記 号 処 理 54－3
（1990．3．12）

LISPプログラムの図式表現VEX

大島義光
(株)日立製作所 中央研究所

Common Lispプログラムの図式表現法 -- VEX -- について述べる。本表現の特徴は、(1) 関数を入出力付きの箱で表し、関数間の関係、データの流れの理解を容易にしたこと、(2) ブロック構造および反復の制御構造を枠で表現し、プログラムのマクロ的な構造を分かりやすくしたこと、(3) 複雑な関数の機能を比喩的な図式で表し、視覚的な諒解度を向上させたこと、である。これによって、従来カッコが多く見にくいと言われていたLispプログラムの欠点が改善され、Lispによる高度なプログラムが容易に理解できるようになった。本稿では、VEX開発の背景、仕様の概要、およびVEXを利用したプログラム開発環境について述べる。

VEX -- A Visual Expression for Common Lisp

Yoshimitsu Oshima
Central Research Laboratory, Hitachi Ltd.
1-280, Higashi-Koigakubo, Kokubunji, Tokyo 185, Japan

A visual expression for Common Lisp programs -- VEX -- is presented. VEX has the following features. (1) Functions are represented by boxes with inputs and outputs, which assists in understanding relationships between functions and data flows through them. (2) Block structures and iteration constructs are expressed by frames containing their bodies. Such expressions assists in the understanding of program macro structures. (3) The functions that have complicated processing mechanisms are expressed by metaphorical figures. By means of these figures processing details can be clearly understood.

In this paper, VEX background, its brief specifications and some topics on the programming environment by VEX are described.

1. はじめに

Lispは、機能が高く優れた言語であり、複雑なデータ構造の記述や、複雑なアルゴリズムのプログラムの記述に適している。しかし、その構文はカッコを基本としたある意味で非常に単純な形をしており、ソース・プログラム中でカッコが重なると極めて見にくいという欠点がある(Lispとは、Lots of Insipid Stupid Parentheses((味気なくバカバカしいたくさんのカッコ))のことであるなどと悪口を言われている[1])。

最近この問題を解決するために、Lispの図式表現法がいくつか提案されている[2][3][4]。しかし、今日のLisp(特にCommon Lisp[5])は単なる関数型言語ではなく、多くの手続き型言語の要素を持っている。したがって単純に関数表現を図式化すれば複雑なLispのプログラムが分かりやすくなるという訳ではない。

本稿では、このような問題を解決することを目指して開発したLispプログラムの図式表現法VEX*について述べる。

以下、最初に、文字表現プログラムおよび既存の図式表現法の問題点について検討し、次に、この検討を通じて設定したLispプログラムの新しい図式表現法VEXの仕様について述べ、最後にVEXによるプログラム開発支援環境のいくつかの特長について述べる。

注) VEXは、Visual Expressionの略。S-式(symbolic expression)の類推からこう名付けた。

2. Lispの図式表現法の検討

2.1 文字表現プログラムの問題点

一般に文字表現によるプログラムには次のような問題がある。

(1) 視覚に訴える力が弱い。インデント表示など出力方法に工夫を加えたとしても[6]、プログラムの内容を理解するのに骨が折れる。特に大きなプログラムでは、その構造を理解するのが大変である。

さらにLisp特有の問題として、以下の諸点を上げることが出来る。

(2) Lispの構文はカッコを基本としたある意味で非常に単純な形をしており、ソース・プログラム中でカッコが重なると極めて見にくい。

(3) Lispのすべての要素は単純な前置記法による構文で記述されるので、大きなプログラムの理解がさらに困難である。特に、多重の関数呼び出しからなるプログラムでは、各レベルの関数のパラメタの対応が取りにくい。

(4) 同様の理由で、数式などは理解しにくい。

(5) Lisp(特にCommon Lisp)には、複雑な機能を持った関数等の言語要素が多い。したがって、たまに使う関数などは細かい仕様を忘れてしまっていることが多い。

