Bフレーム・Mフレームを使用したミニコンリスプ FLISP†

東出正裕†† 小西啓二†††
安部憲広†† 辻 三郎††

近年，人工知能に関する研究が急速に進展するに従い，その対象となる問題も巨大化し，そのために要する手法も複雑，多様化して来ている．こうした状況を背景として，種々のＡＩ用言語が開発されたが，処理効率の点で問題があり，あらためて LISP の機能強化の必要性が認識されるようになっている．

本稿で報告する FLISP は，これまでに開発された AI 用言語の基本的特徴の一つである多様な制御機構（たとえば，バックトラッキングやコルーチン等）を直接 LISP レベルで実現する（Bフレーム方式）とともに，リスト領域の一部を仮想記憶化することにより，約1Mセルのソースプログラムを実行しうる（Mフレーム方式）よう工夫されたミニコン向けの LISP である．

本文ではまず，FLISP のシステム構成，データ型について簡単に記した後，Mフレーム方式の概要とその具体化，必要となる特殊処理，および他の仮想記憶方式との比較検討を行う．そして次に，FLISP のBフレームと INTERLISP の frame との差異について論じた後，その実現手法といくつかの特殊関数について述べる．さらに，Mフレーム，Bフレーム導入に伴うガーベッジコレクションの変更点等について記し，最後に FLISP の評価として，LISP コンテストの例題を用いて，その処理速度，改善点等に関して検討する．

1. はじめに

LISP の柔軟なデータ構造操作能力を利用して，人工知能の分野で多くの研究がなされている．ただ，LISP には大容量の記憶領域が必要なため，従来は大型機を利用して LISP が実現されていた[1]．しかし，ますます発展する人工知能の研究にとって身近に利用可能な LISP が不可欠となり，小型機においても本格的な LISP が実現されている[2,3]．当研究室でも高機能の LISP が不可欠であるとの認識のもとに，ミニコン HP 2108 A に 256（将来は 512）kバイトの IC メモリを付加して，高水準ミニコンリスプ・FLISP を作成した．

現在の人工知能の研究の状況を考えると，その対象領域は言語・画像の処理を含む広範なものとなっており，そのために必要となる手法も複雑になっている．このような巨大な問題を解くための手続きが本質的に，大容量のプログラムと，複雑なプログラミング技法を必要とすることは明らかである．

まず，前者の問題であるが，FLISP は 128（現在 64）k セルの自由リスト領域を有しており，比較的大きなプログラムが実行可能である．しかし，近い将来人工知能研究のためには数百 k セルのリスト領域が不可欠であろうと予想されており，決して十分な量とはいえない．FLISP ではMフレーム (Module-frame) 方式と呼ばれる，特別なリスト領域にユーザの定義する関数集合（プログラム・モジュール）を単位としてオーバレイする方式を用いることで，最大 1M セルまでのプログラムを 16k セルで展開，処理することを可能にしている．

後者の問題は，Bobrow モデル[4]を採用し，コルーチン，バックトラック等の手法を簡単に定義・利用可能とすることで，ある程度解決できると考えられる．FLISP では，このモデルを変更して，プログラマの定義する関数（EXPR と FEXPR）単位にのみフレーム（Bフレームと呼ぶ）を作ることにより，Bフレーム記憶のオーバヘッドを軽減するとともに，制御構造の見通しを良くすることにも役立っている．このBフレームは IC メモリ上に格納されるため，多くのコルーチン，頻繁なバックトラックに必要なBフレーム領域の確保が可能であり，実際に巨大なプログラムを処理することができる．

2. FLISP の構成

ハードウェア構成は図1に示すように，補助記憶として現在 128k ワード（1ワード 18 ビット）の IC メ

† FLISP System for Minicomputer Using B-frame and M-frame Techniques by MASAHIRO HIGASHIDE, KEIJI KONISHI, NORIHIRO ABE, and SABURO TSUJI (Faculty of Engineering Science, Osaka University).
†† 大阪大学基礎工学部制御工学科
††† 富士ゼロックス(株)

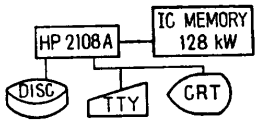

図 1　システム構成
Fig. 1　Hardware configuration.

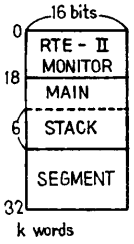

図 2　メインメモリのマップ
Fig. 2　Main memory map of the FLISP.

