プログラミング言語　8－3
（1986. 10. 17）

# 汎用計算機上に実現されたＬＩＳＰプログラム開発支援システム

太田　義勝†　　吉田　雄二‡　　福村　晃夫‡
（†　名古屋大学　大型計算機センター）（‡　名古屋大学　工学部）

汎用計算機の上に実現されたＬＩＳＰ処理系Ｕｔｉｌｉｓｐを対象に，ＬＩＳＰプログラムを開発する際にプログラマーを支援するために開発されたいくつかのシステムについて報告する。プログラム開発支援システムは，プログラムの構造解析ツール（ＡＮＡＬＹＺＥＲ），プログラムの動的解析ツール（ＬＩＳＰＤＡＰ），知的エディタ（ＩＥＤＩＴ）などから構成される。ＡＮＡＬＹＺＥＲはプログラムの構文チェック，プログラムの静的構造の表示を行う。ＬＩＳＰＤＡＰはプログラムの実行プロフィールを表示する。ＩＥＤＩＴは自然言語インターフェイスを備えた構造エディタである。

# PROGRAM DEVELOPMENT SYSTEM FOR LISP ON MAINFRAME COMPUTER

Yoshikatsu OHTA , Yuuji YOSHIDA and Teruo FUKUMURA
Nagoya University
Furo-cho, Chikusa, Nagoya, Aichi, 464, Japan

We describe program development system for LISP. Utilisp is implemented on many mainframe computers. This system assists programmar to develop Utilisp program. This system consist of program structure analyzer(ANALYZER), dynamic program analyzer(LISPDAP), intelligent editor(IEDIT), etc. ANALYZER checks program's syntax and displays program's static structure. LISPDAP displays program's execution profile. IEDIT is structure-editor with natural language interface.

まえがき

Lispは人工知能,知識工学の分野でシステムを作成するのによく用いられるプログラミング言語である。Lispのためのプログラミング支援システムとしてはInterLisp [1] のDWIM(Do What I Mean)が有名であり,また最近のワークステーション上のLispシステムではビットマップ・ディスプレイ,マウスを利用してユーザ・インターフェイスをよくしたシステムがある。

筆者らは,汎用大型計算機上で使われているUtiLisp [2] に対するプログラミング開発支援のためのシステムをいくつか開発してきた。本稿では我々が開発してきたLISPプログラミング開発支援システムのうち,静的解析システム(ANALYZER),動的解析システム(LISPDAP),知的エディタ(IEDIT)について概略を紹介する。

1. 静的解析システム(ANALYZER) [3]

プログラム開発のためには,プログラムの構造を解析するシステムが不可欠である。そのようなシステムとして,InterLispには,LISPプログラムを解析して関数間の呼び出し関係を図示したり変数情報を表示するPRINTSTRUCTURE,PRINTSTRUCTUREに自然言語風のインターフェイスをつけたINTERSCOPEと呼ばれるシステムがある。

筆者らは,PRINTSTRUCTURE/INTERSCOPEをモデルに,UtiLisp上にLISPプログラム静的解析システム(Analyzer)を作成した。 Analyzerは,プログラムを解析して解析結果のデータベースをつくる「解析部」とそのデータベースから解析結果を出力する「出力部」の2つから構成されている(図1)。

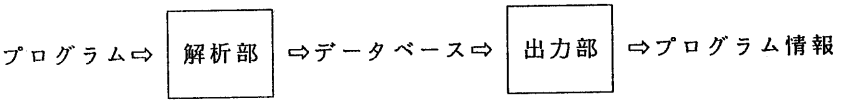



図1. Analyzerの構成

解析部は以下の処理を行う。

1) 構文チェック

解析時に,「呼び出される関数が定義されているか」,「引数の個数はあっているか」,「ラムダ/プログ変数がシンボルかどうか」,「関数GOがPROGの中で呼ばれているか」,「ラベルが定義されているか」,などをチェックする。これにより自明な誤りを1回の解析で検出することができる。

2) 関数の呼び出し情報の登録

関数から直接呼び出されるユーザ関数を登録する。直接呼び出される関数とは,定義中に現われる関数呼び出し,または,関数の引用(関数FUNCTION)である。マクロは展開してから調べる。