2.2 既存の図式表現法の問題点

Common Lispは関数型言語であるが、手続き型プログラミング用の記述要素も多く持っている。そこで、既存の手続き型言語用の図式表現、および関数型言語用の図式表現双方について、その問題点を検討してみる。

2.2.1 手続き型言語のための図式表現

手続き型言語によるプログラムの図式化法についてはこれまで多くの提案がなされている[7][8]。例えばフローチャート、NSチャート、PADなどがある[9]。しかし、これらは手続き型言語の三つの制御構造——連接、選択、反復——を視覚化することに主眼があり、表現可能な範囲が限定されている。これ以外の要素は従来通り文字によるプログラムで表される。したがって、上記三者以外の部分の多いプログラムについては十分機能を果たすことができない。

特に次の点が問題である。

(1) ブロック構造が表現できない。

(2) 関数表現がない。

2.2.2 関数型言語の図式表現

最近、Lispなどの関数型言語を対象とするプログラムの図式表現法がいくつか提案されている[2]-[4],[10]-[12]。

これらの表現方法は関数表現には適しているが、最近の機能の豊富なLisp言語によるプログラムの図式表現のためには不十分である。

2.3 プログラムの表現法の要件

以上の問題分析に基づき、次にプログラム表現法として求められる一般的要件について考えてみたい。

まず、プログラムを理解する観点から見てみる。我々が日常プログラムを理解していく過程を省察してみると、次の二つの視点でプログラムを見ていることがわかる。

(1) トップダウン的視点
→ 構造の理解：プログラムがどのような要素からなっているか理解する。
(2) ボトムアップ的視点
→ 処理手順の理解：単位機能とその組合せ、およびデータの流れを積上げ的に理解する。

通常は、この両者を併用してプログラムの理解を進めていると言えよう。

一方、プログラムの設計時においても、上記両者の視点を行きつ戻りつ思考を進めて、最終的に完全なプログラムを作成していると思われる。

したがって、図式表現を設計するにあたっても、この両者について配慮した表現法を実現することが望まれる。

2.4 VEXの狙い

以上の検討を踏まえ、Lisp言語(特にCommon Lisp)に適した図式表現法の実現をはかるために、以下の狙いと設計条件を設定した。

狙いとして、

(1) 関数の図式表現の実現。
(2) ブロック構造などプログラムの構造の理解容易化。
(3) 関数などの言語要素の機能の可視化。
(4) 数式の理解容易化。

また、設計条件として、

(5) ワークステーションなどの電子機器上での実現を前提とした理解容易性の追求。
(6) 文字表現によるプログラムからの移行、および文字表現のプログラムと図式表現のプログラムとの対応が容易なこと。
(7) 文字表現のプログラムに比べて余分に必要とする表示スペースが少ないこと。

3. Lispプログラムの図式表現VEX

本章では、VEXの考え方、および仕様の概略について述べる。

3.1 VEXの設計方針

前章で述べた狙いに基づき具体的なLispの図式表現を設計するに当たり、Common Lispの言語仕様を考慮して、さらに次のような方針を設定した。

(1) 関数は基本的な表現を一つ定め、通常はそれを用いる。

関数は呼び出す側から見ると基本的に同じである。ただし、後述のmap関数のように、複雑な関数の機能を独自の形で表した場合がよい場合もあるので、固有の表現を設定することも可とした。

(2) Common Lispであらかじめで定められた特別なプログラミング要素に対しては、個別の図式表現を定める。

Common Lispは、プログラム構成するために、多くの特別な記述要素を用意している。それらは、制御フローの記述要素、関数などの定義の形式、引用など評価の制御のための形式、変数への代入の形式、局所変数を導入するためのもの、などである。これらにはCommon Lispの特殊形式またはマクロが用いられており、それぞれ固有の構文と意味を持っている。これらを適切な図式によって表現すれば、プログラムの理解に大きく資することができる。

3.2 VEXの具体仕様

本節ではVEXの仕様について述べる。ところで、Common Lispの仕様は非常に大きい。したがって、ここでVEXの仕様全てについて述べることは出来ない。基本的な説明を与えるにとどめる。

