FORTRAN によって実現された会話型 LISP システムとその応用†

太田義勝†† 吉田雄二†††
稲垣康善†††† 福村晃夫†††††

LISP は，リスト処理向きプログラミング言語であり，プログラムとデータが同じ構造（S式）であること，再帰手続きの記述が容易であることなどの特徴がある．数式処理，人工知能の分野の研究に広く用いられ，各地に多くの処理系が，開発されている．本論文では，移植性に重点をおいて FORTRAN によって実現された会話型 LISP システム (LISP 38, LISP 50) について，その言語仕様，システム構成，実現，応用について述べる．本システムの特色としては，会話型システムとして INTERLISP をモデルとしていること，プログラム開発のために，会話型エディタ，プリティプリンタ，コンパイラなどが用意されていること，仮想機械によって中間コード（コンパイラのオブジクト・コード）が実行されることなどがあげられる．本システムは，中型計算機 FACOM 230-38 (LISP 38)，ならびに，ミニコンピュータ OKITAC System 50/40 (LISP 50) 上に実現され，その移植性が実証されている．また，LISP 38 は，線図形文法推定システムの作成に用いられ，成果をあげている．

1. まえがき

人工知能，数式処理等の分野において，LISP は重要なプログラミング言語の座を占めている．それに伴い，LISP の処理系が，数多く実現されている．また最近では，LISP マシンの試作も，盛んに行われるようになった．

LISP が，このように幅広く使用されるようになってきた今日，移植性のよい，かつ，実用に耐えうる LISP システムの意義が，高まってきていると思われる．また，LISP システムの利用のあり方も，扱う問題の性質上，バッチ処理よりも会話形処理によるほうが，プログラムの開発・応用に適していると思われる．

高級言語によるシステムの記述は，システムの移植性，開発，保守の容易さなどの点において，アセンブラ言語により記述されたシステムより優れている．反面，実行速度，記憶装置の使用効率の面で，アセンブラ言語によるものにおとるので，この点を，いかに補うかが，問題となる．

本論文では，高級言語として，現在最も広く利用されている言語の一つである FORTRAN により記述された会話型 LISP システムについて，その設計方針，実現手法，性能評価，ならびに，応用について，述べる．本システムは，単に会話型の LISP システムというだけでなく，INTERLISP[1] をモデルにして構成された言語とそのインタプリタ，プログラム開発を容易にする強力なエディタ，効率よくプログラムを実行させるためのコンパイラを備え，一まとまりのシステムとして，十分実用に耐えうるものになっている．本システムは実際に2種類の計算機システムの上で実現され，その移植性が確認されている．これら二つの LISP システムは，実現に用いた計算機システムに因んで，それぞれ LISP 38[2]，LISP 50[3] と呼ばれる．

以下では，まず2章において，システムの構成として，言語仕様，処理系の構成，コンパイラ，プログラム開発支援機能について述べ，3章においては，システムの実現の面から，データ構造，インタプリタ，仮想機械，コンパイラ等の実現方式について述べる．4章では，システムの性能評価について述べ，最後に5章において，本システムを実際に応用した例として文法推定システムの構成への応用について述べる．

2. システムの構成

本システムは，LISP 1.5 をもとに INTERLISP の特徴をとり入れた言語仕様をもつ LISP 言語と，こ

† Interactive LISP System Written in FORTRAN and Its Applications by YOSHIKATSU OHTA (Education Center for Information Processing, Nagoya University), YUJI YOSHIDA (Department of Electrical Engineering, Nagoya University), YASUYOSHI INAGAKI (Department of Electronic Engineering, Mie University) and TERUO FUKUMURA (Department of Information Engineering, Nagoya University).
†† 名古屋大学情報処理教育センター
††† 名古屋大学工学部電気工学科
†††† 三重大学工学部電子工学科
††††† 名古屋大学工学部情報工学科

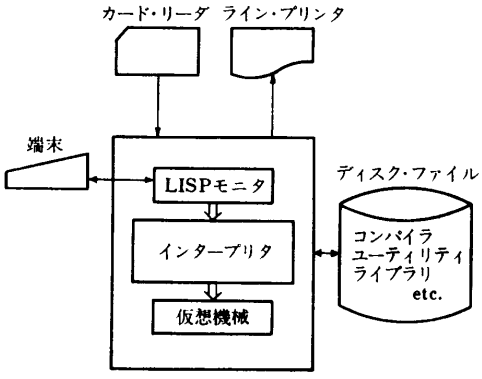

図 1 システム構成
Fig. 1 System organization.