3）束縛変数，自由変数の登録

関数で束縛されている変数，および，自由変数を登録する。UtiLispでは変数の束縛は，LAMBDA式，PROG式，および，関数MATCHで行われる。

解析結果は関数名，変数名の属性リストと，大局変数に作り出される。

出力部は，次のような出力を行う。

1）関数呼び出しの樹状表示

ある関数から始まる関数呼び出しの樹状出力を行う(print-tree)。グラフィック端末に関数の呼び出しの樹状表示を行う(show-tree)。

UtiLispコンパイラの関数呼び出し関係の表示の例を次に示す。

```
> (PRINT-TREE 'COMPFILE)
COMPFILE C:LAMBDA POST-PROCESS EXPAND-MNEM
                               EXPAND-LAB
                               TRANS-REG ...
                               PREPROCESS
                 C:UNDO-UPTO
                 C:SEQ C:EVAL ...
                       C:EVAL-AGAIN ...
                 C:LOAD-A-REG C:LOAD ...
                 C:FORMALS C:BIND ...
                           C:SPECIALP
                           C:SEQ ...
                           C:LOAD ...
                           C:ERR
                           C:VARCHECK ...
                 C:ERR
        LOW-LEVEL-OPTIMIZE LAP:FINAL
                           LAP:SHORTCUT
                           LAP:ELIMINATE
        C:LAP LAP:PASS-ONE
              LAP:PASS-TWO
        PRINT-CODE PRINT-HALF
NIL
```

2）相互参照表の出力

ある関数を呼んでいる関数と自由変数を使用している関数の相互参照表の出力する。

3）束縛元関数，トップレベル関数の検出

変数を束縛している関数(束縛元関数)，他から呼ばれていない(自分自身の呼び出しは除く)ユーザ関数(トップレベル関数)を求める。これらは，「未束縛変数」「未使用関数」を見付ける役にたつ。束縛元関数，大局変数，トップレベル関数の決定は完全ではない。

## 2．LISPプログラムの動的解析システム(LISPDAP)［4］

プログラムの効率を改善するために，プログラムのもつ動的性質(例えば，ステートメントの実行回数，手続きの実行時間，条件の成立した回数など)に関するデータを収集するシステムは有用なものである。筆者らはLISPプログラムの動的性質を測定するシステムLISPDAPを作成した。

LISPDAPシステムは，LISPプログラムの動的性質として，

(1)関数の呼び出し回数，

(2)マッカーシーの条件式で各々の条件が成立した回数，

(3)PROG形式に含まれるラベルを通過した回数，

(4)関数の計算時間，

(5)関数のフリーセル消費量，

(6)再帰呼び出しのネストの深さ，

の測定を行う。

LISPDAPシステムは，図2に示すように，前処理，実行，後処理の3段階から構成される。

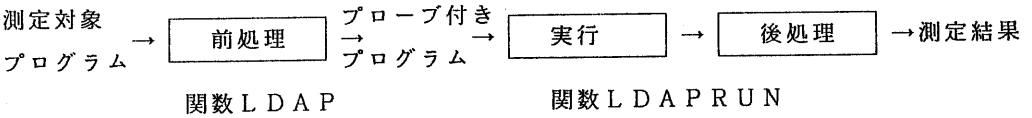



図2　ＬＩＳＰＤＡＰシステムの構成

（１）前処理

前処理部では，測定対象プログラムに，動的性質を測定するためのプローブを埋めこむ。

（２）実行

実行部では，測定項目を格納する大局変数等を初期化した後，前処理部で得られた動的性質用プローブ付きプログラムを実行する。実行に伴いこのプログラムの動的性質が測定されて，大局変数に格納される。

（３）後処理

後処理部では，実行部で得られた測定結果を編集して，出力する。測定結果は，見やすさのために，動的性質(1)，(2)，(3)については，pretty printされたプログラムとともに出力される。また，動的性質(7)については，関数の呼び出し関係を樹状に出力したものとともに出力される。

