数理解析研究所講究録
第119巻 1971年 49-62

Mathematical Theory of Computation における D. Scott の方法

京大 数研 小野 寛晰

§0. D. Scott による mathematical theory of computation を[1]に基づいて紹介する。今までこの方面でおこなわれてきた研究は、semantics の定義等が必ずしも明快であるとはいえない。Scott は program 全体を束としてとらえ完備束上の連続関数の性質をうまく利用して、"loop" を持つ program (又は flow diagram) を自然に導入し、更に program の semantics をエレガントに定義している。なおここではふれないが、Strachey との共著論文[3]で、ここで展開される semantics の記述法を具体的な program に対し、適用を試みている。program は基本的な構成要素からいかに構成されているかという operational な面と、program の"内容"である、関数としての機能、いいかえるならば mathematical な面の、二つの面を持つと考えられる。mathematical theory of computation

は、この二つの面の関係を調べることである。そこでまず operational な面から見て, program を表わすような (formal な) expression を記述しなければならない。次にその expression が mathematical にはどのような関数をあらわしているかを定義する(§2.)。

いま expression の基本的な構成要素として関数 f_0, f_1, f_2, ---, 及び述語 b_0, b_1, b_2, --- が与えられているとする。expression の基本的な operation としては次の二つが考えられる。

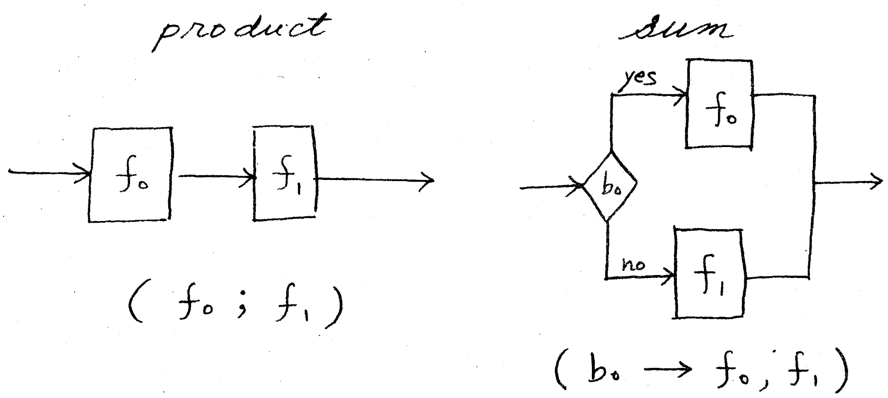


図 0.1

f_0, f_1, --- から product と sum により構成される expression を finite expression とよぶことにする。明らかに finite expression だけでは program 全体を表現することはできない。たとえば次のような, loop を持つ flow diagram を考える。

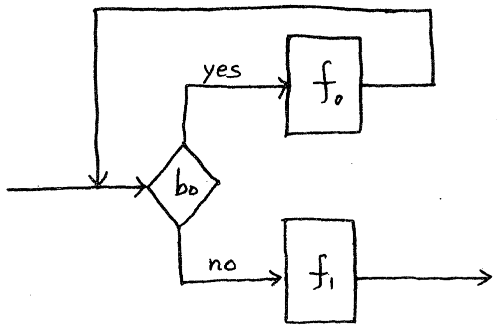


図 0.2

上の program 全体を g と書けば

$$g = (b_0 \to (f_0 ; g), f_1)$$

となる。このような g も expression とするために，"partial" な expression をみとめ，上の g のような program を finite (partial) expression により"近似"することを考える。例えば，上の g に対し，次のような finite expressions $\{g_n\}$ を考える。(但し $\perp$ は内容的には totally undefined function を表わしている。)

$$g_0 = \perp$$

$$g_{n+1} = (b_0 \to (f_0 ; g_n), f_1)$$

そうすれば g は $\{g_n\}_{n=0}^{\infty}$ の極限として表わされるだろう。そこでこの近似の考え方を次のように formulate してみる。いま d, d' を二つの expression とした時に $d \subseteq d'$ とは，ある expression に対し d' が d より"よい"近似になっていることを示す。(例 $g_n \subseteq g_{n+1}$)。$\subseteq$ は順序関係であり，更にこの順序に対し product と sum が次の意味で単

