PL/I の形式的定義について (1)*

情報処理学会 PL/I 研究委員会**

1. 言語の形式的定義

1.1 言語の形式的定義の目的とその試み

プログラミング言語が高度化・複雑化してくると，従来の散文で書かれた文法書や仕様書は厚くなるばかりか，不正確・不完全・あいまいな点が増加し，読む人によって解釈が異なることが多くなる．そのため，コンパイラの製作者や利用者の間での解釈の統一と，コンパイラの互換性を保証する仕様には，不適当である．このようなことを解決するために考えられたのがプログラミング言語の形式的定義 (formal definition) である．これは，ひとことでいうと，言語の文法を数式に似たような式で書き表わし，一定の規則を形式的に適用するだけで，すべてを表現しようとする手法である．

数学では各種の式が多く使われ，そのために紙面が少なくて，しかも，正確に意味・目的を伝えている．もし，たとえば，行列の演算や微分方程式が散文でしか書けないとしたら，大変わずらわしいことになり，数学や自然科学は，今日ほど発展しなかったであろう．

自然に存在するものを観察し，分析し，体系付け，抽象化して理論にまとめていくことが，自然科学の研究の基本的態度であるが，プログラミングの分野でもコンピュータの利用の発展に伴って，自然発生的に生まれてきた高度，かつ，複雑な言語を論理的に定義付けようとする努力が，ここ数年来行なわれている．かくして生れてきたのが，形式的定義なる言語理論である．

こうした理論的研究の先駆者は，J. McCarthy[4]，P. J. Landin[5]らであり，いずれも ALGOL 60 のサブセットについて，形式的定義を試みている．これらの思想を引き継ぎ，PL/I を例として発展させて，本格的な形式的定義付けとして，IBM ウィーン研究所にて 1965 年9月より進められ，開発されたのが，PL/I の普遍的言語定義である．

そして，これは PL/I に限らず，他の言語にも広く適用でき，すでに ALGOL についても，形式的定義が完成している[7]．

IBM のウィーンの研究所は，これら言語の定義について，最新の研究に基づく方法を応用し，基礎概念の展開によって，あいまいな点がない定義を完成させた†．

1.2 構文の定義と意味の定義

ここでいう構文の定義と意味の定義は，それぞれつぎを定義する規則である．

1) 構文 (syntax)
   - 与えられた文字の集合より作られる列(string)
   - これらの列のそれぞれの構造 (structure)
2) 意味 (semantics)
   - 構文により決められた表現の解釈 (interpretation)

構文により定義された列のみが，言語の正しい表現 (expression)——プログラミング言語ではプログラム——とみなされて，解釈の対象となる．構文が表現に与えた構造は，後で解釈の際に重要である．

意味の定義には，解釈を行なう抽象機械 (abstract

---

*On the formal definition of PL/I (1)—a report of the PL/I research committee of the IPSJ, by Shinichi Ikeno (Electrical Communication Laboratory of NTT), Katsuhiko Arai (Electrical Communication Lab.), Kazuhiro Fuchi (Electrotechnical Lab.), Tohru Takeshita (IBM Japan Ltd.), Hideshi Karibe (IBM Japan Ltd.), Takeshi Sunohara (Mitsubishi Electric Co. Ltd.), Tadanori Mizuno (Mitsubishi Electric Co. Ltd.), Tetsuo Takahashi (Nippon Electric Co. Ltd.)

**主査 池野信一（電気通信研究所），委員 新井克彦（電気通信研究所），淵 一博（電気試験所），竹下 亨（日本ＩＢＭ），苅部英司（日本ＩＢＭ），春原 猛（三菱電機），水野忠則（三菱電機），高橋哲夫（日本電気）

情報処理学会 PL/I 研究会は，1968 年6月に発足し，IBM ウィーン研究所でまとめられた普遍的言語定義 (Universal Language Definition, ULD) に関する一連の報告書[1)~3)]を研究してきたが，1969 年8月までに 22 回の会を重ね，ほぼその内容をカバーしたので学会誌の紙面を借りて，本号より数回にわたって，その一部を報告させていただくことにする．

ＵＬＤの報告書は数冊にわたる大部のものであり，全部を紹介することは困難であるので，プログラミング言語の形式的定義を理解するのに必要なことに限定したい．

なお，文献 3) の内容を部分的に翻訳して本誌に使用することについて，IBM 社の正式な了承を得ている．

† その経過はつぎのとおりである．

PL/I の形式的定義の開発経過

1965 年	9 月	形式的定義の開発開始
1966 年	12 月	〃　第1版完成
1967 年	12 月	〃　第2版完成
1968 年	6 月	〃　校訂版完成

第2版完成後に，つぎの版の作成に取り掛かり，すでに出ている報告書に含まれる定義の更新と展開を行なっている（追記：1969年6月に第3版が完成した）．

machine) を考えるが，抽象機械はその状態集合 (set of states) と状態推移関数 (state transition function) で特徴づけられる．初期状態 (initial state) はプログラムと入力データにより決まり，機械のその後の行動 (behavior) は与えられたデータに対するプログラムの解釈を定義する．

解釈を行なう抽象機械が構文で定義された文字の列をいきなり処理するのは大変である．どのプログラムにも構文によって，唯一の解剖木 (parsing tree) が決まるので，それをプログラムの算法 (algorithm) と考えることができる．それでも，原始プログラムでは，同じことが何とおりでも書けるので，同一内容を持つプログラムを同一の形に直す翻訳機 (translator)——抽象的なもので，翻訳の方法と考えてもよい——を作用させる．こうすると，プログラムは，木構造 (tree structure) を持つ標準形のものとなって，抽象機械により解釈される．