れを処理する系としてのインタプリタとコンパイラ，および，システム利用者の便宜を図るための種々のユーティリティなどから構成される．**図1**に本システムの構成図を示す．

以下では，初めに本システムが扱う言語の仕様について述べ，次に，本システムのインタプリタ，ならびに，コンパイラについて述べる．最後に，本システムに用意されているユーティリティについて述べる．

2.1 言語仕様

本システムの LISP 言語の仕様は，LISP 1.5 を基に，会話型 LISP システムとして著名な INTERLISP の言語仕様を参考にして決められている．

INTERLISP から取り入れた機能としては，

ⅰ）implicit progn 機能

ⅱ）各種のユーザ定義関数のタイプの定義機能

がある．また，本システム独自のものとして，

ⅲ）ループ構成用組込み関数 (WHILE, REPEAT)

ⅳ）ファイル入出力機能

ⅴ）ライブラリ機能

がある．

以下では，これらのおのおのについて，やや詳しく述べる．

（1） implicit progn 機能

implicit progn 機能は，LISP 1.5 のラムダ式，または，条件式の機能を拡張する．すなわち，ラムダ式

(LAMBDA vars $e_1, e_2, \cdots, e_n$)

は，変数の束縛を行ったのち，形式 $e_1, e_2, \cdots, e_n$ を順に評価して，e_n の値を関数の値とする．また，条件式

(COND

($p_1, e_{11}, e_{12}, \cdots, e_{1m_1}$)

($p_2, e_{21}, e_{22}, \cdots, e_{2m_2}$)

.....

($p_n, e_{n1}, e_{n2}, \cdots, e_{nm_n}$)

は，$p_1, p_2, \cdots, p_n$ を順に評価して，NIL でない値をとる最初の p_i について，$e_{i_1}, e_{i_2}, \cdots, e_{im_i}$ を順に評価して，e_{im_i} の値をこの式の値とする．

表 1 ユーザ定義関数
Table 1 User-defined Functions.

	expr	expr*	fexpr	fexpr*	cexpr
関数適用前の引数評価	する	する	しない	しない	する
引数の個数	一定	任意個	一定	任意個	一定
定義本体	ラムダ式	ラムダ式	ラムダ式	ラムダ式	仮想コード

（2） ユーザ定義関数のタイプの多様化

ユーザ定数関数のタイプとして，expr, expr*, fexpr, fexpr*, cexpr の五つがある．これらの関数タイプの性質を**表1**に示す．expr, expr*, fexpr, fexpr* のタイプの関数のラムダ式による記法は，それぞれ以下のとおりである．

expr : (LAMBDA vars $e_1, e_2, \cdots, e_n$)

expr* : (LAMBDA var $e_1, e_2, \cdots, e_n$)

fexpr : (NLAMBDA vars $e_1, e_2, \cdots, e_n$)

fexpr* : (NLAMBDA var $e_1, e_2, \cdots, e_n$)

ここに，vars は変数リスト，vars は不定個の引数のリストをうけとるための変数，e_i は形式である．

cexpr は，expr 関数をコンパイルして得られたオブジェクト・プログラム（仮想コードと呼ばれる）で表わされる関数である．

（3） ループ構成用組込み関数

LISP 1.5 等では，プログラム中にループを構成するために，PROG 形式中で，関数 GO とラベルを用いて行うが，本システムでは，このほかに，組込み関数 WHILE, REPEAT を使用することにより，ループ構造を作ることができ，構造的な LISP プログラムを書くことを，可能にしている．

(WHILE $e_1, e_2, \cdots, e_n$)

は，形式 e_1 が NIL でないあいだ，形式 $e_2, e_3, \cdots, e_n$ をくりかえし実行する．

(REPEAT $e_1, e_2, \cdots, e_n$)