調になっていると考えるのは自然である。

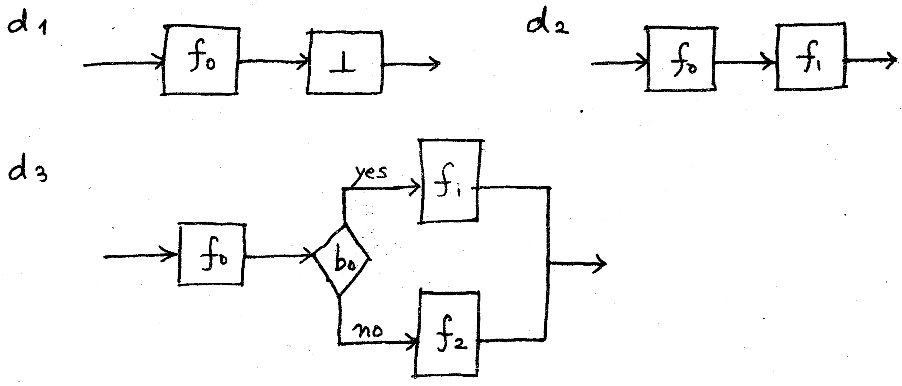
$d_0 \subseteq d_1$ かつ $d_0' \subseteq d_1'$ ならば

1) $(d_0 ; d_0') \subseteq (d_1 ; d_1')$

$$(0.3)$$

2) $(b \to d_0, d_0') \subseteq (b \to d_1, d_1')$

symbol $\perp$, $\top$ を導入し、それぞれ "underdetermined" program, "overdetermined" program を表わすものとする。従って任意の expression d に対し $\perp \subseteq d \subseteq \top$ がなりたつ。このように順序を入れると、finite expression 全体は最小元 $\perp$，最大元 $\top$ を持つ束と考えることができる。（例えば $d \cup d' = \top$ は d と d' が incompatible な program であることをいっている。）



$d_1 \subseteq d_2$　　$d_1 \subseteq d_3$　　$d_2 \cup d_3 = \top$

そして、finite expression 全体のなす束を§1で説明する方法で完備化すれば、loop を含むような program も expression としてとらえることが可能になる。

§1. expressionのなす束

expression のなす束を具体的に定義する前に[1]のkeyになっている定理(定理1.2)にふれておく。

束Dの部分集合Xの任意の有限部分集合がXの中に上界を持つ時、Xは有向(directed)であるという。

定義1.1 関数 $f: D \to D'$ (D, D'はともに束)が連続であるとは、Dの任意の有向部分集合Xに対し

$$f(\bigcup X) = \bigcup' \{f(x) \mid x \in X\}$$

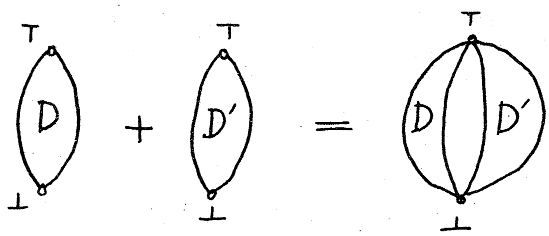
がなりたつことをいう。

定理1.2 Dを完備束,fをD→Dなる連続関数とする。とfは不動点を持つ。更に不動点のうち最小のものが存在する。(最小の不動点をY(f)と書き、Yを minimal fixed point operator とよぶ。)

証. ⊥をDの最小元とし、$p = \bigcup_{n=0}^{\infty} f^n(\perp)$ とする。但し f^0 は identity function, $f^{m+1} = f \cdot f^m$ とする。すると pがfの最小の不動点になる。

二つの完備束D, D'があたえられた時, union $D+D'$ 及び product $D \times D'$ を次のように定める。但し union を考える場合には DとD'とが disjoint であるとする。

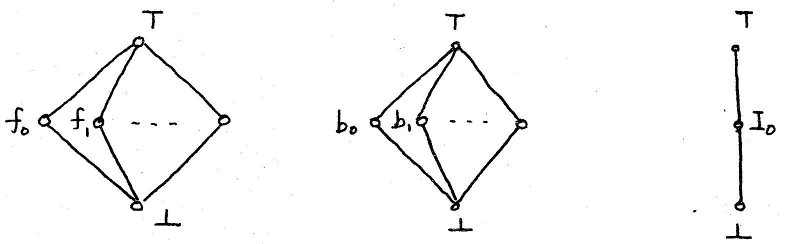
定義 1.3



定義 1.4　$D \times D'$ は集合 $\{(d, d') \mid d \in D,\ d' \in D'\}$ に束演算を component wise に定義して得られる束である。