プログラム——一般には言語の表現——を表わすとみなされる対象の類 (class of object) の定義は，抽象構文 (abstract syntax) と呼ばれ，これに対して，文字の列の集合としてプログラムを規定する構文上の規則を具象構文 (concrete syntax) という．PL/I の具象構文は拡張した Backus 記法で表わされる．

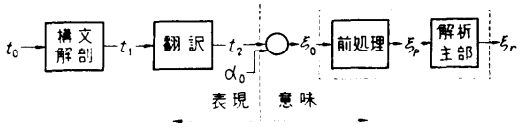
解釈を行なう抽象機械の状態が，プログラムを表わす対象の類と同種の対象の類として表わせると好都合である．そうすると，機械の状態の集合がプログラムの抽象構文の場合と同じ方式によって，定義付けられる．

そこで，木構造を持つ一般の対象の類(general class of object) を考え，その部分類 (subclass) として言語の表現——プログラム——を取り扱い，その解釈を行なうことにしたのが，ULD に採用されている方式である．

1.3 形式的定義の構成

PL/I の形式的定義の基礎とみなされる原始プログラムから実行までの過程は，**第1.1図**のように表わされる．

この処理過程にはいる入力は，文字の列より PL/I なるの原始プログラム — 具象プログラム——t_0 とある初期データ・セット d_0 である．具象プログラムは Backus の標準形の拡張で表わされる具象構文で定義される．具象プログラムを第1の段階で，構文の解剖木 (parsing tree) の形に直す．その際に，便宜上の記述（コメントなど）は除き，構造に無関係なもの（たとえば，宣言文）を標準形に変え，構造に意味があるものは解剖木のままにしておく．その後の解釈は，初期状態 ξ_0 の抽象機械により定義される．その ξ_0 は抽象プログラム t_0 と初期データ・セット d_0 により決まる．この機械はプログラムを解釈しながらつぎつぎとその状態を変えていき，終了状態 (end states) に達する．この一連の状態の系列は計算 (computation) と呼ばれる．形式的定義により定められた解釈機 (interpreter) は，与えられた PL/I のプログラムとデータに対して，原則として1つの計算を生み出す（厳密にいうと，コンパイラごとに決まる要素があり，解釈機が一意に決められないので，同じプログラムに対して幾通りかの計算が生成される場合がある.)，



第1.1図 形式的定義の行程

抽象機械のいずれの状態も，1つの抽象的対象であり，抽象プログラムの構文に適用されるのと同じ形式的手法が，この機械の状態に適用される．抽象機械の状態の本質的構成は状態の抽象構文により示される．状態の抽象構文により定義される状態の集合は，与えられた抽象プログラムとデータ・セットに対して取りうるすべての状態の集合を含む．

解釈機による処理過程は，前処理と解釈主部の2つに大きく分けられる．

前処理 (pre-pass) はつぎを行なう．

1) 静的変数 (static variable) の割付けと初期値設定．
2) 制御される変数 (controlled variable) の空なる割付けを行ない，以後の ALLOCATE 文による割付けの準備．
3) 外部宣言 (external declaration) の範囲 (scope) の結び付け (linkage).
4) 前処理中に機械の状態の中にはいる項目(entry) と宣言との間に必要な結び付けをするためのプログラムの中の宣言に適当な情報をそう入．

中間の状態 ξ_p は 4) により修正された抽象プログラムを含む．

最後に，解釈主部 (proper interpretation) が，個個の文 (statement) の意味に従って，すでに前処理を受けている抽象プログラムの解釈を行なう．

2. 簡単なプログラミング言語——EPL——について

本稿では，プログラミング言語の形式的定義に用いられる様々な手法を紹介していく際に，ごく簡単な，プログラミング言語——これを EPL (Example Programming Language) と呼ぶ——を定義し，それに沿って考察を進めていくこととする．

EPL は PL/I によく似ていはいるが，全く独自のプログラミング言語であり，PL/I の部分集合ではない点に注意する必要がある．

まず，EPL で扱いうるデータは，真偽値と整数値である．これらのデータに対して，2項演算のみが存在するものと仮定する．

式 (expression) は定数，変数および関数指示子を演算子で結んだ形で組み立てられ，値を計算するための規則を表わす．計算は，値を生み出すことのほかになんらかの形で（副次効果として）機械の状態を変えることもできる．

定数は値を表わす．変数は，値を表わす識別子である．変数によって表わされる値は代入文の実行を通じて動的に変わりうる．ある特定の変数の値域は，直偽値か整数値に限られる．

関数指示子は，1つの関数識別子と識別子のリスト (空リストでもありうる) である1つのアーギュメント・リストとから成る．関数識別子はアーギュメントに基づいてある値を計算するための規則を表わし1つの文 (ブロックも1つの文である.) と1つの式に対応する．関数の値はその文を実行し，その後で関数の値を生み出す式を評価することによって決定される．

実行可能な文として，代入文，条件文およびブロックの3種類がある．代入文は1つの変数である左辺と1つの式である右辺とから成る．これが実行されると式が評価され，計算された値が左辺である変数に代入される．この変数がつぎに変更が起こるまで，その式によって計算された値を表わす．

条件文は，1つの式と，2つの文——THEN 文と ELSE 文——とから成る．これが実行されると式が評価され，もし必要ならその値が真偽値に変換される．もしも，その値が真なら THEN 文が実行され，そうでなければ ELSE 文が実行される．

