\ \rho[\![R]\!](\pi[\![A;\ B]\!](s)))$$

を証明すればよい．

③ H3 が妥当性を保存する規則であることを示すには

$$\lambda s.\ \rho[\![R]\!](\pi[\![A]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![Q \wedge p]\!](s) \supset t,\ \rho[\![R]\!](\pi[\![A]\!](s)))$$
$$\lambda s.\ \rho[\![R]\!](\pi[\![B]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![Q \wedge \neg p]\!](s) \supset t,\ \rho[\![R]\!](\pi[\![B]\!](s)))$$

から

$$\lambda s.\ \rho[\![R]\!](\pi[\![\textbf{if } p \textbf{ then } A \textbf{ else } B]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![Q]\!](s) \supset t,\ \rho[\![R]\!](\pi[\![\textbf{if } p \textbf{ then } A \textbf{ else } B]\!](s)))$$

を証明すればよい．

④ H4 が妥当性を保存する規則であることを示すには

$$\lambda s.\ \rho[\![Q]\!](\pi[\![C]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![Q \wedge p]\!](s) \supset t,\ \rho[\![Q]\!](\pi[\![C]\!](s)))$$

から

$$\lambda s.\ \rho[\![Q \wedge \neg p]\!](\pi[\![\textbf{while } p \textbf{ do } C]\!](s)) \sqsubseteq \lambda s.\ (\rho[\![Q]\!](s) \supset t,\ \rho[\![Q \wedge \neg p]\!](\pi[\![\textbf{while } p \textbf{ do } C]\!](s)))$$

を証明すればよい．

証明は LAMBDA 論理と意味記述に関する次のような性質などを用いて行える．

$$\rho[\![x]\!](\pi[\![x \leftarrow e]\!](s)) = \rho[\![e]\!](s)$$
$$\rho[\![y]\!](\pi[\![x \leftarrow e]\!](s)) = \rho[\![y]\!](s) \quad (\text{ここに } x \neq y)$$
$$\pi[\![x \leftarrow e]\!](s) = (\rho[\![e]\!](s) = \perp \supset \perp,\ s([\rho[\![e]\!]/x]))$$
$$\pi[\![C_1;\ C_2]\!](s) = \pi[\![C_2]\!](\pi[\![C_1]\!](s))$$
$$\pi[\![\textbf{if } p \textbf{ then } C_1 \textbf{ else } C_2]\!](s) = (\rho[\![p]\!](s) \supset \pi[\![C_1]\!](s),\ \pi[\![C_2]\!](s))$$
$$\pi[\![\textbf{while } p \textbf{ do } C]\!](s) = (\rho[\![p]\!](s) \supset \pi[\![\textbf{while } p \textbf{ do } C]\!](\pi[\![C]\!](s)),\ s)$$

証明の詳細は，紙数の関係上，読者への演習問題としておく．

[コメント]

この章ではプログラム束の理論の意味論的観点からの応用例について述べた．領域方程式とその解，コンパイラの正当性，ソフトウェアの仕様記述への応用などについても述べたかったが紙数の都合で省略した．たとえば，領域方程式に関しては Plotkin-Smyth [1978]，コンパイラの正当性に関しては Milner-Wayrauch [1972]，ソフトウェアの仕様記述に関しては ADJ [1977] および Burstall-Goguen [1977] を参照されたい．

プログラム束の理論の応用という観点から重要なものとして，Scott による階型モデル (unpublished note において提案されたモデル) を基にして Milner [1972] が proof checker を構成した LCF がある．LCF に関しては Gordon 他 [1979] を参照されたい．

Continuation Semantics に関する説明が不十分になったが，具体例によって理解を深められることを期待したい．[たとえば，n の階乗を求める **while** 文を用いたプログラムに対しては Direct Semantics と Continuation Semantics が等しくなることを示せ．また，n の階乗を求める **goto** 文を用いたプログラムと **while** 文を用いたプログラムの Continuation

Semantics が等しくなることを示せ．**goto** 文があるプログラムの Direct semantics と Continuation semantics とが一致するプログラムの族（あるいは条件）を与えよ．] Hoare のプログラム論理には **goto** 文に対する規則もあり，その規則が妥当性を保存することを示すには Continuation Semantics を用いる必要がある．（これも演習問題としておく．）