は，形式 e_n が NIL でなくなるまで，形式 $e_1, e_2, \cdots, e_{n-1}$ をくりかえし実行する．

（4） ファイル入出力機能

本システムでは，テキストファイルを扱うことができる．ファイルはカードイメージの順編成ファイルとして扱われる．

LISP プログラムからディスク上のファイルのオープン，クローズ，生成，削除を行うための機能として，次の組込み関数が用意されている．

(OPEN ファイル名) 値は論理機番

(CLOSE 論理機番) 値は NIL

(CREATE ファイル名) 値は論理機番

(REMOVE ファイル名) 値は NIL

ディスク・ファイルとの間のS式を単位とする入出力は，INTERLISP の仕様と同じように，関数

(READ 論理機番)

(PRINT S式 論理機番)

により行う．(論理機番が省略されると，端末との入出力となる．論理機番5と6は，それぞれカード・リーダとライン・プリンタに対応している．)

このほかに，文字単位の入力，改行，改ページ，ファイルの巻き戻しなどのための関数が，いくつか用意されている．

(5) ライブラリ機能

ライブラリは，関数名とラムダ式の対を格納したディスク・ファイルである．ライブラリには，システム・ライブラリと，ユーザ・ライブラリの2種類がある．ユーザは，システム・ライブラリを，変更することはできない．

ライブラリからの関数のロードは，組込み関数 LOADFNS により行われる．ライブラリの探索は，ユーザ・ライブラリ，システム・ライブラリの順に行われる．

ユーザ・ライブラリへの関数の登録は，組込み関数 SAVEFNS によって行われる．

2.2 処理系の構成

本システムでは，ユーザが端末から入力する LISP プログラムは，モニタ，インタプリタ，および，仮想機械などにより処理される．以下に，これらの機能およびそれらの間の関係を述べる．

(1) モニタ

モニタは，ユーザとの会話を制御する．モニタは，端末にプロンプト ▽<−▽ を出力して，ユーザに評価すべき eval 形式のS式の入力を要求する．S式が入力されるとモニタは，インタプリタを呼び出してその式を解釈させ，その値を端末に出力する．

(2) インタプリタ

インタプリタは，モニタから渡された LISP プログラムの解釈実行を行う．実行中に，関数タイプ cexpr の関数が現われると，仮想機械を呼び出して，実行を依頼する．

本システムのインタプリタの特色として未定義関数の処理があげられる．すなわち，インタプリタの動作時に，未定義関数を検出すると，ライブラリ中を自動的に探索し，もしその関数が登録されていれば，ロードして計算を続行することを可能とした（登録されていなければ，未定義関数のエラーとなる）．

(3) 仮想機械

インタプリタから呼び出されて，関数タイプ cexpr の関数を実行する．

仮想機械の構造，実現については，3.3 節で詳しく述べるのでここでは省略する．

2.3 コンパイラ

コンパイラは，expr タイプの関数プログラムを仮想コードに変換する．コンパイラは，ディスク・ファイル上に格納されていて，必要なときにシステム内に読みこまれる．

コンパイラにより出力されたオブジェクトプログラムは，ローダ (LISP 組込み関数) により，システム内の仮想コード領域にロードされ，cexpr タイプの関数として登録される．

実現されるシステムの規模を抑えるために，コンパイルされたプログラムからは，インタプリタを呼べないようにしている．

2.4 プログラム開発支援機能

プログラムの開発を支援する機能が整備されていることは，本システムの特徴の一つである．具体的には，LISP システムでしばしばデバッグのために組み込まれているトレース機能，バックトレース機能のほかにユーティリティ・プログラムとして会話型エディタ，プリティ・プリンタ，および，コールツリー印刷プログラムが，LISP で作成され用意されている．以下では，これらユーティリティ・プログラムのおのおのについて述べる．

(1) エディタ

会話型システムにおいては，プログラムの編集のためのエディタは，不可欠なものである．本システムには，INTERLISP のエディタをモデルとした会話型エディタが，関数 EDITF として，組み込まれている．このエディタは，フリー・スペース中の関数定義（ラムダ式）を直接編集するもので，一般に用いられているテキスト・エディタと異なり，LISP 専用のものである．