関数の起動に伴うアーギュメントの引渡しは，それぞれのパラメタが対応するアーギュメントとそれらの値域内で完全に同義であるような形で定義される．

ブロックは，宣言部と任意の型の文のリストから成る．宣言部はある識別子が(変数整数または論理変数)であるか関数識別子であるかを宣言する．識別子によって表わされる関数は，その宣言の中で与えられる．つまり，関数の場合には関数識別子の宣言と同時にパラメタ・リスト，文および式も宣言の中で定義されるわけである．

ブロックが実行されると，宣言部で宣言された識別子が，その宣言に従って新しい実体を表わす名前として導入され，ブロックの実行が終わるまでこの意味を保つ．前に宣言された識別子と同じ識別子が新たに宣言されると，前者の意味はそのブロックの終わりまですべて抑止される．あるブロックの宣言部で宣言された識別子は，そのブロックに局所であるという．関数のパラメタは，その関数の実行時に，あたかもそれらが，その関数の中で宣言されているかのように扱われる．これは，関数のパラメタがそれぞれの関数の非局所識別子であると考えられることを意味する．これらのパラメタの意味は，その関数が宣言されているブロックの中で，同じ識別子が新たに宣言されると凍結される．ここで述べる言語によるプログラムは常にブロックである．

3. 表 記 法

本稿では様々な表記法を使用するが，ここでは，その中からすでに一般に知れわたっているものを取り出してまとめてみる．これらの表記法や記号の主要部分は LISP や述語計算，あるいは通常の数式や関係式，集合論などから，その伝統的な意味で採用したものである．これらの記号や約束は，とくにこれ以上の注釈なしで使用する．

3.1 条 件 式

形 式

$(P_1 \rightarrow e_1,\ P_2 \rightarrow e_2, \cdots\cdots P_n \rightarrow e_n)$

P_i: 真偽値を表わす式

e_i: ある対象 (e_i の値) を表わす式

このコンマを取り去った，別な形式を用いることができる．

$P_1 \rightarrow e_1$
$P_2 \rightarrow e_2$
$\vdots$
$P_n \rightarrow e_n$

意 味

条件式は，i が，それに対して P_i が真でそれ以前のすべての P_j (ただし，$1 \leq i \leq n$, $1 \leq j \leq i$) が偽であ

るような最も小さな整数である場合の e_i の値を表わす．このような整数が無ければ，式は値を持たない．個々の条件 P_i の調べられる順番が左から右であるということが意味を持つことに注意する必要がある．もしも P_i が真ならば，上の定義の結果として，P_i の後にくるもの，たとえば P_k，$i<k\leq n$ の値は条件式の評価とは無関係であり，未定義でさえありうる．

3.2 等 式

= 等しい．

≠ 等しくない．

$\overset{=}{D_f}$ 定義により同等である．

3.3 真偽値，論理演算子および限定作用素

(1) 真偽値

T 真

F 偽

(2) 論理演算子

¬	否定	≡	同等
&	論理積	≢	非同等
∨	論理和(Vel)	⊃	含意

これらの意味は条件式を用いて定義することができる．

上記の演算子によって作られる式については，かっこの省略記法を用いうる．

印刷を容易にするために，多数項の合接および離接に“Et”および“Vel”の記号を使用する．

$$\overset{n}{\underset{i=1}{E_t}} P_i \underset{D_f}{=} ((n>0)\rightarrow P_1 \,\&\, P_2 \,\&\cdots\&\, P_n,\ (n=0)\rightarrow T)$$

$$\overset{n}{\underset{i=1}{\mathrm{Vel}}} P_i \underset{D_f}{=} ((n>0)\rightarrow P_1 \vee P_2 \vee \cdots \vee P_n,\ (n=0)\rightarrow F)$$

(3) 限定作用素

∃ 存在作用素

∀ 全称作用素

上記の記号はつぎの形の式で用いられる．

$$(\exists x_1, x_2, \cdots, x_n)\,(P(x_1, x_2, \cdots, x_n))$$

変数 $x_1, x_2, \cdots, x_n$ を式の束縛変数†† と呼ぶ．この式は，もしも，すべての可能な n 組ベクトル $x_1, x_2, \cdots, x_n$ に対して $P(x_1, x_2, \cdots, x_n)$ がこれら変数の値域内で真ならば真，そうでなければ偽である．

上の形の式の束縛変数の値域が，常に定義されていなければならないという点に注意する必要がある．これは式によって明確に定義されるか，変数名の特定類に，ある値域を関連づける手段を用いて，暗黙に定義されるかのいずれかである．便利のために，式の束縛変数部の代わりに，成分として束縛変数を含んでいる複合構造を書くことができる．たとえば

$$(\exists <x, y>)\,(P(x, y, <x, y>)).$$

(4) 記述作用素

ι イオタ作用素

この記号は，つぎの形の式で用いられる．

$$(\iota x)\,(P(x))$$

x を式の束縛変数と呼ぶ．この式は $P(x)$ が x の値域内で真であるような値を表わす．x の値域内で性質を持つ値が無いか2つ以上ある場合には，この式は値を持たない．

3.4 演算子と関係子

(1) 演算子

＋ 前置正号，2項正号

－ 前置負号，2項負号

× 乗算

(2) 関係子

<	小
≤	小または等しい
=	等しい
≠	等しくない
≥	大または等しい
>	大

3.5 集合演算子，関係子および集合表記法

(1) 集合演算子

∪ 和

∩ 積

(2) 関係子

∈	要素である
∉	要素でない
⊂	真部分集合である
⊆	部分集合であるか等しい
⊇	上部集合であるか等しい
⊃	真上部集合である

(3) 集合表記法

$\{a, b, c, \cdots\}$	要素 $a, b, c\cdots$ はこの集合の要素である
$\{\ \}$	空集合
$\{x \mid P(x)\}$	集合の暗黙の定義

x は式の束縛変数と呼ぶ．この式は $P(x)$ が真であるような全要素の集合を表わす．通常，便利のために式の束縛変数部の代わりに，成分を含んでいる複合構造を書く．たとえば，$P(x, y, <x, y>)$ であるような対 $<x, y>$ の集合は

†† 束縛変数とは，いかなる置換も許されない変数である．

$\{<x, y> \mid P(x, y, <x, y>)\}$

と書ける.