(1) 関数呼び出し

(i) 基本表現

関数の基本表現を図3.1に示す。関数本体を箱で表し、引数(パラメタ)を右側から入る矢印、戻り値を左から出ていく矢印で表す。

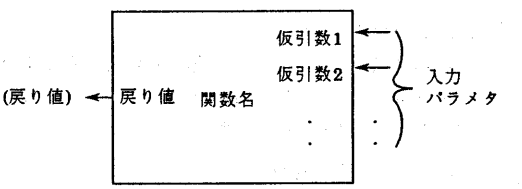


図3.1 関数の基本表現

仮引数、戻り値の名称は省略してもよい。

なお、多値関数は、戻り値を表している矢印を追加することによって表現可能である。

(ii) 各種パラメタの記法

Common Lispでは、必須パラメタの外に、オプショナル(&optional)、レスト(&rest)、キーワード(&key)などの多くの種類のパラメタが用意されている。これらを、図3.2に示すように、引数の矢印にマークを付加することによって表現する。なお

誤解の恐れがないときは、これらのマークは適宜省略することも出来る。

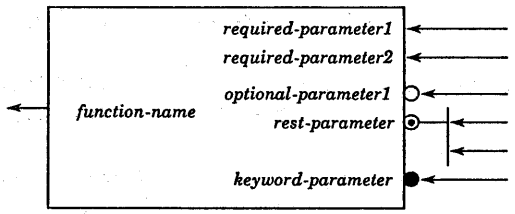


図3.2　各種パラメタの記法

(iii) 関数引数

関数を引数として取る関数では、引数で与えられた関数が実際の仕事をすることが多い。これを分かりやすく表すために、関数引数をもとの関数の箱の中に入れて表す。図3.3にその例を示す(mapcarの例)。

本図ではさらに、処理の対象となるリストが分解され、要素の一つ一つが引数の関数に与えられ、結果がまたリストに組み立てられて返される様子を、前後に付けた三角形で比喩的に表現している。

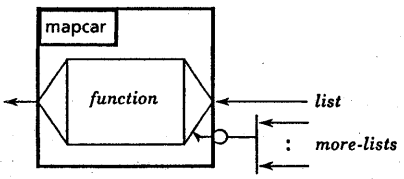


図3.4　関数mapcarの図式表現

(iii) 再帰呼び出し

自己再帰関数、すなわち定義の中で自分自身を読んでいる関数は、それを強調する意味で、その呼び出し部分を太枠の箱で表現する。

(2) ブロック構造と反復

Common Lispはいくつかのブロック構造の記述要素を用意している。それらは、関数等の定義の記述要素(defun、defmacro、等)、局所変数設定の記述要素(let、flet、等)、静的な脱出機構の記述要素(Block、return-from)、プログ(prog)形式などの記述要素がある。これらを、その部分プログラムを内に含む枠で共通的に表現する。図3.5、図3.6に例を示す。後述のプログラム例から分かるように、枠で囲むことによってまとまりの範囲が明確となり、プログラムの構造が理解しやすくなる。

一方、doなどの反復の記述要素は、あるまとまった処理単位を表現すること、制御変数を持つ

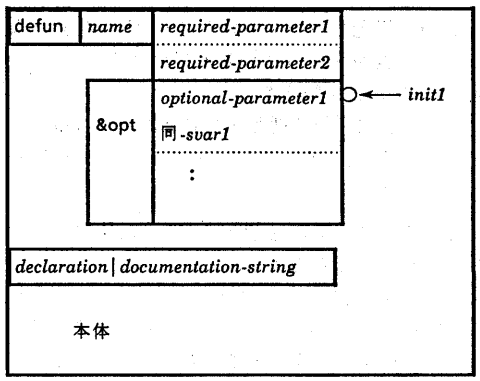


図3.5　defunの図式表現

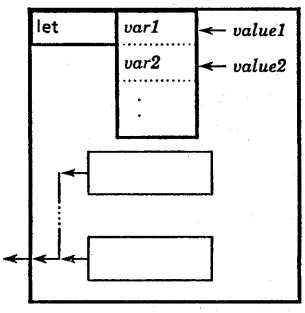


図3.6　letの図式表現

ことなどから、機能的にブロック構造とかなり共通の要素を持っている。そこで、反復についてもブロック構造と似た枠による表現を採用することにした。枠による反復の図式表現の例として、doの図式表現を図3.7に示す。