本エディタは編集対象であるラムダ式中のある部分

リストを指す“ポインタ”をもっている．すべての編集コマンドは，この“ポインタ”の指しているリストに対して適用される．

編集コマンドとしては，リストの印刷，“ポインタ”の移動，リストの挿入，削除，置換を行うコマンドなど約 30 種が用意されている．

特徴的なコマンドとしては，次のものがあげられる．

i）LISP プログラムでよくあらわれるカッコ構造の誤りを修正するコマンド（たとえば，i 番目から j 番目のリスト要素をカッコで囲む BI コマンド，その逆を行う BO コマンド）

ii）パターンマッチングによって“ポインタ”を移動させるコマンド（F コマンド）

iii）リストの挿入，削除，置換など，リスト構造を変更するようなコマンドを間違えて実行した場合に，元に戻すコマンド（UNDO コマンド）

（2） プリティ・プリンタ

LISP プログラムのプリティ・プリンタとして関数 PRETTY-PRINT が用意されている．PRETTY-PRINT は，関数の定義本体を，ラムダ式，条件式，PROG の構文にあわせて段づけを行い出力する．

（3） コール・ツリー

LISP プログラムを構成するユーザ定義の関数群の間の呼出し関係を表示する関数 CALL-TREE が，用意されている．

3. システムの実現

本 LISP システムは，二つの計算機システム上に実現された．それぞれのシステムは，実現に用いた計算機システムに因んで，LISP 38, LISP 50 と呼ばれている．

LISP 38 は，中型計算機 FACOM 230-38 の OS II/VS 上に，LISP 50 は，ミニコンピュータ OKITAC system 50/40 の DOS 上に，それぞれ実現された．LISP 50 は，LISP 38 を，ハードウェアの規模，O.S. の機能にあわせて，システムの規模（フリー・セル数，スタックサイズ等），端末入出力，ディスク・ファイル入出力を修正したものである．**表 2** に，LISP 38, LISP 50 のハードウェア環境，システム規模等を示す．二つのシステムの FORTRAN ステップ数が，かなり異なるのは，LISP 50 システムでは，O.S. の機能を考慮して，動的ファイル管理機能を削除したことによる．

表 2 LISP 38 と LISP 50 との比較
Table 2 LISP 38 vs. LISP 50.

	LISP 38	LISP 50
計算機と OS	FACOM 230-38 OS II/VS	OKITAC System 50/40 DOS
主記憶	384 kB	192 kB
ディスク容量	200 MB	20 MB
フリー・セル	12 k セル	8 k セル
文字アトム	500 個	500 個
スタック・サイズ	5 kW	3 kW
仮想プログラム領域	5 kW	3 kW
端末 I/O	メーカ提供のオンライン用モジュール (SOM)	アセンブラで作成
FORTRAN ステップ数	約 3,000	約 2,000

以下では，内部データ構造，インタプリタ，仮想機械，コンパイラ，ならびに，ユーティリティの実現について述べる．

3.1 内部データ構造

本 LISP で使用可能なデータ型であるリスト・セル，文字アトム，数値アトムの内部表現形式について述べる．

（1） リスト・セル

CAR, CDR の 2 語（1 語 16 ビット）より構成される．ガーベッジ・コレクション時のリスト・セルのマークづけは，CDR 部の符号を反転させることにより行う．

（2） 文字アトム

図 2 に文字アトムの内部表現を示す．印字名は，文字コードのリストで表現されている．関数定義は，expr 系の関数では，ラムダ式へのポインタ，cexpr 関数では，仮想コード領域へのポインタ，組込み関数では，関数通番がそれぞれ格納されている．

（3） 数値アトム

数値アトムとしては，small integer のみが扱われる．$|n| \leqq \text{maxint}$ を満たす整数値 n が，$n+\text{izero}$ を値とするポインタとして内部表現される．ここに，maxint, izero はそれぞれ次の式に従う．

$$\text{maxint} \leqq \frac{(32767 - n_1 - n_2)}{2}$$

値セル	P-リスト
P-ネームの文字数	P-ネームへのポインタ
関数タイプ*	関数定義*
引数の個数*	—

* 関数名アトムの場合

図 2 文字アトムの構成
Fig. 2 The organization of literal atom.