3.6 関数合成

∘ 関数合成作用素

この作用素は

$$(f \circ g)(x_1, \cdots, x_n) \underset{Df}{=} f(g(x_1, \cdots, x_n))$$

によって定義される. ただし, f および g は単純な関数名か関数を表わす式のいずれかでありうる. 後者の場合は式をかっこでくくらなければならない.

かっこの省略に関して, つぎの規則が適用される.

(a) 関数合成作用素は関数作用より結合力が強い

例: $f \circ g(x) = (f \circ g)(a)$

(b) $f_1 \circ f_2 \circ \cdots \circ f_m(x_1, \cdots, x_n) = f_1(f_2(\cdots(f_m(x_1, \cdots, x_n)\cdots))$

(c) $f(x_1)(x_2)\cdots(x_n) = (\cdots((f(x_1))(x_2))\cdots)(x_n)$

3.7 優先順位

かっこはつぎの優先順位規則に従って省略できる.

∘	高順位 (結合力が最も強い)
+, −	
×	
+, −	
<, ≤, =, ≠, ≥, ⊂, ⊆, ⊇, ⊃, ∈, ∉	
¬	
&	
∨	
≡, ≠	
⊃	低順位 (結合力が最も弱い)

4. 対 象

4.1 対象の一般類

ここでは, 言語の表現および抽象機械の状態を部分類として表現するために, 対象の一般類を導入する. ある対象は一般にいくつかの成分から構成されるが, 各成分自体も対象である, 対象の成分を示すのに名前を用いるが, この名前を**セレクタ**と呼ぶ, セレクタの集合 S は可付番無限集合である. 以下, 記号 s, s_1, s_2, …でセレクタを表わし, 記号 A, A_1, A_2, …で任意対象を表わすものとする.

与えられた対象 A の成分 s を取り出す操作を $s(A)$ で表わす, つまり, $s(A)$ は A の s 成分を表わす.

ここで考察する対象は, 言語表現や抽象機械の状態であり, いずれも有限の構造を持つものであるから, 有限の成分を持ち, 有限の深さの入れ子を持つものに限定される. したがって, 与えられた対象につぎつぎとセレクタを作用させていくと, 有限回でもはや分解できない成分に到達できる. この種の対象を基本対象 (elementary object) と呼び, この集合を EO, 個々の基本対象を $eo, eo_1, eo_2, \cdots\cdots$で表わす. 基本対象以外の対象を複合対象 (composite object) と呼ぶ.

名前 s をつけた対象 A は, $<s : A>$ で表わす. この記法を用いれば, 複合対象は名前のついた対象の有限集合として一意に記述できる. ただし, 各成分は複合または基本対象であってもよいが, 各名前は互いに異なっていなければならない. このことを

$\{<s_1 : A_1>, <s_2 : A_2>, \cdots, <s_n : A_n>\}$

と表わす. ここに $n \geq 1$ であり, $i \neq j$ なら $s_i \neq s_j$ である. 対象 $A_i (1 \leq i \leq n)$ を上記対象の**直属成分**と呼ぶ.

上述のように名前を持つ対象の集合として, 新たな複合対象を作ることを μ_0 関数で以下のように表わす.

$\mu_0(<s_1 : A_1>, <s_2 : A_2>, \cdots, <s_n : A_n>)$

対象はその成分の集合として一意に記述されるから成分の順序には関係しない. したがって, 同一の対象でも, 一般にはいくつかの表現方法がある.

4.2 対象のグラフ表現

対象は木 (tree) に表現して考えるとわかりやすい. つまり, 名前のついた枝 (branch) と, 有限個の端節 (terminal node) および端節に基本対象を与えたもので木を表現する.

前節で定義した線型表現のうち, 基本対象 $eo \in EO$ は縮退木として**第 4.1 図**(a)のように表わされ, 複合対象 $\{<s_1 : A_1>, <s_2 : A_2>, \cdots, <s_n : A_n>\}$ は, **第 4.1 図**(b)のように表わされる.

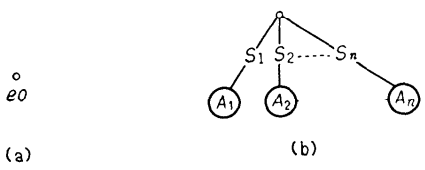
4.3 空対象

セレクタを対象に作用させるのは, そのセレクタによる成分を対象が持つ場合にのみ意味がある. つまり

$s(\mu_0(<s_1 : A_1>, <s_2 : A_2>, \cdots, <s_n : A_n>))$

が意味を持つのは, $S = si (1 \leq i \leq n)$ のときのみである.