モリがあり，これを自由リスト領域として使用することにより，大容量のプログラムの実行が可能になっている．

FLISP は，HP 2108 A のディスクをベースとしたマルチプログラミングシステム RTE-II のもとで動作し，実行時の CPU メモリーマップは図2のようになっている．MAIN はメインメモリの容量不足のため，FLISPをセグメントに分割してオーバレイするためのモニタである．

3. FLISP のデータ構造

3.1 データの構成

FLISP では，データ型判別の簡単化，メモリ領域の能率的な使用を行うために，各データをタイプ別に IC メモリ上に実現している（図3）．M, B フレーム領域については後述するが，この領域は使用していな

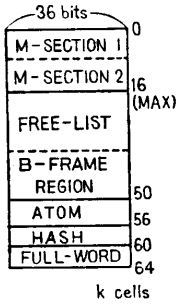

図 3　ICメモリの構成
Fig. 3　Organization of IC memory.

* 最下位ビットを利用して，勝手な car, cdr 部へのアクセスを禁止している．

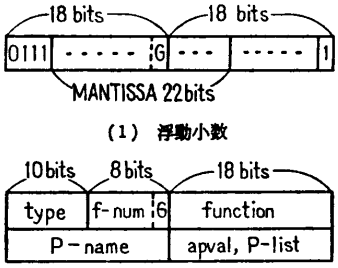

図 4　データ構造の例
Fig. 4　The data structure in the FLISP.

い場合，自由リスト領域となる．アトム領域は 1～8 k セルの範囲で可変であり，最大 4,096 個登録できる．ハッシュテーブル，フルワード空間は固定であり，−4,096～4,095 の整数がこの領域に対応する（FLISP ではストリングというデータ型はない）．なお，上述の数量は IC メモリの容量が 128 k セルに増設された時点では，倍増される．

3.2 LIFO スタックの構成

スタックは図2に示したようにメインメモリ上にあり，IC メモリ上のポインタ（18 ビット）用のデータスタックと，メインメモリ内のもどり番地（16 ビット）用のアドレススタックの2本で構成されていて，それぞれが 2 k 個の容量をもっている．

3.3 データの表現

データ型には，(a)自由セル，(b)小整数，(c)浮動小数，(d)文字アトム，(e)アレイがあり，(c)，(d)は図4の(1)，(2)に示すようになっている．浮動小数は特別な領域を設けず，自由セルの最下位ビットを1にして他と区別している*．文字アトムは2セルで構成されていて，type 部はそのアトムの使用状況を示すのに使用される．また，f-num 部には，Mフレーム使用時に必要となるMフレーム番号が記入される．function 部には，(f)subr のエントリ，または(f)expr の定義体へのポインタが記録され，array の場合には，アレイ本体へのポインタが書き込まれる．紙面の都合上，より詳細な事柄については，文献 14) に譲る．

4. Mフレーム方式

1章で述べたように，FLISP は比較的大きな自由リスト領域を有するが，より大規模なプログラムの実行を考えると，後述のBフレーム領域が 10～25 k セル程度は必要となり，決して十分ではない．ソースプロ

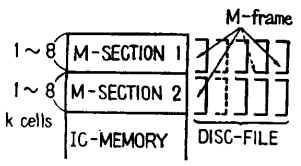

図 5 Mセクションへのディスクファイルのオーバーレイ
Fig. 5 An overlay of disc file onto the M-frame section.

グラムの展開を工夫して，自由リスト領域の実質的な拡張を計るために考案されたのがMフレーム方式である．

ところで，一般に大規模な問題を処理するプログラムの計算の流れに着目してみると，それはいくつかの部分問題を解くためのモジュールの連続処理を行っていると考えることができる．従って，処理の各時点で必要となるのは高々数個のモジュールであり，すべてのモジュールが必要なわけではない．Mフレーム方式は，プログラムをこうしたモジュールを単位として分割し，そのおのおののセグメントを自由リスト領域の専用領域（Mセクション）にオーバレイし，実際に必要とされるセグメント（以後，Mフレームと呼ぶ）をロードして実行するものである．

4.1 Mフレームの構成

Mセクションは**図5**に示すように，IC メモリの一部に2つ用意されており，一つのセクションのサイズは 1～8 k セルの範囲で設定でき，従って 16 k セルまでのサブ・プログラムが直接参照可能である．