前処理部は，関数LDAPとして，また，実行部と後処理部は関数LDAPRUNとして実現されている。このほか，関数LDAPによって動的性質測定用に関数定義が変更された関数を，もとの形に戻す関数UNLDAPからLISPDAPシステムは構成されている。

前処理による変換は，次に示すように行われる。

```
(defun rev (x)
  (prog (y)
   loop (cond ((null x) (return y))))
        (setq y (cons (car x) y)
        (setq x (cdr x))
        (go loop)))
```

⇩

```
(defun rev (x)
  (setq g0012 (add1 g0012))    ..... 関数の実行回数の測定
  (prog (value)
    (setq rev#f (add1 rev#f))
    (cond ((zerop rev#f)
           (setq rev#1 (clock))
           (setq rev#2 (conscount)))    ..... 実行時間，セル消費量の測定
    (setq value
      (prog (y)
       loop (setq g0013 (add1 g0013))    ..... ラベルの通過回数の測定
         (cond ((null x)
                (setq g0014 (add1 g0014))    ..... 条件の成立回数の測定
                (return y))))
         (setq y (cons (car x) y)
         (setq x (cdr x))
         (go loop)))
    (cond ((zerop rev#f)
           (setq rev#t (plus rev#t (difference (clock) rev#1)))
           (setq rev#c (plus rev#c (difference (conscount) rev#2))))
                    ..... 実行時間，セル消費量の測定
          ((greaterp rev#f rev#s)
           (setq rev#s rev#f)))    ..... 再帰呼び出しのネストの深さの測定
    (setq rev#f (sub1 rev#f))
    (return value)))
```

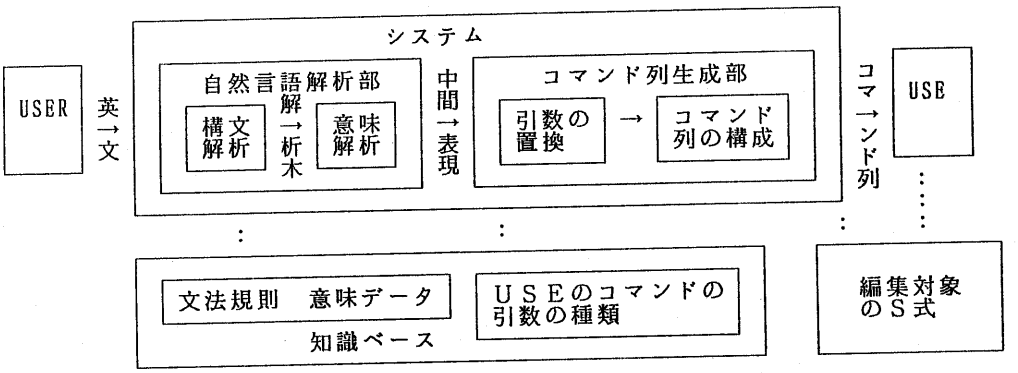
LISPDAPシステムによる解析結果の出力の例を次に示す。

```
> (ldap '(rev))
(rev)


> (ldaprun '(rev '(1 2 3 4 5 6 7 8 9 0)))
*** EXECUTION TIME 0 MSEC ***** USED CELL 10 ***    .....実行時間，セル消費量
rev---1    ..... 関数の実行回数の測定
  (lambda (x)
    (prog (y)
      loop---11    ..... ラベルの通過回数の測定
      (cond
        ((null x)---1    ..... 条件の成立回数の測定
         (return y)
        )
      )
      (setq y (cons (car x) y)
      (setq x (cdr x))
      (go loop)
    )
  )
DEPTH OF RECURSION **0**    ..... 再帰呼び出しのネストの深さの測定
(0 9 8 7 6 5 4 3 2 1)
>
```

## 3．知的エディタ(IEDIT)［5］

エディタはプログラム開発に必須のツールである。しかし，エディタごとにコマンド体系が異なり，ユーザーはエディタごとにコマンドを覚える必要がある。IEDITは既存のエディタに自然言語インターフェイスをつけたもので，ユーザーはエディタの操作法を容易に覚えることが可能となる。本システムは，既存のエディタとしてUTILISPの構造エディタUSEを用いている。ユーザーは英文で編集要求を入力する。システムはこれを解析し，ユーザーの要求に応じて，USEのコマンド列を生成し，USEに引き渡し実行させる。下図にシステムの構成を示す。