セレクタの対象への作用 $s(A)$ を常に可能とするた



第 4.1 図 対象のグラフ表現

めに，空対象 Ω を導入する．空対象とは成分を持たない対象（つまり空集合）である．記号的には

$$\Omega \underset{Df}{=} \mu_0(\ )$$

と表わされる．セレクタに対応した成分を持たない対象に，そのセレクタを作用させた結果は Ω となるものと定義する．記号的には，下記のように表わせる．

$$s(\mu_0(<s_1 : A_1>, <s_2 : A_2>, \cdots, <s_n : A_n>)) = \begin{cases} A_i (s = si \mid 1 \leq i \leq n) \\ \Omega \ \text{(otherwize)} \end{cases}$$

特殊な場合として，つぎのことが導かれる．

$$s(eo) = \Omega,\ s(\Omega) = \Omega$$

$<s : \Omega>$ の形の名前のついた対象も μ_0 のアーギュメントに許されるが，その場合，つぎの関係が成り立つ．

$$\mu_0(<s_1 : A_1>, <s_2 : A_2>, \cdots, <s_n : A_n>, <s : \Omega>) = \mu_0(<s_1 : A_1>, <s_2 : A_n>, \cdots, <s_n : A_n>)$$

つまり，$<s : \Omega>$ は演算子 μ_0 に関する単位元と考えられる．

4.4 複合セレクタ

関数的合成との類推によって，複合セレクタ $s_1 \circ s_2 \circ \cdots \circ s_n$ をつぎのように定義する．

$$s_1 \circ s_2 \circ \cdots \circ s_n(A) \overset{Df}{=} s_1(s_2(\cdots(s_n(A))\cdots))$$

複合セレクタは，$\chi, \chi_1, \chi_2, \cdots, \chi_1', \chi_2', \cdots$ 等で表わす．前節の単位元と同様，恒等関数 I を導入する．

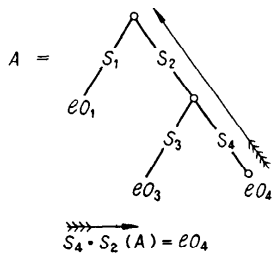
$$I(A) = A$$

$$I \circ \chi = \chi \circ I = \chi$$

複合セレクタ全体の集合を S^* と表わす．複合セレクタとして I を導入することによって，前節の同値関係は，対象が基本対象・複合対象であるかに関係なくつぎのように一般化できる．

$$(\forall \chi)(\chi(A_1) = \chi(A_2)) \equiv A_1 = A_2$$

第 4·2 図の例でも明らかなように，木表現の下から上への方向が，複合セレクタの左から右への方向に対応する．



第 4.2 図　複合セレクターの例

つぎに，複合セレクタ間の従属関係を述語 dep により定義しよう．2 つの複合セレクタ χ_1, χ_2 は，どちらかが，他方の尾（tail）であるとき互いに従属であるという．すなわち，述語 dep をつぎのように定義する．

$$\mathrm{dep}(\chi_1, \chi_2) \overset{Df}{\equiv} (\exists \chi)(\chi_1 = \chi \circ \chi_2 \vee \chi_2 = \chi \circ \chi_1)$$

従属関係について，つぎの 2 つの定理が成り立つ．

〔定理 1〕　$\mathrm{dep}(\chi_1, \chi_2) \equiv \mathrm{dep}(\chi_1 \circ \tau, \chi_2 \circ \tau)$

〔定理 2〕　$\mathrm{dep}(\tau \circ \chi_1, \chi_2) \supset \mathrm{dep}(\chi_1, \chi_2)$

4.5 演算子　μ

演算子 μ は 2 項演算でその被演算子は対象 A と対 $<\chi : B>$ である（χ は複合セレクタ，B は対象）。演算の結果は対象—A—である．すなわち，A の χ 成分は B で置換される．この演算は次のように書く．

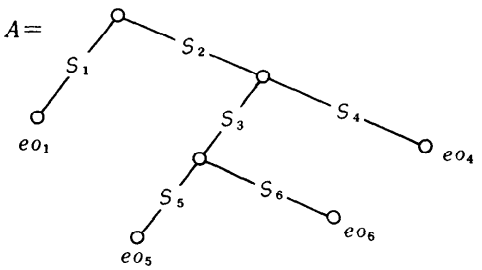
$$\mu(A ; <\chi : B>)$$

μ では，つぎの 2 つの特殊ケースがある．

（a）　$\chi(A) = \Omega$ なら，μ の結果 $\chi(A) = B$ である．

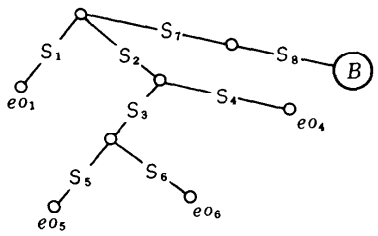
（b）　もし $B = \Omega$ なら，$\chi(A) = \Omega$ である．