Mフレームに展開されるべき関数群はソースプログラム内でユーザにより指定され，初期入力の際に各関数がMセクションで展開された後，そのポインタデータがバイナリ形式でディスクファイルに書き込まれる．Mフレームは展開される順にMフレーム番号が付され，その中で定義された関数名のアトム部の f-num にはその番号が記入される（Mセクション以外のリスト領域**に展開される関数の f-num は 0 である）．Mフレームの個数は 127 個まで許され1つのMフレームの最大の大きさが 8 k セルなので，約 1 Mセル* を要するソースプログラムを 16 k セルで展開・処理することができる．

* 正確には $8\times(2^7-1)=1,016$ k セルである．
** 以後，この領域を実リスト領域とよぶ．
*** ディスク↔メインメモリ↔IC メモリ間の高速データ転送をいう．
**** M フレーム番号 $M_i(0\leq i\leq 3)$ の M フレームが図5の M-SECTION 1 または M-SECTION 2 のいずれか一方でのみ展開されていることを意味する．

4.2 Mフレーム内関数の実行

インタプリタが expr, fexpr 関数を評価する場合，まずその f-num 部を見て，そのMフレーム番号と実際にロードされているMフレーム番号を比較する．f-num 部が 0 か同じ番号が発見されれば，ただちに実行する．そうでない場合は，必要なMフレームをそのMセクションにロードして実行する（この際，アドレス変更がないので，DMA (direct memory access) 転送***する）．

先にも述べたように，各処理の時点で必要とされる関数群がMセクション，または実リスト領域に存在しているならば，Mフレーム使用によるオーバヘッドはまったく無いといってよい．しかし現実には，異なるMフレーム内の関数を次々と実行して行かなければならない場合がある．そうした時のMフレーム間の呼び出し，もどりが正しく行われることが保障される必要がある．インタプリタは，あるMフレームが他のMフレームの関数を呼び出した場合，もとのMフレーム番号をスタックに記憶し，再びコントロールのもどる時点でそのMフレームを正しくロードするようになっている．

理解のため，次の例を考えてみよう．いま，F_1 が，(DE *F*1 (LAMBDA (X) (PROG 2 (*F*2 X) (*F*3 X))))と定義されていて，*F*1 が *F*0 から呼ばれたと仮定する．ここで，*F*0, *F*1, *F*2, *F*3 は皆異なるMフレーム内で定義された関数であり（*Fi* のMフレーム番号を *Mi* とする），そのMセクションが同一である****としよう．そうすると，*F*0 が *F*1 に制御を移す前に *M*0 がスタックに記憶され，続いて *M*1 のフレームがロードされて *F*1 の定義が読み出される．*F*1 は *F*2 を呼ぶため，*M*1 をスタックし *M*2 のフレームをロードして *F*2 が計算される．*F*2 の処理が終ると，スタックされていた情報により，再び *M*1 のフレームにもどって次に実行すべき関数が *F*3 であることがわかる．そこで，*F*2 の場合と同様の過程を経て *F*3 が計算される．ただし，*F*3 からもどる場合，実は *F*1 の残りの処理が値を返すことだけであるため，*M*1 のロードを省いて直接 *M*0 のフレームに帰って *F*0 を続行する．

実際には，プログラムを上例のようにセグメンテーションして用いるならば，Mフレームのロードによるオーバヘッドは深刻な問題となるだろう．しかし，ユーザがプログラムを作成してゆく時には，少なくとも，*F*1, *F*2, *F*3 は同じモジュールに含めるであろうし，またもし *F*2, *F*3 が多くの関数から頻繁に呼び出

される性質のものならば，実リスト領域に定義されるようにソースファイルを構成すると考えてよいであろう．

4.3 Mフレーム内の特殊データ

Mフレーム方式の基本的な考え方は，局所的情報の仮想化，すなわち，計算に用いられるデータの中で，将来任意の処理途上で必要となる可能性のあるものはすべて非仮想記憶（すなわち実リスト領域）に表現し，不要のページング（Mフレームのロード）を阻止することにある．従って，Mフレーム内のデータ，つまり関数定義体またはその一部が参照されるのは，その関数の実行されている時に限られるように工夫してある（その代償として，Mフレーム内のデータを勝手に読み出そうとすれば，エラーとなる．従って，定義の動的変更はできない*）．上述の原則を危うくするものは，実リスト領域に出して評価することにより，計算の正当性を確保している．具体的には，（1）fexprの引数リスト，（2）特殊なfsubrの引数リスト，たとえば，QUOTE, FUNCTION 等の引数リスト，（3）浮動小数，（4）計算結果によってはじめて関数名の決まるリスト（具体的には，EVAL の後半部の処理）** の4つが挙げられる．この内（1）～（3）はあらかじめ実リスト領域に展開しておき，（4）はインタプリタがMセクションにロードされているMフレーム内の対象リストを実リスト領域にコピーした上で，それを評価している．その理由を，一例をとって説明する．