明らかに $D + D'$ 及び $D \times D'$ も完備になる。

まず finite expression のなす束を作る。基本的な関数 $f_0, f_1, \cdots$ 及び述語 $b_0, b_1, \cdots$ に対し次のような束 F, B を作る。



F　　B　　I

図 1.5

又, identity function I_0 に対し, 上のような束 I を作る。$D_0 = F + I$ とする。D_0 は基礎となる関数のみからなる束である。次に D をすでに得られた, expression のある集合のなす束とするとき $\{(d; d') \mid d \in D,\ d' \in D'\}$ 及び

$\{(b \to d, d') \mid b \in B,\ d, d' \in D\}$ も又束をなすと考えられる。ところが(0.3)よりこれらの束は $D \times D'$ 及び $B \times D \times D$ と同一視できることがわかる。したがって

<u>定義 1.6</u>　　$D_0 = F + I$

$$D_{n+1} = D_0 + (D_n \times D_n) + (B \times D_n \times D_n)$$

(カンジを出すために $D_{n+1} = D_0 + (D_n ; D_n) + (B \to D_n, D_n)$ とも書く。)

<u>定理 1.7</u>　　$m \leq n$ ならば D_m は D_n の部分束である。

さて集合 $\bigcup_{n=0}^{\infty} D_n$ を考えてみる。定理1.7より $\bigcup_{n=0}^{\infty} D_n$ は束となることがわかるが、完備ではない。($\bigcup_{n=0}^{\infty} D_n$ は finite expression 全体の集合である。)そこで $\bigcup_{n=0}^{\infty} D_n$ を完備化することを考える。

<u>定義 1.8</u>　　$\psi_m : D_{m+1} \to D_m$ を m についての帰納法を用いて次のように定義する。

$$\psi_0(d) = \begin{cases} d & \text{if } d \in D_0 \\ \perp & \text{otherwise} \end{cases}$$

$$\psi_{n+1}(d) = \begin{cases} d & \text{if } d \in D_0 \\ (\psi_n(d'); \psi_n(d'')) & \text{if } d = (d'; d'') \\ (b \to \psi_n(d'), \psi_n(d'')) & \text{if } d = (b \to d', d'') \end{cases}$$

(但し $d', d'' \in D_n$)

mに関する帰納法を用いて次のことがわかる。

<u>定理1.9</u>　$d \in D_{m+1}$ に対し

$$d \in D_m \iff \psi_m(d) = d$$

<u>定理1.10</u>　各mについて

1) ψ_m は連続関数

2) ψ_m は D_{m+1} の各元に対し D_m の元で best approximation になるものをあたえる関数である。つまり $\psi_m(d) \subseteq d$, かつ $d' \in D_m$ かつ $d' \subseteq d$ ならば $d' \subseteq \psi_m(d)$.

さて次のような $\bigcup_{n=0}^{\infty} D_n$ の元の sequence $\langle d_n \rangle_{n=0}^{\infty}$ を考えてみる。

1) 各nについて $d_n \in D_n$

2) 各nについて $\psi_n(d_{n+1}) = d_n$

定理1.10 2)と考えあわせるとこのような列 $\{d_n\}_{n=0}^{\infty}$ の<u>極限</u>を $\bigcup_{n=0}^{\infty} D_n$ につけ加えてやればよいことになる。このことを formal に表現すると次のようになる。

<u>定義1.11</u>　Eを次の 1), 2)をみたすような $\bigcup_{n=0}^{\infty} D_n$ の sequence $\langle d_n \rangle_{n=0}^{\infty}$ の集合とする。

1) 各nについて $d_n \in D_n$

2) 各nについて $\psi_n(d_{n+1}) = d_n$。

更に E の上の順序 $\subseteq$ を次のように定める。

$\langle d_n \rangle_{n=0}^{\infty} \subseteq \langle e_n \rangle_{n=0}^{\infty} \iff$ 各nについて $d_n \subseteq e_n$。