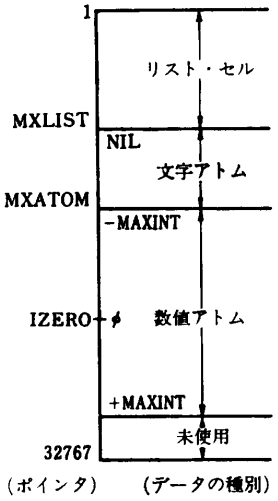

図 3 ポインタとデータの種別の関係
Fig. 3 Relation between pointer and type of data.

$$izero = n_l + n_a + maxint + 1$$

n_l, n_a はそれぞれリスト・セルの最大個数，文字アトムの最大個数である．

図3に，ポインタの値とデータの種別の対応を示す．

3.2 インタプリタの実現

インタプリタは，その本体を構成する FORTRAN サブルーチン EVAL と，組込み関数を実現するサブルーチン群とから構成されている．インタプリタを実現するに際して問題となる変数の束縛方式，および引数の評価方式については，本システムでは，次のような方策を採用した．

(1) 変数の束縛には，シャロー・バインディングを用いる．

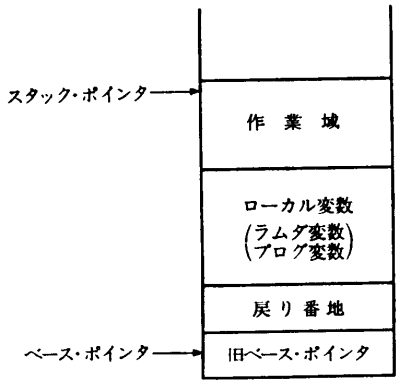

図 4 仮想機械のスタック構成
Fig. 4 Stack configuration of virtual machine.

(2) subr, expr 関数に対して引数の評価リストを作らずに，スタック上に評価された引数を直接扱う．

3.3 仮想機械の実現

コンパイラの出力する仮想コードは，仮想機械によって実行される．仮想機械は，一種のスタック機械でスタック1個と，レジスタ1個をもっている．仮想機械のスタック使用形式を図4に示す．スタックに対するプッシュ，ポップの操作は，レジスタを介して行われる．関数の評価は，レジスタ上の値およびスタックの頭部の値（引数が一つの場合はレジスタ上の値のみ）を引数として行われ，その結果はレジスタの上に求められる．したがって，条件分岐，および，1引数の組込み関数の実行は，スタック操作なしで実行される．仮想機械は，また文字アトム中の値セルを，大局的変数として使用する．

仮想機械の命令は，全部で 21 種類ある．その一覧を，表3に示す．レジスタに値をロードする命令 LI,

表 3 仮 想 命 令
Table 3 Virtual Instructions.

名 称	記 号	引 数	機 能
Load immidiate	LI (PLI)	S式	レジスタにS式をロードする
Load	L (PL)	ベース・ポインタからのオフセット	レジスタにローカル変数の値をロードする
Load global	LG (PLG)	文字アトム	レジスタにグローバル変数の値をロードする
Jump	J	番 地	ジャンプ
NIL jump	NJ	番 地	レジスタの値が NIL ならジャンプ
T jump	TJ	番 地	レジスタの値がTならジャンプ
Store	ST	ベース・ポインタからのオフセット	レジスタの値をローカル変数へストア
Store global	STG	文字アトム	レジスタの値をグローバル変数にストア
Mark	MK (PMK)	—	レジスタにφをロード
Bind	BIND	プログ変数の個数	プログ変数の確保と初期化
Call cexpr	C (PC)	文字アトム	cexpr 関数の呼出し
Call subr	C0, C1, C2	関数通番	subr 関数の呼出し
Call subr*	CN	関数通番	subr* 関数の呼出し
Return	RT	引数の個数	cexpr 関数からのリターン

L, LG, MK, C には，実行効率を考慮して，命令実行前のレジスタ内容をスタックにプッシュした後，対応する同じ動作を実行する命令 PLI, PL, PLG, PMK, PC が対になって用意されている．