(DE *F*5 (LAMBDA (X Y Z) (X Y Z)))

をMフレーム内の関数とする．*F*5 の呼び出す関数はXの値によって決定されるが，いまそれが同一のMセクションを使用する他のMフレーム内の fexpr 関数と仮定しよう．fexpr の計算では引数評価自体がその関数にまかされているため，まずその定義体が必要になる．ところが，定義を見て引数の処理を行う際には (X Y Z) の cdr 部のリストが必要であり，二つのMフレームの頻繁なローディングを生じてしまう．さらに，引数の一部がそのまま値の一部となったり，SET, SETQによってa-listに登録される場合がある（(2), (3) も同様の可能性を持つ）．そうすれば，このようなデータを参照しようとする度にMフレームのロードを必要とし（そのための準備もいる），効率が極めて悪くなってしまう．関数が expr の場合には，引数計算が先に実行されるし，その引数の値は a-list から引用されるか，QUOTE されている定数等であるため，上記の対策がなされているならば，そうしたリストは実リスト領域に存在しており，fexpr 等のような問題の生じることはない．なお，ある関数が fexpr として定義されているか否かを構文解析時に判別するには手間がかかるため，ソースプログラムの最初で宣言させることで解決している．

以上の操作により，Mフレーム領域外からMフレーム領域内を指すポインタは，関数定義体へのものと，PROG 形式による go-list 中のポインタのみとなる．前者はインタプリタが管理し，後者の参照されるのはその関数実行時に限られ，他の方法ではMフレーム内のデータは参照不能であるため，計算は正しく行われることができる．

4.4 Mフレーム方式と従来の仮想記憶方式との比較

Mフレーム方式も仮想記憶方式の一つであるが，実際の使用効率の面で大きな差異がある．通常の仮想記憶の場合，評価に要するすべてのセルが仮想記憶領域からとられている．そのため，処理の進展とともに参照の対象となるリストがいくつものページにまたがり，頻繁にページフォルトを生じる可能性が強い．Mフレーム方式では，プログラム・モジュールのみが仮想記憶化されているだけであり，セグメンテーションが余程不都合でない限り，はるかにすぐれているといえる．

また，文献 2) で提案された定義関数の掃き出し方式と比較すると，次のような差異を認めることができる．

〈a〉 Mフレーム方式

（1）定義体のロード等に関与する必要がない，（2）ロードの対象はバイナリーデータであり，単位は大きいが速度は速い，（3）欠点として，動的定義の変更がMフレーム内の関数に対して行えない，ことが挙げられる．（3）の欠点は，動的定義変更を要する関数は，Mフレームでなく実リスト領域を用いて定義すれば問題はなく，実用上の不都合はまったく無いといってよい．

〈b〉 掃き出し方式

（1）定義関数体のスワップを指定する必要がある，（2）スワップの対象は，バイナリ対アスキであり，変換をともなうので速度は速くない，（3）コルーチン，

---

* M フレーム内の関数を定義し直すと，新たな定義体はリスト領域に作られる．定義の一部を適当に変更することができないという意味である．

** T→eval[cons[sassoc[car[f]: *a*: λ[[]: error[EV 8]]]]; cdr[f]]: *a*]]: 以降の処理をいう．

頻繁なバックトラックの可能性のある場合，スワップに注意しないと定義体を掃き出してもすぐロードされ，一時的に同内容のリストの重複を生じ，効率が悪くなる，こと等を挙げることができる．いずれも一長一短であり，その可否を明確に論じることは現段階ではできない．

5. Bフレーム方式

人工知能等の問題解決には，バックトラッキング等の手法がよく用いられるが，これをLIFOスタックで処理しようとすれば，スタック保護の処理がめんどうになる．Bobrow モデルはスタック保護を容易にするために，Bフレーム方式という特別な環境記述方式を提案し，種々の有効な計算過程が実現可能なことを示している．