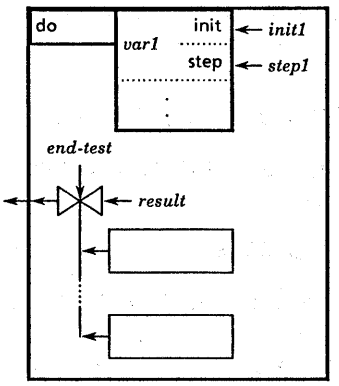


図3.7　doの図式表現

図3.7で*end-test*と*result*の矢印が指しているバルブ表現(頂点を共有する二つの三角形)は、後述の選択の表現を流用したもので、doの終了判定を表している。

(3) その他の制御構造の記述要素

(i) 連接

Common Lispには、連接を記述する要素として、progn、prog1、prog2の三つがある。それぞれ処理の結果として返す値が異なる。そこで、処理結果を示す矢印を、対応する要素の位置に付けることにより、その違いを表現する(図3.8)。

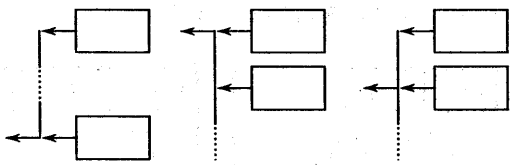


図3.8　連接の図式表現

(ii) 選択

選択の基本表現を図3.9(a)に示す。これはバルブを比喩的に表現したものである。選択の条件部がバルブを通る流れを制御しているというニュアンスを表している。これを用いたif、condの図式表現を、図3.9(b)-(c)に示す。

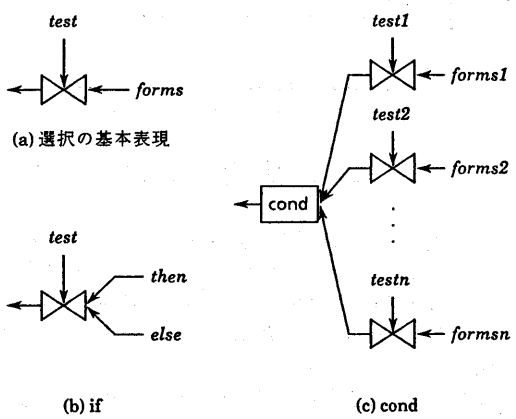


図3.9　各種選択の表現

(4) 変数と代入

(i) 変数

変数は、局所変数、大域変数ともS-式のまま、すなわちシンボルのままで表す。

(ii) 代入

setqによる代入は、図3.10に示すように、代入先の変数をsetqの箱の中に入れて表示し、代入する値をそれに流入する形で表現する。

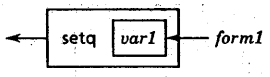


図3.10　setqの図式表現

(iii) 汎変数

Common Lispでは、値を格納可能な任意の記憶場所に直接値を代入する機能がある。この、値の格納可能な記憶場所のことを汎変数(generalized valiable)という。

汎変数への代入(setf)の図式表現は、上記の変数への代入表現を拡張し、変数の位置に記憶場所の参照表現を置いて表す。

(5) 引用

(i) 単純引用

Lispにおいて、主として引用(quote)の対象となるのはシンボルまたはリストによるS式であるが、その内容は単なるデータの場合とプログラムの場合とがある。

(a) データ

quoteまたは引用符(')付きで、文字表現のまま表わす。

(b) プログラム

図3.11に示すように、引用(quote)されている部分を二重線による枠で囲んで表す。これによって、評価(実行)を遮断しているというニュアンスを表現している。

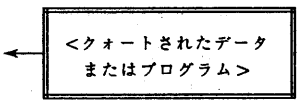


図3.11　quoteの図式表現

どちらの表現を採用すべきかは、その表現が現れた文脈(context)に依存する。

(ii) 関数引用

関数引用(function)は、三重線による枠で表現する。

(iii) バッククォート(backquote)