本章の **5.1** は Scott [1973], Milner [1972] および Weyrauch（スタンフォード大学）が筆者らに行った Scott 理論の話を参考にしている．**5.2** は Scott-Strachey [1971], Tennent [1976], Donahue [1976], Stoy [1977], Gordon [1979] を参考にしている．**5.3** のプログラムの停止性および正当性の表現は，1970 年～1974 年頃にかけての筆者の未発表の結果に基づく．（その中には，文献 16) の 668 頁の脚注に記した形式化に基づく，次のような正当性，誤謬性の議論も含まれる．弱正当性は，『$C \sqsubseteq Q \circ C \circ R \sqcup \bar{Q} \circ C$』と表わせる．停止述語 $\hat{\tau}$ を『$\hat{\tau}[\bot]=\bot$, $\hat{\tau}[d]=e\ (d \neq \bot)$』と定義すると*，強正当性は『$e \sqsubseteq \hat{\tau}[Q \circ C \circ R]$』と表わせる．停止しないプログラムは誤りであると考えると，『$C \sqsubseteq \bot$』；入出力条件が満足されないか，停止しなければ誤りであると考えると，『$Q \circ C \circ R \sqsubseteq \bot$』などのようにプログラムの誤謬性も形式化できる．プログラムの正当性・誤謬性の概念は，これら以外にも考えられるが，その形式化は読者への演習問題としておく．）この章の多くの部分は昭和 53 年度における九州大学および東北大学での講義資料を基にまとめたものである．

参考文献

1) ADJ (Goguen, J. Thatcher, J. et al.): Initial algebra semantics and continuous algebra, JACM, 24, 1, pp. 68-95 (1977).
2) Donahue, J.: Complementary Definitions of Programming Languages Semantics, SLN-CS, 42 (1976).
3) Gordon, M.: The Denotational Description of Programming Languages, Springer-Verlag (1979).
4) Gordon, M., Milner, R. et al.: Edinburgh LCF, SLN-CS, 78 (1979).
5) Jones, C.: Denotational semantics of **go to**, SLN-CS, 61, pp. 278-304 (1978).
6) Milner, R.: Implementation and applications of Scott's logic for computable functions, Proc. ACM Conf. Proving Assertions about Programs (1972).
7) Milner, R. and Weyrauch, R. Proving compiler correctness in a mechanied logic, Machine Intelligence 7, pp. 51-70 (1972).
8) Plotkin G. and Smyth, M.: The category theoretic solution of recursive domain equations, Proc. IEEE-FOCS (1977).
9) Reynolds, J.: On the relation between direct and continuation semantics, SLN-CS, 14, pp. 142-156 (1974).
10) Scott, D. Models for various type-free calculi, Logic, Methodology and Philosophy of Science, pp. 157-187 (1973).
11) Scott, D.: Data types as lattices, SIAM J. Computing, 5, pp. 522-587 (1976).
12) Scott D. and Strachey, C.: Toward a mathematical semantics for computer languages, pp. 9-46, Proc. Symp. Computers and Automata, Polytechnic Inst. of Brooklyn Press (1971).
13) Strachey, C.: Towards a formal semantics, Formal Language Description Languages, pp. 198-220, North-Holland (1966).
14) Stoy, J.: Denotational Semantics, MIT Press (1977).
15) Tennent, R. The denotational semantics of programming languages, CACM, 19, 8, pp. 437-453 (1976).
16) 伊藤：プログラミング言語の意味論，情報処理，第 21 巻 6 号 (1980).

* そこでは述語 p が真なら e，偽なら $\bot$ を対応づけた。すなわち『$e \triangleq$ **if** p **then** e **else** $\bot$』としている（ここに e は単位演算子とする）．

[註]

「プログラム理論とその応用」は，当初，概論的な事柄だけを紹介する 4 回の連載で計画し，プログラム図式，プログラム束の理論，プログラム論理，並行プロセスの理論とそれらの応用について浅く広く解説する予定であった．しかし関数型プログラミングへの関心の高まりや編集委員のアドバイスを参考として概説的なものでなく詳論的なものとした．このために，プログラム論理（含，Hoare の公理系，dynamic logic, intensional logic），並行プロセスの理論（含，モデルとその記述能力，公理系）については 4 回の範囲内では言及できないことになった．これらの事柄とともに，数学的仕様記述・プログラム変換などについては別の機会に連載を継続できればと考えている．その際，最近の話題の 1 つである PROLOG の Denotational Semantics についても言及したいと考えている．

この連載を振り返ってみると説明不足や careless error が多々あり，読者諸賢よりのアドバイスや意見を頂ければ幸甚である．（昭和 56 年 11 月 24 日受付）