インタプリタと仮想機械との結合は，次のようにして行われる．仮想機械の呼び出し時には，インタプリタは，必要な引数をすべてスタックにセットした後，特別な戻り番地“0”で call cexpr 命令を実行する．これにより制御が仮想機械に移る．一方，仮想機械からインタプリタへの戻りは，仮想機械が戻り番地“0”の return 命令を実行したときに，行われる．このとき，スタック内の引数はすべてポップされていて，関数値がレジスタに入っている．

3.4 コンパイラの実現

コンパイラは，27 個の LISP 関数によって実現されている．コンパイラが生成するオブジェクト・プログラムは，

$(f_n \text{ vars}((\text{op}_1 \text{ arg}_1)(\text{op}_2 \text{ arg}_2)\cdots(\text{op}_n \text{ arg}_n)))$

の形をしている．ここで，f_n は関数名，vars はラムダ変数リスト，op_i, arg_i はそれぞれ仮想命令とその引数である (LISP プログラムの仮想コードへの変換規則については，文献 3) を参照)．

コンパイラは，ラムダ変数，プログ変数をスタック上のローカル変数に，それ以外の変数（自由変数）をグローバル変数に割り当てて，仮想コードを生成する．また，コンパイル時に，レジスタの状態をシミュレートすることにより，レジスタの値のプッシュを伴う命令と伴わない命令を，適切に使いわけることによって，3.3 節に述べた仮想機械のアーキテクチャが生かされるようにしている．

コンパイラにより生成されたオブジェクト・プログラムは，実行前にローダにより，システム内の仮想コード領域に格納される．ローダは，オブジェクト・プログラムをロードすると同時に，以下を実行する．

（1） 相対番地から絶対番地への変換

（2） 組込み関数名の関数通番への変換

（3） 関数名アトム・セル中の関数タイプを cexpr に，関数定義を格納した仮想コードの先頭アドレスにセット．

なお，ローダは，FORTRAN で実現されている．

インタプリタのみからなるシステムから，コンパイラを備えたシステムへの拡張は，インタプリタへの仮想機械の埋込みと，組込み関数としてのローダの実現，ならびに，LISP でコンパイラを書くことだけで，容易に行うことができた．

3.5 ユーティリティの実現

LISP で書かれたユーティリティ・プログラムを組込み関数と同様に使えるようにするために，インタプリタの未定義関数処理機能と，ライブラリ機能を用いて，ユーティリティ・プログラムを，本システムの中に次のようにして実現した．すなわち，ユーティリティを構成する関数群のファイルを作成し，一方，ライブラリ中には，各ユーティリティごとに同じ関数名で，そのファイル中の関数群をロードし，自分自身を呼ぶような関数（これをユーティリティ・ローダと呼ぶ）を登録する．このようにすることにより，ユーティリティが最初に呼ばれたときには，インタプリタの未定義関数処理により，そのユーティリティ・ローダがライブラリよりロードされて実行され，必要な関数をファイルからロードして，ユーティリティを実行する．2 度目以降のユーティリティの呼出しは，ユーティリティがすでにシステム内に存在するために，ただちに，実行される．

3.6 移植について

本システムは，移植性という点から，広く使用され計算機間で互換性の高い FORTRAN で記述されている．LISP 38, LISP 50 の経験から明らかになった移植に際して問題となる部分は，次の 2 点である．

（1） O.S. に依存する部分．たとえば，端末入出力，ファイル管理．

（2） システムの規模（メモリ，機器構成）に依存する部分．たとえば，フリー・ストレージの大きさ．

本システムでは，これらに次のように対処した．

① 端末入出力部分は，LISP 38 では，メーカ提供のオンライン入出力のサブルーチン・パッケージを用い，また，LISP 50 では，アセンブラ言語を用いて実現した．これらは，それぞれ 100 ステップ程度のプログラムで，容易に作成できた．(動的な) ファイル管理は，LISP 50 においては，実現困難のために削除した．

② フリー・ストレージの大きさを，メモリの容量にあわせるためには，プログラム中の対応する配列の大きさをすべて変換した．これは FORTRAN を，システム記述に用いる際の欠点の一つと思われる．

これら 2 点の問題点を除けば，計算機が，機種，規模ともに大きく異なっているにもかかわらず，移植が容易であったのは，システム記述に FORTRAN を用

表 4 実行速度とフリーセル消費量の測定結果
Table 4 Measurement results of execution speed and amounts of used free-cell.