バッククォートは、基本的にはquoteと同じであるが、その中に再評価のための覗き穴(peep hole)を設けることができる。この状況を表すためにバッククォートされた部分全体にシェード(影)を施し、再評価する部分を白ヌキの穴によって表現する。例を図3.12に示す。

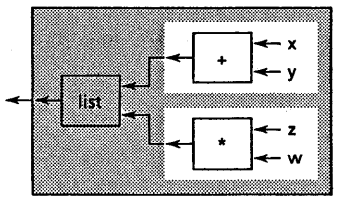


図3.12　バッククォートの図式表現の例

(6) 数式

数式の要素——加算、減算、乗算、割り算、平方根、べき乗、絶対値、各種の数の比較など——は、全て数学の教科書にあるような数式の形で表す。

(7) オブジェクト指向

Common Lispは、固有のオブジェクト指向機能——CLOS(Common Lisp Object System)[13]——を用意している。しかし、その構文はほぼLispの構文に則っており、CLOSの各言語要素の図式表現はこれまで述べた各種表現を敷延して設定することができる。

3.3 VEXによるプログラムの記述

VEXによるプログラムの記述例を図3.13に示す。

この例および数種のプログラムの記述結果から、VEXは以下の点で有効であることがわかった。

(1) 枠によるブロック構造の表現と関数表現の組合せによって、プログラムの構造がより明確に理解できるようになった。
(2) 関数の図式表現によって、いくつもの関数を経由するデータの流れが明確になった。関数を一つ一つ追うことによって、プログラムの機能を下から上へと段階的に理解していくことが出来る。
(3) mapcarなどの処理が少々複雑な関数も、その機能を視覚的に表現することによって、その処理内容が理解しやすくなった。

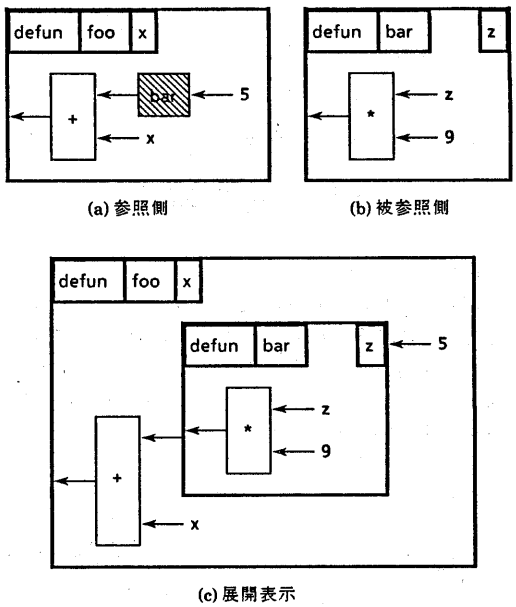
4. VEXによるプログラム開発環境

VEXによるプログラム開発環境を現在開発中である。本章ではそのいくつかの特長について述べる。

文字表現によるプログラム開発環境と同様に、VEXによるプログラム開発環境は、ブラウザ、エディタ、デバッガ、文書出力などのツールから構成される。

(1)　ブラウザ

ブラウザは、入力済のプログラムの関連情報の検索と表示を行うツールである。VEXのブラウザは、ブラウザとしての一般的な機能に加え、図式表示の特長を生かしたプログラムの展開表示機能を持っている。表示例を図4.1に示す。



(a) 参照側　(b) 被参照側

(c) 展開表示

図4.1　展開機能の表示例

ユーザが図の(a)に示したプログラムを見ているとき、関数barの中身を見たいと思ったとしよう(これは図の(b)に示されている)。この時、両者を合わせた複合的な図式表現(図の(c))の形式でプログラムを表示することができる。これによって、プログラムの呼出し側と呼ばれた側を一体として見ることができる。