例　$B \neq \Omega$

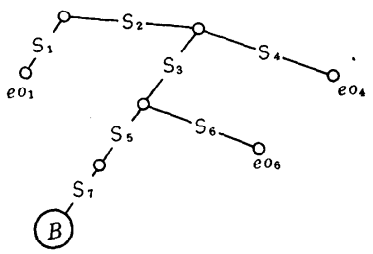


このとき，χ のいろいろな選択により $\mu(A ; <\chi : B>)$ の結果は，つぎのようになる．

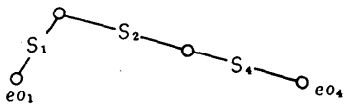
（1）　$\mu(A ; <s_8 \circ s_7 : B>) =$

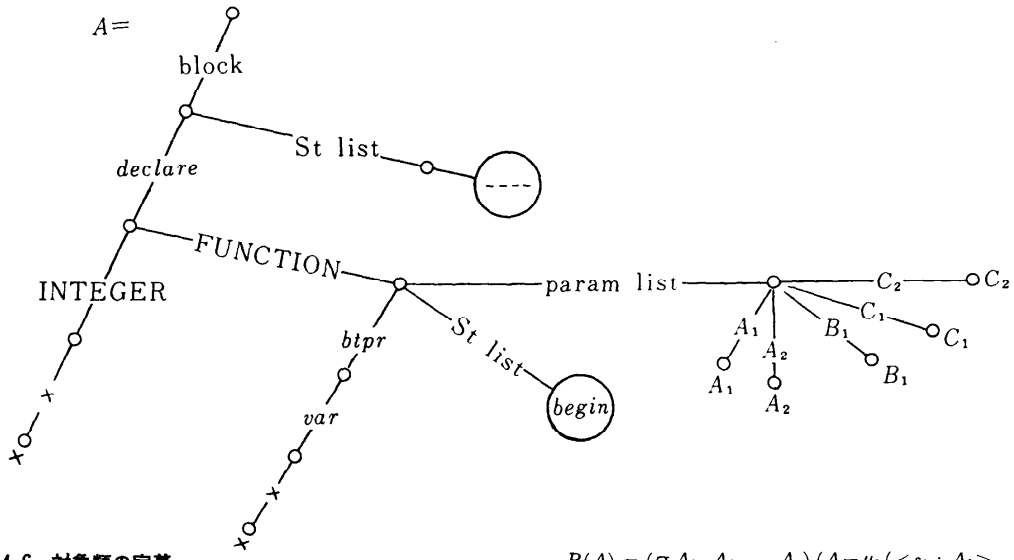


(2) $\mu(A; <s_7 \circ s_5 \circ s_3 \circ s_2 : B>) =$



(3) $\mu(A; <s_3 \circ s_2 : \Omega>) =$





4.6 対象類の定義

特定の言語の良形文 (well-formed sentence) は，対象の一般類の部分類であり，その記述は，述語論や集合論と同じ手段で与えられる．ある言語の良形文と同一視される対象の部分類に対する規定を，**抽象構文** (abstract syntax) と呼ぶ．

対象の部分類を定義することは，その部分類のメンバに対して，述語が真になると定義するものとする．$P, P_1, P_2, \cdots$ は任意の述語を，$\hat{P}, \hat{P}_1, \hat{P}_2, \cdots$ はそれぞれの述語で決定される対象の部分類を表わすとする．このとき，つぎの定義図式は，抽象構文の記述に使われる．

(1) 基本対象の部分類に対して真であるような述語 P が存在する．すなわち

$$\hat{P} \subset EO$$

である．

この述語の定義は，EO の集合の定義による．

(2) 述語 $P_1, P_2, \cdots, P_n$ を与えると，新しい述語 P をつぎのように定義できる．

$$P \underset{Df}{=} P_1 \vee P_2 \vee \cdots \vee P_n \dagger\dagger\dagger$$

また集合の形で，上の等式は

$$\hat{P} \underset{Df}{=} \hat{P}_1 \cup \hat{P}_2 \cdots \cup \hat{P}_n$$

とできる．

(3) n 個の述語 $P_1, P_2, \cdots, P_n$ と，互いに異なる n 個のセレクタ $s_1, s_2, \cdots, s_n$ ($i \neq j$ に対して $s_i \neq s_j$) が与えられれば，新しい述語 P はつぎの式で定義できる．

$$P(A) \underset{Df}{=} (\exists A_1, A_2, \cdots, A_n)(A = \mu_0(<s_1 : A_1>,$$

†††† 演算子 ▼∨▼ は命題にだけでなく述語にも適用できるよう拡張される．すなわち，定義

$$P(X) \underset{Df}{=} P_1(X) \vee P_2(X) \vee \cdots \vee P_n(X)$$

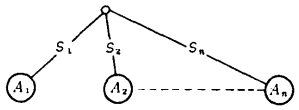
は

$$P \underset{Df}{=} P_1 \vee P_2 \vee \cdots \vee P_n$$

と略記できる．

$$<s_2 : A_2>, \cdots, <s_n : A_n>) \ \& \ \overset{n}{\underset{i=1}{E}}t(P_i(A_i))$$

上式の述語 P を木表示すると，つぎのようになる．



ここで，$A_1 \in \hat{P}_1, A_2 \in \hat{P}_2, \cdots, A_n \in \hat{P}_n$ である．

上の等式は，つぎのようにも書く．

$$P = (<s_1 : P_1>, <s_2 : P_2>, \cdots, <s_n : P_n>)$$

以上，(1)，(2)，(3)で定義した等式は，対象を解析するために使われ，これらは再帰的に定義できる．

とくに，(3)は明らかな含意を使って生成法 (production rule) として公式化できる．

すなわち

$$P_1(A_1) \& P_2(A_2) \& \cdots \& P_n(A_n) \supset$$
$$P(\mu_0(<s_1 : A_1>, <s_2 : A_2>, \cdots, <s_n : A_n>))$$