	LISP 38			LISP 50			セル消費量
	インタプリタ(秒)	コンパイラ(秒)	比	インタプリタ(秒)	コンパイラ(秒)	比	
TPU-1	154.7	45.2	3.42	215.5	58.9	3.66	15,255
2	437.7	128.0	3.42	606.3	170.7	3.55	45,967
3	186.3	54.1	3.44	256.5	71.0	3.61	18,630
4	259.8	75.8	3.43	359.5	98.8	3.64	26,257
5	28.8	8.1	3.55	40.2	9.4	4.28	2,796
6	746.7	217.8	3.43	1041.9	293.2	3.55	75,913
7	166.4	47.6	3.50	229.1	62.2	3.68	18,347
8	116.3	33.2	3.50	158.2	41.9	3.78	11,439
9	87.6	24.7	3.54	119.7	31.6	3.79	8,397

表 5 仮想命令の静的出現頻度と動的実行回数 (TPU-6)
Table 5 Occurrence Frequency and counts of execution of virtual instruction (TPU-6).

命 令	静的出現頻度 (%)	動的実行回数 (%)
LI	44 (3.5)	15,025 (0.7)
PLI	21 (1.7)	7,909 (0.4)
L	236 (18.9)	455,426 (21.0)
PL	160 (12.6)	428,293 (19.7)
LG	2 (0.2)	3,584 (0.2)
PLG	7 (0.6)	8,683 (0.4)
J	99 (7.8)	93,896 (4.3)
NJ	77 (6.0)	274,296 (12.6)
TJ	10 (0.8)	23,930 (1.1)
ST	177 (13.9)	55,319 (2.5)
STG	3 (0.2)	3 (0.0)
MK	33 (2.6)	4,135 (0.2)
PMK	4 (0.3)	10 (0.0)
BIND	13 (1.0)	6,311 (0.3)
C	0 (0.0)	0 (0.0)
PC	39 (3.0)	127,386 (5.9)
C0	0 (0.0)	0 (0.0)
C1	214 (16.9)	344,476 (15.8)
C2	55 (4.3)	193,037 (8.9)
CN	37 (2.9)	4,145 (0.2)
RT	39 (3.1)	127,387 (5.9)

いたことによるところが大きいと思われる.

4. システムの性能評価

本システムの性能評価のために, LISP コンテスト[4] で使用された TPU プログラムに関して以下の測定を行った.

i) インタプリタ, 仮想機械の実行速度 (表4)
ii) フリー・セルの消費量 (表4)
iii) コンパイル時間, オブジェクト・サイズ
iv) 仮想コードの静的出現回数と動的実行回数 (表5)

以下では, これらの結果について考察する.

(1) インタプリタ, 仮想機械の実行速度

インタプリタの実行速度は, 計算機のスピードと, FORTRAN で実現されていることを考えると, まずまずの速度である.

コンパイルされたプログラムを仮想機械によって実行させた速度は, インタプリタと比較して約3.5倍速くなった (SORT のプログラムでは, 4.6〜4.8 倍であった).

(2) フリー・セル消費量

フリー・セルの消費量は, 変数の束縛にAリストでなくシャロー・バインディングを採用し, さらに, 引数リストも作成せずに, 直接スタック上の引数を処理することにより, 必要最小限に抑えられている. これは, フリー・スペースの小さいシステムでは, 重要である. また, インタプリタ, 仮想機械のいずれで実行してもフリー・セルの消費量に違いはない.

ガーベッジ・コレクション所要時間は, リスト・セルのほとんどがゴミのときに, 約0.8秒, 半分ほどゴミのときに約1.2秒である.

(3) コンパイル時間とオブジェクト・サイズ

コンパイル時間は, インタプリタで約 79.0 秒仮想機械 (コンパイルされたコンパイラ) で約 19.8 秒である. 仮想コード領域上でのオブジェクトのサイズは, 約 2.5kW で, ラムダ式としてフリー・スペース中にプログラムが占めるメモリの約半分である.