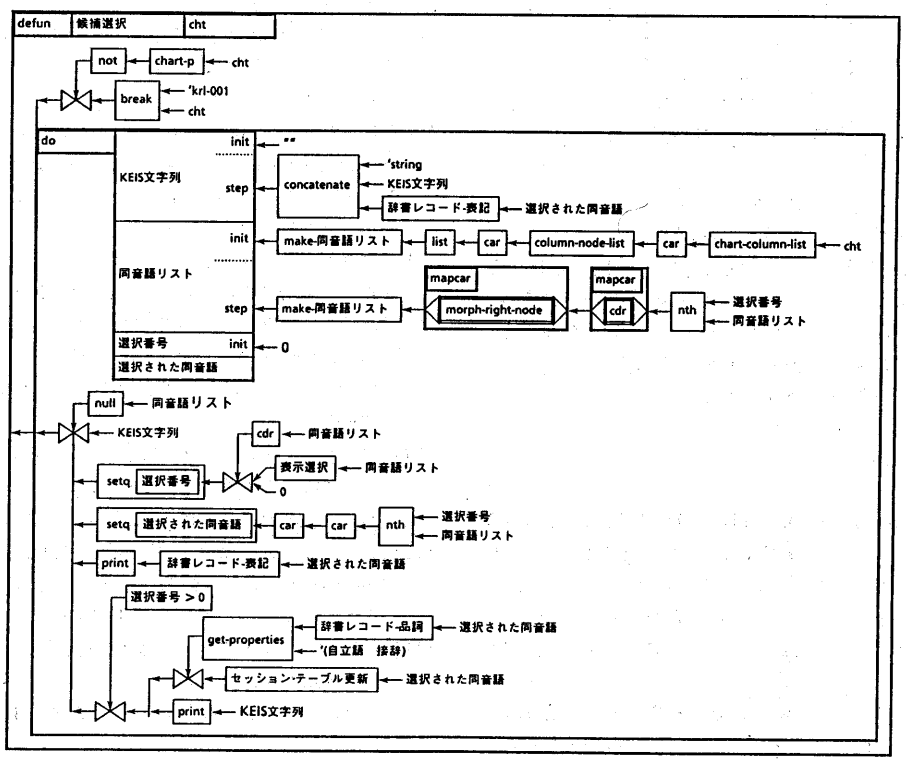
(2) エディタ

VEXのエディタは、二種類の入力編集機能を備えている。

(i) 文字で入力してそれを図式表現に変換する機能
(ii) 図形ベースの入力編集機能

```
(defun  候補選択  (cht)
  (if  (not  (chart-p  cht))
       (break  'krl001  cht))
  (do  ((KEIS文字列
         ""
         (concatenate  'string
           KEIS文字列
           (辞書レコード-表記  選択された同音語)))
        (同音語リスト
         (make-同音語リスト
           (list
             (car  (column-node-list
                     (car  (chart-column-list  cht))))))
         (make-同音語リスト
           (mapcar
             #'morph-right-node
             (mapcar  #'cdr  (nth  選択番号  同音語リスト)))))
        )
       (選択番号  0)
       (選択された同音語)
       )
      ((null  同音語リスト)  KEIS文字列)
  (setq  選択番号
         (if  (cdr  同音語リスト)
              (表示選択  同音語リスト)
              0))
  (setq  選択された同音語
         (car  (car  (nth  選択番号  同音語リスト))))
  (print  (辞書レコード-表記  選択された同音語))
  (if  (>  選択番号  0)
       (progn
         (if  (get-properties  (辞書レコード-品詞  選択された同音語)
                               '(自立語  接辞))
              (セッション-テーブル更新  選択された同音語))
         (print  KEIS文字列)))
  )
)
```