である．

$P_1(A_1)$ を満たす対象 A_1, $P_2(A_2)$ を満たす対象 A_2 …, $P_n(A_n)$ を満たす対象 A_n を与えると

$$A = \mu_0(<s_1 : A_1>, \cdots, <s_n : A_n>)$$

は $P(A)$ を満たす．このような対象をタイプ P の対象という．

逆に，タイプ P の対象に対して，定義図式(3)を使って，つぎのように定義できる．

すなわち

$$P(A) \supset P_1(s_1(A)) \& P_2(s_2(A)) \& \cdots \& P_n(s_n(A))$$

である．

(1)，(2)，(3)で定義される対象の類は

「その類に対し上限 N がある」

という性質を持つ．上限とは

「類のメンバとして N 個の直属成分だけを持つ」

ということである．

類に対して，直属成分が有界だという性質は時には不便である．つぎにこれを克服する定義法を述べる．すなわち，セレクタの無限集合を論ずる必要がある．セレクタ上の述語を Q, Q_1, Q_2 とする．

(4) 対象上の述語 P_1 と，セレクタ上の述語 Q を与えると，新しい述語 P が定義される．その述語 P は，$\hat{Q}$ からのセレクタと $\hat{P}_1$ の対象とから作られるすべての対象に適用できる．

したがって，さらに正確に，その定義図式が述べられる．すなわち

$$P(A) = (\exists A_1, A_2, \cdots, A_n, s_1, s_2 \cdots, s_n)$$
$$(\overset{n}{\underset{i=1}{E}}t(Q(s_i) \& A_i)) \& A = \mu_0(<s_1 : A_1>, <s_2 : A_2>, \cdots <s_n : A_n>))$$

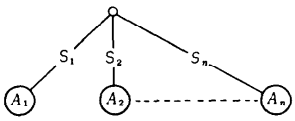
である．

木表示を使えば，上の等式は，「述語 P は，つぎに述べるような対象に対して真である」ことを示している．すなわち，任意の n と，制限

(a) $1 \leqq i \leqq n$ のとき $s_i \in \hat{Q}$

(b) $1 \leqq i \leqq n$ のとき $A_i \in \hat{P}_1$

に対して，対象は



である．

この等式は，つぎのような記法で表現する．すなわち

$$P = (\{<s : P_1> \,||\, Q(s)\})$$

である．

もっと一般的には，明らかにつぎのようになる．

$$P = (<s_1 : P_1>, \cdots, <s_n : P_n>,$$
$$\{<s : P_{n+1}> \,||\, Q_1(s)\}, \cdots$$
$$\{<s : p_{n+m}< \,||\, Q_m(s)\})$$

である．

ある適用のため，直属の部分成分の順序集合であるような対象を定義する必要がある．それは，セレクタまたはその部分集合に順序を定義することである．

このため，自然数をセレクタの集合に1対1に写像する関数"elem"を定義する．その形は

$$\mathrm{elem}(i)$$

である（ここで i は自然数）．

これは，$i \geqq 1$ に対して1つのセレクタを生み，つぎの式が成り立つ．

$$\mathrm{elem}(i) \neq \mathrm{elem}(j) \quad \text{ここで } i \neq j$$

したがって，「対象の類に対して，そのメンバが，任意個の直属の順序部分成分を持つ」と定義できる．このような対象を**リスト**と呼ぶ．特別な基本対象を導入し，これを空リストと呼び，つぎのように表現する．

$$< >$$

それで，つぎの定義図式を導入できる．

(5) 1つの述語を与えると，その述語に接尾語"-list"をつけ加えて，新しい述語

$$P\text{-list}$$

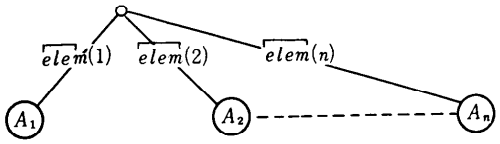
をつぎのような式で定義する．

$$P\text{-list}(A) \underset{Df}{=} A = < > \vee (\exists A_1, A_2, \cdots, A_n)$$

$$(\overset{n}{\underset{i=1}{E}}t\,P(Ai)\,\&\,\overset{n}{\underset{i=1}{E}}t(A_i \neq \Omega)\,\&\,n \geqq 1\,\&$$
$$A=\underset{\circ}{\mu}(<\mathrm{elem}(1) : A_1>, \cdots, <\mathrm{elem}(n) : A_n>))$$

である．

いい換えれば，上式は，つぎのことを表わす．すなわち，P-list は空リスト$<\ >$および下図のような対象に対して成り立つ．



ここで，$n \geqq 1$，また $1 \geqq i \geqq n$ に対して

$P(A_i)$，$A_i \neq \Omega$

である．

4.7 その他の表記法の約束

(2) μ 演算子の意味の拡張

対象の成分を，順序を考えずに置き換えを容易にするため，μ 演算子を拡張する．

(i) $\mu(A ; <\chi_1 : B_1>, <\chi_2 : B_2>, \cdots, <\chi_n : B_n>)$

これは，反復的につぎのように定義される．

$$\mu(A ; <\chi_1 : B_1>, <\chi_2 : B_2>), \cdots, \chi_n : B_n>)$$
$$\underset{Df}{=}\mu(\mu(A ; <\chi_1 : B_1> ; <\chi_2 : B_2>, \cdots, <\chi_n : B_n>)$$

ただし，$n=0$ のとき

$$\mu(A ;) = A$$

である．

(ii) $\mu(A ; \{<\chi : B> | P(\chi, B)\})$

この2番目のアーギュメントは，命題 $P(\chi, B)$ を満たす対 $<\chi : B>$ の有限集合である．空集合に対して $\mu(A ; A\ \{\ \}) = A$．また，2番目のアーギュメントは集合の和でもよい．

$$\mu(A ; \{<\chi_1 : B_1> | P_1(\chi_1, B_1)\} \cup \cdots \cup \{<\chi_m : B_m> | P_m(\chi_m, B_m))\})$$

