会員の声

作用的でない関数型言語について

小長谷明彦（日本電気）

情報処理 Vol. 22, No. 6 の「特集 プログラミング言語の最近の動向」[1]はなかなか読み応えのあるものであった．プログラミング言語に関しては数年に一度くらい，このようなサーベイを特集して欲しいと思う．

現在の私の興味は LISP や Prolog といった関数型言語あるいは関係型言語にあり，横井氏，渕氏らの解説は大変に参考になった．特に，横井氏の解説は関数型言語のあり方について多くの示唆を与えてくれるものと期待する．しかしながら，関数的 (functional) と作用的 (applicative) を同一視し，全ての関数型言語の基礎をラムダ計算に置いたために単一割り当て型言語[6)~8)]，データ・フローグラフ言語[5),9)]の関数型言語での位置づけが不明確な気がしてならない．以下，データ・フロー言語を含めた関数型言語と作用的言語に対する私見を述べさせてもらう．

関数的 (functional) というのはプログラミング・スタイルの名前で『手順を処理要素（式）間のデータの入出力依存性で表現するものである』という横井氏の定義には賛成である．

次に作用的 (applicative) であるが，この言語は命令的 (imperative) という言葉に対比させて使用することが多い．

Henderson の "Functional Programming"[2)]では副作用を持たない pure な関数型言語を作用的言語 (applicative language) と呼んでいる．しかしながら，Henderson の定義にはあまり賛成できない．なぜなら，完全に副作用を除去した言語は並列性の抽出が容易というメリットはあっても実用性が少ないからである．

私は，作用的 (applicative) というのもプログラミングスタイルの名前で，関数的 (functional) のサブセットではないかと考える．作用的言語は下記のプログラミングスタイルからなる言語と定義できる．

（1） 関数の適用 (application)　$f(x)$, $f(g(x))$

（2） 条件式　if $p(x)$ then $f(x)$ else $g(x)$

（3） 関数の再帰呼び出し　def f=——f——

以上が基本構成で，さらに下記のようなより命令的 (imperative) な制御構造への拡張が可能である．

（4） Landin の where 記法[3)]（ブロック文）

例）$(x+1)*(x-1)$

where $x=2$

または let $x=2$;

$(x+1)*(x-1)$

以上は $(\lambda x.(x+1)*(x-1))$（2）とラムダ式の application に変換することができる．

（5） Landin の jump 記法[4)] (GOTO 文, EXIT文)

ラムダ式の中で例外処理を表現する．

$(\lambda L.\cdots L(x)\cdots L(y)\cdots)(Jf)$

Lが評価された時点で，ブロックの外側に抜け出し，$f(x)$ あるいは $f(y)$ を値とする．

例）

def $f(x,y)=(\mathrm{Sq}(L(x,2))+y$

where $L=J(\lambda(x,y).x^2-y^2))$

とすると

$f(3,4)$
$=\mathrm{Sq}(L(3,2))+4$
$=L(3,2)$　　{ブロックから抜け出す}
$=3^2-2^2$
$=5$

となる．

（6） Id のループ表現[5)] (while 文)

例）

```
(initial i←1;
        sum←0
 while i<n do
    new i←i+1;
    new sum←sum+f(i)
 return sum)
```

次に，作用的でない関数型言語とは何かというと，単一割り当て型言語[6)~8)]やデータ・フローグラフ言語[5),9)]がそうではないかと思う．

図-1 に示すように単一割り当て言語の各代入文は命令的であり，関数の適用 (application) という概念はない．しかしながら，代入文を評価する手順はオペランド間のデータの入出力依存性で表現されており，

① $C=A+B$
② $A=2$
③ $E=C\times D$
④ $B=1$
⑤ $DB-A$

図-1 単一割り当て言語のプログラム

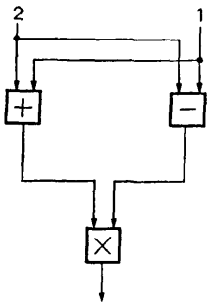

図-2 データ・フローグラフによるプログラム

各代入文はプログラム上の位置とは無関係にオペランドの揃った代入文②と④，①と⑤，③の順（すなわちデータ依存関係）で実行できる．

図-2 のデータ・フローグラフの場合も同様で，各ノードを評価する手順はトークンの入出力依存性（アーク）で表現されている．

従来，関数型言語といえば LISP のような関数の適用を中心とした作用的言語を意味していたが，単一割り当て型言語，データ・フローグラフ言語の出現は関数型言語の新たなプログラミングスタイルを作り出したと考えても良いのではないだろうか．

参 考 文 献

1) 特集 プログラミング言語の最近の動向，情報処理，Vol. 22, No. 6 (1981)
2) Henderson, P.: Functional Programmig Application and Implementation, Prentice-Hall International Series in Computer Science, London (1980).
3) Landin, P. J.: The Next 700 Programming Languages, CACM, Vol. 9, No. 3, pp. 157-166 (Mar. 1966).
4) Landin, P. J.: An Abstract Machine for Designers of Computing Languages, IFIPS 65, pp. 438-439.
5) Arvind, Gostelow, K. P., Plouffe, W.: An Asynchronous Programming Language and Computing Machine, TR 114a, Dept. of Information and Computer Science, University of California, Irvine, California (Dec. 1978).
6) Tesler, L. G. and Enea, H. J.: A Language Design for Concurrent Processes, AFIPS Conf. SJCC, Vol. 32, pp. 403-408 (1968).
7) Chamberlin, D. D.: The "Single-assignment" Approach to Parallel Processing, AFIPS Conf. NCC, Vol. 39, pp. 263-269 (1979).
8) Comte, D., Durrieu, G., Gelly, O., Plas, A. and Syre, J. C.: "Parallelism, Control and Synchronization Expression in a Single Assignment Language," SIGPLAN Notice, Vol. 13, No. 1, pp. 25-33 (Jan. 1978).
9) Dennis, J. B.: First Version of a Data Flow Procedure Language, Lecture Note in Computer Science, Vol. 19, Springer-Verlag, pp. 362-376 (1974).

(昭和 56 年 8 月 24 日受付)

小長谷氏のコメントに対するコメント

横井俊夫（電子技術総合研究所）

小長谷氏のコメントの骨子は，単一代入（割当）規則等による記述が作用的でないことを理由に，関数型言語を大きく，作用的言語とデータ・フロー言語にわけるべきであるというものである．一方，筆者の解説 (81 年 6 月号"関数型言語") は，λ-計算を出発点に，様々の拡張や修飾を施すことにより，有用な言語要素の多くが，関数型言語という枠組の中に包含されるというのが骨子である．単一代入規則も，そのような拡張の 1 つと見なすというもので，陽には述べられてはいないが，データ・フロー言語という言語カテゴリは，あえて設けるほどのものではないというものである．

いずれにしろ，単一代入規則をどう見立てるかが意見の別れめである．λ-計算の拡張の 1 つと見立てた方が妥当だという理由の大筋を以下に述べる．

代入という行為は，(変数) 名のついた記憶領域にある値を書き込むということと，それ以後は，その名前で，値を参照できるようにするという 2 つのことからなる．単一代入規則は，後の機能のみを利用するというものである．この副作用なしに名前付けを行うという仕組は，ラムダ変数への結合ということによって実現できる．

以上は，解説でふれたことである．たとえば，小長谷氏の例は，次のようなラムダ式で表される．