しかしながら，このモデルを直接ミニコンで実現するにはいくつかの問題がある．最も深刻な問題は，環境保持のために必要になるスタック量が莫大なものとなることである．文献8)にも示されているように，従来のLISPに比して数十倍のスタック量が必要になっている．さらに，環境保持によるオーバヘッドが大きい点も問題である．その原因が，文献4),7)の方式では，(a)評価に必要な変数テーブル等を記録するベーシックフレームをスタックに記憶させるため，スタック量が大きくなる．(b)環境の単位として個々の関数がとられているため，ユーザの関与しないインタプリタ関数に対してもBフレームが作られ，環境細分化によるBフレーム作成・回収のオーバヘッドが大きくなる．(c)Bフレームを一本のスタックに記憶させるため，Bフレーム回収によりスタックに空所を生じ，その再利用のために，スタック・ガーベッジコレクション，圧縮を頻繁に必要とする，という点にあると考えることができる．こうした問題を解決するために，FLISP では，次のような工夫を行ってある．

(1)ベーシックフレームを作らず，変数参照用にはa-list 方式を用い，Bフレームにはそのポインタのみを記憶させることにした．このことと(2)の工夫によって，必要なスタック量を大幅に減少させ，かつ環境保持のための手法続きを簡単にすることができた．(2)環境保持の単位をユーザの定義する関数 (expr, fexpr)単位と考え，Bフレームをより大きな単位で作っている．これによってBフレーム作成・回収によるオーバヘッドを軽減するとともに，Bフレーム相互の関係を理解しやすくすることができ，環境保持に無駄を生じることがない．(3)Bフレームをスタックに作らず，文献6),9)と同様にあらかじめ設定された単位ブロックを連結して用いることにより，(c)の問題点がなくなっている．

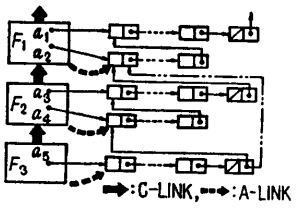

図 6 Bフレーム構造
Fig. 6 Snapshot of B-frame structure.

5.1 FLISPのBフレーム方式

FLISP のBフレーム方式の概略を図6を用いて解説する．関数 F_1 が F_2 を，F_2 が F_3 を呼んだ状況が図示されている．図中の F_i が関数 F_i の評価のためのBフレームを表わしている．個々のBフレームは，その関数評価後にどのフレームに制御をもどす（次にどの関数を計算する)かを示す C-LINK と，制御があるBフレームにもどった時点で，そこでの関数評価に必要な最新の a-list がどれであるかを示す A-LINK を持っている．図では，C-LINK が太線矢印で，A-LINK が破線矢印で示されている．LIFO スタックの場合，一つの関数評価終了時に次に計算する対象が時前に決定されているのに対し，Bフレームでは C-LINK, A-LINK に操作を加えることによって，動的に制御の流れ，変数参照過程の変更が可能である．FLISP の A-LINK の役目は文献 4),7) とは少し異なっていて，変数参照過程の変更のために用いられるだけであり，インタプリンタ自体の使用する a-list は各Bフレームに記憶されていて（図7の各Bフレーム内の a_j が a-list へのポインタを表わしている)，計算はこのポインタ・データをポップアップして用いる，つまり LISP 1.5 の評価形式をとっている．A-LINK は，例えば F_3 が F_2 の変数を見ることなく F_1 以前の変数を参照する場合，図中の二点鎖線のようなa-list の変更を行わせる際に利用される(後述の set-access 参照)．なお，図中のセル ◪ は各Bフレームの束縛交数の区切りを表わすためのものであり，文献4),7) のベーシックフレームの変数テーブルの終りに対応している．

以上のことからもわかるように，評価形式は完全にLISP 1.5 のままであり，インタプリンタへの変更はほとんど無く，expr, fexpr に対してBフレームの処

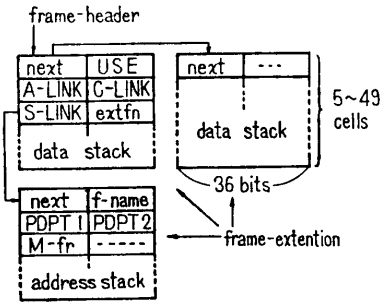

図 7 Bフレームの構成
Fig. 7 General B-frame structure.

理を加えるだけで良いようになっている.