(4) 仮想コードの静的出現頻度と動的実行回数

静的出現頻度の測定結果より, L, PL, ST 命令によるローカル変数に対するロード, ストアが多いことが知られる. MK, PMK 命令を除いてすべて2語命令であることから, これらの命令を1語命令にすることにより, オブジェクトのサイズが約 25% 小さくなることがわかる. この改善の実現は, 今後の課題である.

動的実行回数の測定より, 次のことが知られる.

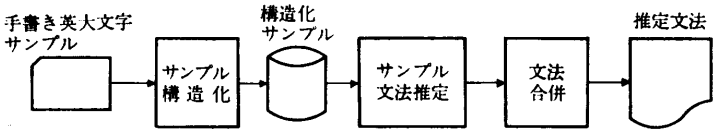

図 5 G.I. システムの構成
Fig. 5 The organization of the G.I. system.

① L 命令と PL 命令が，同じくらいの割合で実行されている．すなわち，ロードの半数はスタック操作を伴わない．

② 関数呼出しでは C1 命令が多い．この命令は，レジスタ上だけで処理が済む．

これらの結果より，仮想機械に採用したアーキテクチャは妥当であると判断する．

5. 応用例—文法自動推定システム—

本システム (LISP 38) は，実際に，筆者の研究室で，構文的パターン認識の研究の一環として，線図形サンプルから，それを言語として生成する文法を，自動的に推定するシステム (以下，G.I. システムと呼ぶ) の作成に用いられて，成果をあげている[5]．

以下では，G.I. システムで，本システムがどのように用いられているかを述べる．線図形の文法推定の詳細は，文献 6) に述べられている．

5.1 G.I. システム

G.I. システムは，あるカテゴリに属する方向指数系列で表された線図形サンプルを入力として，そのカテゴリの文法を推定する．

G.I. システムは，**図 5** のように，サンプル構造化，サンプル文法推定，文法合併の三つの部分から構成されている．サンプル構造化は，数値計算が主体であるので FORTRAN で実現されている．サンプル文法推定，文法合併は，扱うデータが構造をもったものであり，リスト処理が主体であるので，LISP 38 で実現された．

サンプル文法推定，文法合併の部分は，25 個の LISP 関数から構成されている．プログラムは，フリー・スペース中の約 2k セルを占め，1 サンプルあたりの処理時間は，2.0〜4.8 秒である．

6. あ と が き

本システムは，構文的パターン認識の研究に用いられているほかに，人工知能に関する学生の講義・演習にも，用いられている．

FORTRAN によって記述された LISP システムとして，実際の研究に応用でき成果を上げていること，LISP 38 から LISP 50 への移植修正が容易であったこと，コンパイラを含むシステムへの拡張が容易であったことなどにみられるように，高級言語で書かれた LISP システムが移植・保守・拡張の容易なシステムであることが示された．

LISP システムとして，より有用性を増すために残された課題としては，使用できるデータ型の拡張 (任意精度整数，ストリング，配列など) が，あげられる．

謝辞 最後に，日頃ご指導いただく，名古屋大学工学部本多波雄教授ならびに討論に参加いただいた名古屋大学本多・福村研究室，三重大学稲垣研究室の皆様に，深く感謝いたします．

参 考 文 献

1) Teitleman, W.: *Interlisp Reference Manual*, Xerox Palo Alto Research Center (1974).
2) 太田，吉田，福村：Interlisp の機能をとり入れた会話型 LISP の実現，昭和 52 年度電子通信学会総合全国大会 (1977)．
3) 太田，稲垣，吉田，福村：LISP 50 システムとそのコンパイラ，情報処理学会記号処理研究会資料 10-2 (1979)．
4) 竹内：LISP 処理系コンテストの結果，情報処理学会記号処理研究会資料 5-3 (1978)．
5) 太田，中山，吉田，福村：会話型 LISP の実現とその grammatical inference への応用，情報処理学会記号処理研究会資料 3-2 (1978)．
6) 中山，吉田，福村：不規則成分を伴う線図形の文法推定による構造抽出と記述の一方法，電子通信学会パターン認識と学習研究会資料 PRL 77-29 (1977)．

(昭和 56 年 1 月 5 日受付)
(昭和 56 年 12 月 17 日採録)