定理1.12　Eは完備束である。

このEをexpression全体のなす束とよぶことにする。いま$d \in D_m$とし、$\langle d_n \rangle_{n=0}^{\infty} \in E$で$n \geq m$なるすべての$n$に対し$d_n = d$となるものを$d^*$と書く。すると$\{d^* \mid d \in D_m\}$と$D_m$とは束として同型になることがわかる。従って$d$と$d^*$を以後同一視して考えることにする。つまり，各$m$について$D_m$は$E$の部分束となる。更に任意の$d = \langle d_n \rangle_{n=0}^{\infty} \in E$について$d = \bigcup_{n=0}^{\infty} d_n$がなりたつ。$E$の元$\langle d_n \rangle_{n=0}^{\infty}, \langle d'_n \rangle_{n=0}^{\infty}$に対しproductとsumを次のように定める。

$$(\langle d_n \rangle_{n=0}^{\infty} ; \langle d'_n \rangle_{n=0}^{\infty}) = \langle (d_n ; d'_n) \rangle_{n=0}^{\infty}$$

$$(b \longrightarrow \langle d_n \rangle_{n=0}^{\infty} , \langle d'_n \rangle_{n=0}^{\infty}) = \langle (b \rightarrow d_n , d'_n) \rangle_{n=0}^{\infty}$$

定理1.13　$E = D_0 + (E ; E) + (B \rightarrow E, E)$

Eが図0.2のようにloopを持つprogramのexpressionを本当に含んでいることを次に示す。

図0.2の例は

$$g = (b_0 \rightarrow (f_0 ; g), f_1)$$

と書ける。

関数 $\Phi : E \longrightarrow E$ を次のように定義する。

$$\Phi(x) = (b_0 \to (f_0; x), f_1)$$

明らかに Φ は連続である。すると定理1.2，定理1.12より Φ は不動点 $g(\in E)$ を持つ．$\Phi(g)=g$。したがって $g=(b_0 \to (f_0; g), f_1)$ となる g が E の中に存在することが証明された。これは次のように一般化できる。

finite expression の中にあらわれる関数を適当に variable $x_0, x_1, \cdots$ でおきかえたものを polynomial とよぶことにする。

定理1.14　$\Pi_0(x_0, \cdots, x_m), \cdots, \Pi_m(x_0, \cdots, x_m)$ を，$x_0, x_1, \cdots, x_m$ 以外の variable は含んでいないような $m+1$ 個の polynomial とする。すると

$$x_0 = \Pi_0(x_0, \cdots, x_m)$$

$$\cdots\cdots$$

$$x_m = \Pi_m(x_0, \cdots, x_m)$$

の解 $(d_0, \cdots, d_m)$ が E の中に存在する。

証．関数 $Ⓗ: E^{m+1} \longrightarrow E^{m+1}$ を次のように定義する。

$$Ⓗ(x_0, \cdots, x_m) = (\Pi_0(x_0, \cdots, x_m), \cdots, \Pi_m(x_0, \cdots, x_m))$$

Ⓗは連続であるから，定理1.2よりⒽの不動点 $(d_0, \cdots, d_m)$ が存在する。（すなわち $(d_0, \cdots, d_m) = Y(\lambda x_0 \cdots \lambda x_m\ Ⓗ)$。）