5.2 Bフレームの構成

LISP 1.5モードでのスタックが3.2節で述べたように2本のスタックで構成されているため，Bフレームもそれぞれのスタックに対応するエクステンションを有している(図7)．一つのエクステンションの大きさは5～49セルの範囲で選択でき，一つのエクステンションに入らないデータは自由フレームを next に連結して，そこに格納される．図中の USE は，そのフレームの使用状況を示すものであり，通常は1である．USE カウントが2以上の B フレームは保護され，その関数の計算状況が保存される（詳細は文献 4) 参照)．PDPT 1, PDPT 2 はそれぞれデータ，アドレススタックのメインメモリ上のスタックポインタであり，M-fr は 4.1節で述べたMフレーム番号記憶用のものである．また，f-name には定義関数名が入り，extfn はそのBフレームを出る際の特殊処理関数をセットするためのものであるが，通常は NIL であり，その場合は何も行わない.

Bフレームは IC メモリのBフレーム領域に格納されているが，実行中の関数のBフレームはメインメモリへ DMA 転送されて用いられる．この場合，USE カウントが1のものは転送後はただちに回収されて自由フレームとなる．USE カウントが2以上のものは，その数を1減じた後メインメモリへ転写する（そのUSE は1にする)．逆に，スタックの内容を IC メモリへ掃き出す場合には，必要なエクステンションを自由フレームからとり，そこへ記録される．なお，単に USE カウントを変えたり，C-LINK を変更したりする場合は，直接その部分を書きかえ，上のような転送は行わない.

* フレームの解除は，USE を1減じ，もしその値が0ならばそのフレームを回収し，更にその C-LINK のフレームを解除する.

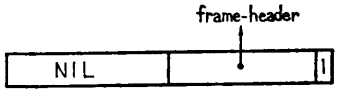

図 8 EDセル
Fig. 8 The structure of ED-cell.

5.3 Bフレーム関数

コルーチン等のための環境保護，変数参照変更用等のため，いくつか関数が組み込まれている．基本的であり，かつ重要なもののみここで示すことにする．以下で使用する pos, epos 等の表現形式は文献 4) に同一であるため，ここでは詳細については述べない.

(1) environ [pos]; pos で示されるBフレームを指す特別のデータ型である ED セルを作り，それを値とする．ED セルは図8のように浮動小数に準ずる構造をしていて，car, cdr への勝手な操作はエラーとなる．この関数によって pos の USE の値が1増されてその転写されたものが実行されることで，pos の示す環境が保存される.

(2) setenv [ed; pos]; ED セル ed のフレームヘッダを pos のBレームにする．特に，pos が NIL ならば ed の指示するBフレームを解除する*.

(3) enveval [form; apos; cpos]; コントロールを cpos に移し，form を apos の示す a-list で評価する (enveval 用のフレームは作らない)．cpos のない時は apos=cpos とみなされる.

(4) setcontrol [epos; cpos]; epos のBフレームの C-LINK を cpos のものに設定する．epos の以前の C-LINK が指していたBフレームは解除される.

(5) setaccess [epos; apos]; epos のフレームの束縛変数以後の部分に apos の意味する a-list を連結する（図6の a-list の変更は setaccess [$F3 : F1$] による)．この場合は(4)と異なり，フレームの解除は行わない．従って，図9のようにまったく別の系列のフレーム（例えばコルーチン）の変数を参照する場合，setaccess [$B1$; Bj] の後，setenv [ED 0; NIL] を行

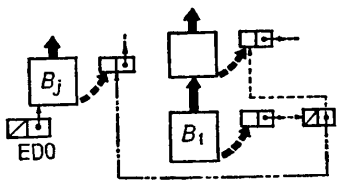

図 9 環境 Bj の下で $B1$ の評価
Fig. 9 Evaluation of function $B1$ in access context of Bj.

うと，Bj は解除されその a-list のみが残される．setenv しなければ，$B1$が解除された後も Bj は残されるが，ED0 がどこからも参照されない場合には，ガーベッジコレクションにより ED0 も Bj も解除回収される．

（6） nonframeset [fn-list]；fn-list 中の関数に対してフレームを作らないようにする．これは単なる recursive call によって，関数に入る度にBフレームが作られることによるBフレーム領域の消費を防ぐのに有効である．この場合，a-list は必要なだけ伸ばされ，前述のようにインタプリタは LISP 1.5 形式であるため，関数引数処理を含むすべての評価は正しく行われる．例えば，8-QUEEN のバックトラックの問題[4]を QUEEN のフレームのみで解くこともできる（実行中には，そのコピーが多数とられるため効率はよくない）．他にもいくつかあるが，紙面の都合上，省略する．