図．3　システム構成図

### (1) 自然言語解析部

自然言語解析部は，構文解析部と意味解析部から成る。ユーザーが入力した英文を解析し，構文解析木を作る。次に意味解析部が構文解析木から「意味」すなわち中間言語表現を作る。

自然言語解析部は，特定のLISPエディタには依存しない。中間言語表現は，LISPプログラムの編集の際に考えられる様々な要求を表現する。

### (2) コマンド列生成部

コマンド列生成部は，フレームを使って中間言語表現からコマンド列を導出する。USEのコマンドの機能・引数などに関する知識，中間言語表現からUSEのコマンド列を導く方法がフレームで与えられている。コマンド列生成部は，これらのフレームを起動することにより，ユーザーの要求に即したUSEのコマンド列を生成する。

フレームには，

(1)ユーザーの要求に対応するコマンド名(コマンド名の組)

(2)コマンドの引数の型(種類)

(3)中間表現で表わされた引数をUSEに適した形に変換する方法

(4)前処理，後処理のためのコマンド列，

に関する知識が与えられている。

知的エディタの使用例を以下に示す。

```
> (edit factorial)
(DEFUN FACTORIAL (N)
       (COND (ZEROP N) 1)
       (T (TIME N (SUB1 N))))
? move the right parenthesis of cond phrase to tail.
(DEFUN FACTORIAL (N)
       (COND (ZEROP N) 1 (T (TIME N (SUB1 N)))))
? parenthesize the second expression ans "1" of cond phrase.
(COND ((ZEROP N) 1) (T (TIME N (SUB1 N))))
? replace "time" with "times".
(COND ((ZEROP N) 1) (T (TIMES N (SUB1 N))))
? insert "factorial" after "n".
WHICH "N"?
? the second "n".
(TIMES N FACTORIAL (SUB1 N))
? parenthesize last two expression.
(TIMES N (FACTORIAL (SUB1 N)))
? print all.
(DEFUN FACTORIAL (N)
       (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N))))))
? ok.
STORED AS THE DEFINITION OF FACTORIAL
NIL
>
```

## あとがき

筆者らの研究室で開発してきたプログラミング開発支援システムには，本稿で述べたもののほかに，パターン指向型プログラミング・システム [6]，オブジェクト指向型プログラミング・システム [7] がある。

パターン指向型プログラミング・システムは，lisp関数の入力と出力の仕様をパターンで与えることにより，容易にlispプログラムを開発できるようにしたものである。

オブジェクト指向型プログラミング・システムは，Maclispのflavorに似たオブジェクト指向型プログラミングの機能をUtilispに組み込んだものである。このシステムを用いて天気図の表示システムの開発が行われている。

これらのシステムはまだ有機的に統合されたものではないが，これらをまとめて一つのプログラミング環境としていくことは今後の課題である。

## 謝辞

日頃ご指導頂く名古屋大学 稲垣康善教授，鳥脇純一郎教授，並びに福村・稲垣・鳥脇・吉田研究室の皆様に感謝致します。

参考文献

(1)Teitleman, W, 「Interlisp Reference Manual」, Xerox Palo Alto Research Center, 1974

(2)「UtiLispマニュアル」, 富士通, 1984

(3)太田, 吉田, 福村, 「UtiLispプログラムの解析ツール」, 昭和61年度電子通信学会総合全国大会(1718)

(4)太田, 七條, 吉田, 福村, 「LISPプログラムの動的解析システム-LISPDAP-」, 電子通信学会論文誌D

(5)吉川, 吉田, 福村, 「LISPプログラミングのための知的エディタの構成」, 信学技報AL85-21-26

(6)川村, 吉田, 福村, 「LISPのためのパターン指向型プログラミングシステム」, 信学技報AL85-1-9

(7)小林, 太田, 吉田, 福村, 「Lispによるオブジェクト指向プログラミングの図形処理への応用」, 昭和60年度東海支部連合大会(481)