上のような polynomial の解として表わされるような expression は program としても意味のあるものだが，E

はこのような解として表わされないような expression を非可算個含んでいる。次に一例を示す

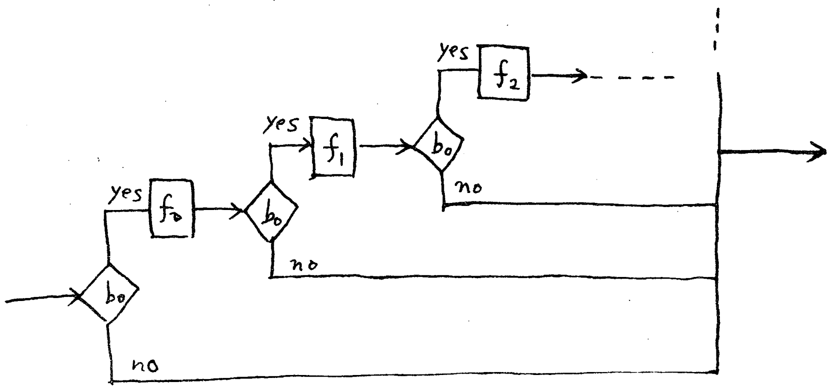


図 1.15

§2. program の semantics

<u>定義 2.1</u>　D, D' を完備束とする。$[D \to D']$ を D から D' への連続関数の集合に，次の順序をいれたものとする。

$$f \subseteq g \iff f(d) \subseteq' g(d) \text{ for any } d \in D.$$

<u>定理 2.2</u>　$[D \to D']$ は完備束をなす。

次に semantics の記述に必要な束 T 及び relation $\supset$ を定義する。

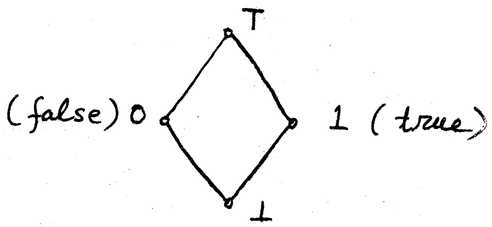


図 2.3 束 T (the lattice of truth values)

任意の束Dに対し"⊃"は $T\times D\times D \longrightarrow D$ なる関数で次のように定義される。但し $\supset(t, d, d')$ と書くかわりに $t\supset(d, d')$ と書く。

$$t\supset(d, d') = \begin{cases} d\cup d' & \text{if } t=\top \\ d & \text{if } t=1 \\ d' & \text{if } t=0 \\ \perp & \text{if } t=\perp \end{cases}$$

Eの semantics を考える際に次の仮定をおく。

<u>"information"全体は完備束をなし，更に"information"の変換は連続的におこなわれる。</u>

具体的には次のように表現される。まず 1)〜3) をみたすような triple $(\mathcal{S}, \mathcal{F}, \mathcal{B})$ を考える。

1) $\mathcal{S}$ は完備束

2) $\mathcal{F} \in [F \to [S \to S]]$ で $\mathcal{F}(\perp)=\perp$ かつ $\mathcal{F}(\top)=\top$

3) $\mathcal{B} \in [B \to [S \to T]]$ で $\mathcal{B}(\perp)=\perp$ かつ $\mathcal{B}(\top)=\top$。

このような triple $(S, \mathcal{F}, \mathcal{B})$ に対し，次のような

$\mathcal{V} \in [E \to [S \to S]]$ を考える。

任意の $d \in E$ 及び $\sigma \in S$ に対し

$$\mathcal{V}(d)(\sigma) = \begin{cases} \mathcal{F}(d)(\sigma) & \text{if } d \in F \\ \sigma & \text{if } d = I_0 \\ \mathcal{V}(d_2)(\mathcal{V}(d_1)(\sigma)) & \text{if } d = (d_1 ; d_2) \\ \mathcal{B}(b)(\sigma) \supset (\mathcal{V}(d_1)(\sigma), \mathcal{V}(d_2)(\sigma)) & \\ \qquad\qquad \text{if } d = (b \to d_1, d_2) \end{cases}$$

このような$\mathcal{V}$が存在することは次のようにして確かめられる。上の式において右辺を$\mathcal{H}$とあらわすと

$\mathcal{H} : [E \to [S \to S]] \to [E \to [S \to S]]$ は$\mathcal{V}$についての連続関数とみなせる。従って定理1.2より$\mathcal{H}$の不動点として$\mathcal{V}$が定まるが，特に$\mathcal{V}$を$\mathcal{H}$の最小の不動点にとる。このように$(S, \mathcal{F}, \mathcal{B})$から定まる$\mathcal{V}$のことを$E$の<u>解釈</u>とよぶ。このようにして得られる解釈は通常の解釈と一致している。(例えばdが recursion schema で定まるような expression であるならば$\mathcal{V}(d)$は recursion schema に対して定義されているdの解釈と一致する。[4], [5]参照。)

<u>定義 2.4</u>　Eの二つの元 d, d' が<u>同値</u>である$(d \cong d')$とは，任意のEの解釈$\mathcal{V}$に対して$\mathcal{V}(d) = \mathcal{V}(d')$となることをいう。

REFERENCES

[1] D. Scott, The lattice of flow diagrams, (1970).

[2] D. Scott, Outline of a mathematical theory of computation, Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems (1970).

[3] C. Strachey and D. Scott, Mathematical semantics for two simple languages, (1970).

[4] D. Scott, Some definitional suggestions for automata theory, J. Computer and System Sciences, 1 (1967).

[5] J. McCarthy, A basis for a mathematical theory of computation, in "Computer Programming and Formal Systems," North-Holland, 1963.