6. ガーベッジコレクション

インタプリンタは evlis 等で不要のセルを消費しないよう工夫されているので，従来よりもガーベッジコレクションの回数を減らすことができている．

ガーベッジコレクション自体は，Bフレーム，Mフレームの使用によって処理が異なっていて，通常のものより複雑になっている部分もある．

（1） BフレームもMフレームも使用しない場合自由セル不足，アトム領域の不足によりガーベッジコレクションが起こる．前者の場合，すべてのアトムに連結されているポインタ（定義関数体，アレイ，p-リスト等）およびレジスタ，スタック内のデータにマークを付して保護した後，リスト領域のごみが回収される．後者の場合は，印刷名 (p-name) のみをもつアトムには直接マークを付けず，そのアトムが実際に参照されている場合にのみマークを付す．マーキングの終了時に，マークのないアトムとセルを回収する．

（2） Bフレームを用いている場合

ガーベッジコレクションは自由フレームの欠乏によっても引き起されるが，本質的に異なるのはBフレームの保護と回収方法である．まず，その時コントロールのあるBフレーム内のポインタのマーキングから始める．その際 ED セルに出合うとそれを別のスタック*に保護し，マーキングを続行する．一つの B フレームの処理が終るとその C-LINK の指すフレームについて同様のことを行う．マーキングが完了した時点で，先程スタックされていた ED セルの指しているBフレームに対しても，同様の方法でマーク付けを行う（USE が2以上のものが重複して調べられることのないように，Bフレームにもマークを付す）．その他のリストの保護についても同様の手続きを用いる．最後に，不要セル，ED セル等を回収するが，ED セル回収時にはその指示するBフレームは解除される．このような ED セルの例として，例えば 5.3節(5)の図9に示された ED0 がある．

（3） Mフレームを用いる場合．

自由セル（Bフレーム）不足による場合は，Mセクション内を指すポインタを発見すると，そこでマーキングを停止する．というのは，Mフレーム内データはガーベッジコレクションの対象とせず，またMフレーム内から実リスト領域を指すポインタは4.3節で述べた特殊データとアトムだけであり，前者はあらかじめスタックより保護されており，後者は(1)と同様にしてガーベッジコレクションから守られているからである．

アトム領域の不足によってガーベッジコレクションを生じる場合には，上述のいずれの場合よりも時間を要する．なぜならば，どこからも参照されないアトムを発見するには，Mフレーム内のデータも調べなければいけないからである．その手続きの前半は，(1)の場合とほぼ同様であり，印刷名のみのアトムは他から参照された時初めてマークされ，(2)と同様にMセクション内のデータに対するマーキングは行わない．そして，Mフレーム外のポインタを保護した後，その時点でのMセクションの状況を記憶させる．そうした後，Mフレームを次々にMセクションにロードしそのポインタを調べ，まだマークの付いていないアトムを発見すれば印を付す．これをすべてのMフレームに対して行った後，不要のセル，アトムを回収し，Mフレームの状況をもとに復元して，処理を完了する．

7. その他の機能

FLISP は decode 管理システムであり，decode がエラーの受け手になっていて，decode を用いてユーザレベルの管理システムが容易に実現可能である．現在，185 種の組込み関数があり，エラーメッセージも 70 種用意されており，どの関数（変数）が不適切かが明示され，またバックトレースも定義関数ごとに区切られて表示されるため，迅速なデバッグが可能にな

* メインメモリ内のスタックの残りの部分を使用する．

っている．さらに，FLISP の PROG 変数には CONNIVER 同様の初期値設定機能があり，また LAMBDA, COND の述部にはインプリシット PROGN 機能も加えられていて，GO も PROG 内のどこにでも書けるようになっている．従来，LISP は入出力が貧弱であるといわれていたが，ファイルの入出力，CRT へのプロットを含めて 15 種の関数があり，極めて使いやすいシステムになっている．

8. FLISP の評価

表1は各種テストプログラムの実行時間を示したものである．WANG A, B は 50 回の平均であり，min は実行に必要な最小の B フレーム領域の大きさであり，単位は 1k セル，すなわち 2k データである．また，表の結果は一つのエクステンションを7セルに選んだ時のものである*．いずれの場合も B フレーム使用によるオーバヘッドは約一割であり，かつ最小の B フレーム領域も従来よりはかなり小さくなっている．これは FLISP の B フレーム方式の良さを示すものであり，IC メモリ上に任意の大きさに B フレーム領域がとれることを考えれば，十分実用的な問題が解けると結論してよいだろう．