(iii) $\mu_0(<\chi_1 : A_1>, <\chi^2 : A_2>, \cdots, <\chi_n : A_n>)$

$i \neq j$, $1 \leqq i, j \leqq n$ に対して χ_j は χ_j の尾でないという制限がとれる．μ_0 の意味はつぎのように再定義できる．

$$\mu_0(<\chi_1 : A_1>, <\chi_2 : A_2>, \cdots, <\chi_n : A_n>)$$
$$\underset{Df}{=}\mu(\Omega ; \{<\chi_1, A_1>, <\chi_2 : A_2>, \cdots, <\chi_n : A_n>)$$

(iv) $\mu_0(\{<\chi : A> | P(\chi, A)\})$
$$\underset{Df}{=}\mu(\Omega ; \{<\chi : A> | P(\chi, A)\})$$

(v) $\delta(A ; \chi_1, \chi_2 \cdots, \chi_n)$ は，A から χ_i 成分を除き次式で定義する．

$$\delta(A ; \chi_1, \cdots, \chi_n) \underset{Df}{=} \mu(A ; <\chi_1 : \Omega>, \cdots, <\chi_n : \Omega>)$$

(vi) $\delta(A ; \{\chi | P(\chi)\}) \underset{Df}{=} \mu(A ; \{<\chi : \Omega> | P(\chi)\})$

(2) リストの操作

リストは対象の重要な類である．リスト操作のいくつかの関数，演算子，略語をあげる．

(i) 述語 is-list

$$\text{is-list} \underset{Df}{=} (\exists P)\,(P\text{-list}(A))$$

(ii) リストの要素を表現する略記法

$$\mathrm{elem}(i, L) \underset{Df}{=} \mathrm{elem}(i)\,(L)$$

(iii) リストの長さは，空でない対象の要素の最大のインデックスとして定義する．

$$\mathrm{length} = (L) = \text{is-list})\,L) \rightarrow (L = <\ > \rightarrow 0,$$
$$T \rightarrow (\iota i)\,(\mathrm{elem}(i, L) \neq \Omega\,\&\,\mathrm{elem}(i+1, L) = \Omega))$$

(iv) リスト要素の取り出し

リストの先頭（もしあれば）の要素

$$\mathrm{head}(L) = \text{is-list}(L)\,\&\,(L \neq <\ >) \rightarrow \mathrm{elem}(1, L)$$

リストの最後の要素（もしあれば）

$$\mathrm{last}(L) = \text{is-list}(L)\,\&\,(L \neq <\ > \rightarrow \mathrm{elem}(\mathrm{length}(L), L)$$

リストの尾（先頭の要素を除いたもの）

$$\mathrm{tail}(L) = \text{is-list}(L)\,\&\,(L \neq <\ > \rightarrow (\mathrm{length}(L) = 1 \rightarrow <\ >, \mathrm{length}(L) > 1 \rightarrow \mu_\circ(\{\mathrm{elem}(i) : \mathrm{elem})\,i+1, L) > | 1 \leqq i \leqq (\mathrm{length}(L) - 1)\}))$$

(v) 2つのリストの連結

$$L_1 L_2 = \text{is-list}(L_1)\,\&\,\text{is-list}(L_2) \rightarrow \mu(L_1 ; \{<\mathrm{elem}(\mathrm{length}(L_1) + i) : \mathrm{elem}(i, L_2) > | 1 \leqq i \leqq \mathrm{length}(L_2))\})$$

多重連結はつぎのように定義する．

$$\mathrm{CONC}\ L_i = L_1 \frown L_2 \frown \cdots L_n$$

(vi) $<A_1, A_2, \cdots, A_n> \underset{Df}{=} \mu_0(<\mathrm{elem}(1) : A_1>, <\mathrm{elem}(2) : A_2>, \cdots, <\mathrm{elem}(n) : A_n>)$

ただし，

$n \geqq 1$，$A_i \neq \Omega\,(1 \leqq i \leqq n)$

である．

(vii) $\overset{n}{\underset{i=1}{\mathrm{LIST}}}\,A_i \underset{Df}{=} <A_1, A_2, \cdots, A_n>$

参考文

1) Walk, K. et al.: Abstract syntax and interpretation of PL/I.—IBM Lab. Vienna, Techn. Report TR 25.082, 28, June 1968.
2) Lucas, P. et al.: Informal introduction to the abstract syntax and interpretation of PL/I. IBM Lab. Vienna, Techn. Report TR 25.083, 28, June 1968.
3) Lucas, P. et al.: Method and notation for the formal definition of programming languages. IBM Lab. Vienna, Techn. Report TR 25.087, 28, June 1968.
4) McCarthy, J.: Towards a mathematical science of computation. — Information Processing 1962, pp. 21-28; Amsterdam 1963.
5) Landin, P. J.: Correspondence between ALGOL 60 and Church's Lambda-Notation. Part I.—C. ACM 8(1965), No. 2, pp. 89-101, Part II. C. ACM 8(1965), No. 3, pp. 158-165.
6) Bandat, K.: On the formal definition of PL/I. SJCC 1968, pp. 363-373.
7) Lauer, P.: Abstract syntax and interpretation of ALGON 60. —IBM Lab. Viema, Lab. Report LR 25.6.001, 12, April 1968.

(昭和42年2月3日受付) (つづく)