(a) ソースプログラム



(b) VEXによる記述

図3.13　VEXによるプログラムの記述例

(i)は初期入力に適しており、(ii)は既作成のプログラムの修正作業に適している。

(3) デバッガ

VEXのデバッガは図式表示の特長を生かした幾つかの動作表示(アニメーション)機能を備えている。

(i) VEXによる図式表現プログラム上でのトレース表示

実行中のプログラムに、その現在位置と関連データを動的に重畳表示する。

(ii) map関数などの複雑な処理機構を有する関数の詳細トレース表示

例を図4.2に示す。入力リストと出力リスト、および処理中の入力データとその処理結果が各リストの対応位置に示されている。これによって、複雑な処理機構を有するシステム関数の動作がよくわかる。

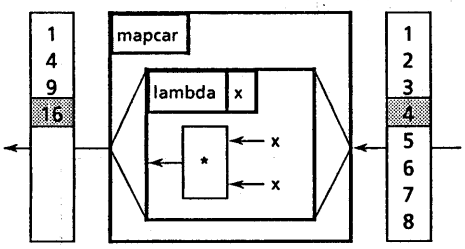


図4.2 関数mapcarの詳細動作表示の例

(iii) 関数・マクロの動的展開表示

ブラウザの項で説明した展開表示を、プログラムの実行時に動的に行う機能である。関数呼び出しがあったとき、その呼び出し位置にその内容を展開して表示する。パラメタやリターン値の受渡しが視覚的に容易に把握できるようになるので、プログラムの動作の理解に果たす効果が大きい。

5. 終わりに

Common Lispの図式表現VEXを提案した。3.3節で述べたように、VEXによってLispのプログラムをより明確かつ容易に理解できるようになった。VEXは、また、プログラムの理解だけでなく、大きなプログラムの設計やデバッグのためにも有用である。

VEXによるプログラム開発支援環境は現在開発中である。開発支援環境の詳細およびVEXの試用評価については別の機会に報告したい。

ちなみに、今がCommon Lispの図式表現の標準化について討論するいい機会ではないかと思う。本稿がその一助になれば幸いである。

6. 参考文献

[1] 中西正和 : LISP, 近代科学社

[2] Edel, M. : The Tinkertoy Graphical Programming Environment, IEEE Trans. on Software Engineering, Vol.14, No.8, pp.1110-1115 (Aug. 1988)

[3] Levien, R. : Visual Programming : Byte, February, 1986, pp.135-144

[4] Lakin, F. : Spacial Parsing For Visual Languages, in Visual Lnguages, edited by S. K. Chang, T. Ichikawa and P. A. Ligomenides, Plenum Publishing Corp. (1986)

[5] G. L. Steele Jr., Common LISP:The Language, Digital Press, 1984

[6] Baecker, R., and Marcus, A., : Design Principles for the Enhanced Presentation of Computer Program Source Text, Proceedings of CHI"86, Human Factors in Computing Systems (Apr. 1986), pp.51-58

[7] プログラム制御構造の新しい表現法:木構造チャートが普及期に, 日経コンピュータ, 1989.1.9, pp.71-83

[8] Tripp, L. L. : A Survey of Graphical Notations for Program Design — An Update : ACM SIGSOFT Software Engineering Notes, Vol.13, No.4, pp.39-44 (Oct. 1988)

[9] 二村良彦, 川合敏雄 : プログラムの木構造化図面 "PAD", 日立評論, Vol.62, No.12 (1980-12)

[10] 飯島正, 岡田謙一, 横山光男, 北川節 : T式による関数型プログラム開発, 情報処理学会第33回全国大会講演論文集, 7G-8, pp.789-790 (1986)

[11] 布川博士, 富樫敦, 野口正一 : 図式をシンタックスに持つ関数型言語, コンピュータソフトウェア, Vol.6, No.2, pp.11-23 (1989)

[12] 二村良彦, 野木兼六, 高野明彦 : LAD(Lambda Diagrams) : ラムダ式の図形表現、第30回プログラミング・シンポジウム 1990.1

[13] 井田昌之, 元吉文男, 大久保清貴 編 : Common Lisp オブジェクトシステム—CLOSとその周辺—, bit別冊, 共立出版, 1989