問題点としては，LISP としての速度が少しおそい点がある．この原因としては，(1)a-list 方式のため，変数参照，set, setq に時間がかかる，(2)16 ビットマシンで 18 ビットデータを処理している，(3)リストが IC メモリ上にあり，読み書きに時間を要する，点がある．sassoc** は比較的マシン化しやすく (Bobrow モデルのベーシックフレーム方式より簡単)，(1)はハード化することで高速化し，(2)はマイクロプログラムで，型判別，情報転送，スタック処理を記述することにより，かなり改善できる．(3)は文献3)のように，専用の装置を付加するのが最適の方法である．このうち，(1)，(2)はインタプリタの変更点が少ないので，これらの改善を早急に行いたいと考えている．

表 1 プログラム実行例

Table 1 The results of executing some programs.

program name		LIFO-mode	B-frame	min
WANG	A	140 ms	160 ms	1 k
	B	720 ms	860 ms	1 k
BIT	A	17.3 s	18.9 s	1 k
(7 elements)	B	4.9 s	5.3 s	1 k
SORT	20	7.2 s	8.1 s	1 k
	60	35.4 s	40.5 s	1 k
	100	77.4 s	86.6 s	2 k
8-QUEEN		31.1 s	33.0 s	1 k
Backtraking		——	77.5 s	1 k

* この場合が，最も処理時間が短かかった．

** 変数参照のために，a-list を探索してゆく関数をいう．

9. おわりに

FLISP の諸機能・評価について述べた．現在，これを用いて自然言語処理，CONNIVER に似た言語 CONPAC の作成・利用が行われており，非決定的過程の処理が行われやすくなっている．今のところ，M フレームを必要とする程の巨大なプログラムは作成されてはいないが，近い将来，これは不可欠のものになると考えている．

なお，FLISP をより使いやすくするために，デバッグ機能の拡充，リスプ専用エディタの作成等が必要であり，IC メモリの増設および高速化も含めて改良してゆきたいと考えている．

最後に，資料の提供，御教示を頂いた電子技術総合研究所の島田俊夫・山口喜教両氏に感謝いたします．

参考文献

1) 黒川：LISPのデータ表現—TOBAC-5600 LISP を中心として—，情報処理，Vol. 17, No. 2, pp. 127-132 (1976).
2) 長尾，中村：高速補助記憶装置を使用したミニコン用 LISP 1.6 システム，情報処理，Vol. 17, No. 8, pp. 720-728 (1976).
3) 堂下，平松，角井：バルクメモリを使用したミニコン向き LISP システム (LISP 1.7)，信学会研資 AL-77-3 (1977).
4) D. G. Bobrow and B. Wegbreit : A Model and Stack Implementation of Multiple Environments, Commun. ACM, Vol. 16, No. 10, p. 591 (Oct. 1973).
5) D.G. Bobrow and D.L. Murphy : Structure of a LISP System Using Two-Level Storage, Commun. ACM, Vol. 10, No. 3, p. 155 (Mar.1967).
6) R. Greenblatt : The LISP Machine, MIT AI Lab. Working paper 79, (Nov. 1974).
7) 島田，山口，坂村：LISP マシンとその評価，信学論(D)．Vol. J 59-D, No. 6, pp. 406-413 (1976).
8) 島田，山口，坂村：LISP マシンとその評価，情報処理学会計算機アーキテクチャ研資(1974).
9) 山口，島田，守屋，坂村：ACE 上の LISP マシン，信学会研資 EC 76-13 (1976).
10) D. V. McDermott and G. J. Sussman : The Conniver Reference Manual, MIT AI Memo,

No. 259a (1974).
11) J. McCarthy: LISP 1.5 Programmar's Manual, MIT Press (1962).
12) 雨宮: LISPとその応用例, 産業図書 (1972).
13) 電子技術総合研究所: LISP User's Manual, EPICS 5-ON 2 (Mar. 1976).
14) 東出, 小西, 安部, 辻: Bフレーム, Mフレームを用いるミニコンリスプ FLISP について, 信学会研資 AL 77-55 (1977).
15) 安部, 小西, 東出, 辻: 関数フレームとBobrowフレームを持つミニコンリスプ FLISP, 52年信学全大 (6-74).
16) 大阪大学・辻研: FLISP User's Manual (May 1977).

(昭和53年1月24日受付)
(昭和53年4月28日採録)