記号処理 18－8
(1982. 3. 25)

マイクロプログラミング方式によるLISP処理系
― AIMの基本構想とプログラミング環境 ―

伊藤貴康　岸本光弘　田村卓
(東北大学　工学部)

1. まえがき

筆者等の研究室においては、プログラムの自動作成や検証、パターン理解、LSIのCADなどの研究用ソフトウェアをLISPベースで開発するために、高性能LISP処理系を必要としている。大規模な"組み合せ問題"(Combinatorial Problem)に対するLISPプログラムを実行させるために超大型機上での高性能LISP処理系と同程度の実行性能を持つLISP処理系を研究室内で実現することを計画した。[注:このような高性能処理系を研究室内に持つ利点の1つは、数日間にも亘る連続運転が容易に行なえる事である]

マイクロプログラミング方式によるLISP処理系AIMの研究・開発が上述のような観点から筆者等の研究室で進められている。本論文においては、AIMの基本構想とAIM固有のプログラミング環境について報告する。[注:AIMプロジェクト自身は、AIM 1.0, AIM 1.5, AIM 2.0のようなフェーズで計画され推進されている。AIM 1.0およびAIM 1.5はマイクロプログラミング方式によるLISP処理系として実現しているが、AIM 2.0では、その他の言語的特徴も取り込んだAIマシンあるいは推論型マシンとして発展させる予定である]

2. AIMの基本構想

マイクロプログラミング方式によるLISP処理系AIMは、筆者等の研究室に設置されているWCS(Writable Control Store)を持つマイクロプログラミング方式のミニコン上で核システムとして下記のような特徴を有する高性能LISP処理系を実現するという観点から計画された。

[I] LISPインタプリタのマイクロプログラム化による高速化：LISPインタプリタあるいはインタプリタに使われる関数群をマイクロプログラム化し、これらをWCS内に常駐させて直接実行することによって高速化を計る。

[II] ユーザ定義LISP関数のマイクロプログラム化直接実行：ユーザ定義LISP関数をマイクロプログラムに変換し直接実行することにより、処理速度を大巾に向上できるが、大規模なLISPプログラムに対してもこのような特徴を生かすために、
①マイクロプログラム作成支援環境
②マイクロプログラム実行支援環境
を用意する。

[III] LISP関数の解析・変換による実行性能の改善：LISPの特徴の1つは、再帰的関数定義によって、アルゴリズムを透明な形で記述・表現できる点にあるが、その代り実行効率が悪いことが多い。透明であるが効率の悪いプログラムを解析して、①実行効率のよいプログラムに変換したり②LISP処理系の実行に際してオーバーヘッドの少ない実行モードを取らせるようにすることが考えられる。特にWCSにロードされるマイクロ化LISP関数群の管理のためのLISPプログラムの解析とプログラム変換に基く効率の改善を採用する。

AIMの開発は次のようなフェーズで進められている。

1) AIM-1.0：[64Kセル, 2Kマイクロ語のWCS]で動作するLISP 1.5ベースの実験システムで、上述[I]および[II]を実証するという観点から作成され、基礎的な評価実験が行なわれた。(2.1参照)

2) AIM-1.5：[256Kセル, 2Kマイクロ語のWCS]で動作する実行システムで開発中。[I]および[II]の特徴に加えて[III]を支援するプログラミング環境を具備したシステムである。(2.2参照)

3) AIM-2.0：マイクロプログラミング方式による人工知能マシン(AIマシン)あるいは推論型マシンとして開発を計画中のもので、AIM-1.5で採用されたプログラム解析・変換・演算機能の高度化、プログラム検証、論理的プログラミングおよび代数的プログラミングなどの支援をAIMのマイクロプログラミング環境の下で行う予定である。

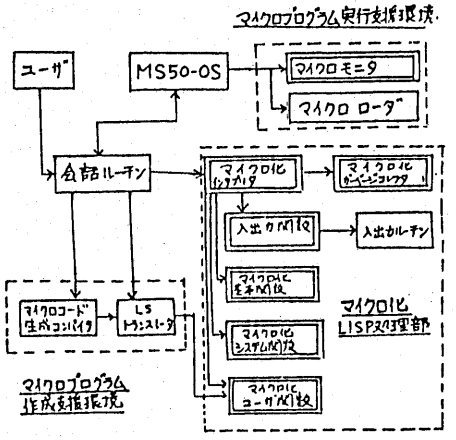
2.1 AIM-1.0の概要

AIMの特徴である
[I] LISPインタプリタのマイクロプログラム化
[II] ユーザ定義LISP関数のマイクロプログラム化直接実行

によるLISP処理系の高速化の実現法を確立し、高速性を実証するためにAIM-1.0が開発された。AIM1.0は筆者等の研究室に設置されているWCSを持ったマイクロプログラミング方式のミニコン[NEAC-MS-50(2Kマイクロ語のWCS, 2Mバイト主記憶実装可能)]上で作成された。図1にAIM1.0の構成と構成要素の機能説明を与えた。[注:LS言語はLISP向きの仮想スタックマシンの中間言語である。]

AIM-1.0の基礎的な評価実験結果を付録に与えたが、この結果からWCSを持ったマイクロプログラミング方式の商用ミニコン[MS-50]によってもAIMの方式によって超大型機[ACOS 900およびACOS 1000]の高性能LISP処理系に匹敵するLISP処理系が安価に実現できる事が知られる。

AIM-1.0の大きな特徴は大規模なユーザ定義LISP関数(群)を全てマイクロプログラム化して直接実行させる点にあるが、WCS領域が2Kマイクロ語に限定されているために、主記憶上にあるマイクロ化LISP関数を適宜、マイクロモニタによってWCSにロードして実行させる必要がある。AIM-1.0の問題点の1つとして、このオーバヘッドがある。このオーバーヘッドを少くするにはLISP関数群の実行時の関係を調べ、その結果をマイクロモニタのWCS管理に利用すればよく、AIM-1.5ではこのような観点からのプログラム解析機能が具備されている。なお、AIM1.0の詳細については文献[1]~[6]を参照。[注:AIM1.0の経験から、AIMの構想を実現するに適したミニコンのアーキテクチャとして①大規模WCS ②WCSがマイクロプログラム領域およびデータ領域として使用可能なこと ③再帰およびスタックをマイクロプログラムレベルでサポートすることなどが望ましい。MS-50はWCSをマイクロプログラム領域として使用できるだけである。]



- マイクロ化ユーザLISP関数：マイクロプログラム化されたユーザ関数。マイクロローダによって主記憶の ロード領域にロードされている。
- マイクロ・モニタ：マイクロ・ローダによってマイクロ・モニタのロード領域(主記憶上)にロードされた関数を必要に応じてWCSにロードして実行させる。
- マイクロ・ローダ：マイクロプログラムをマイクロモニタのロード領域にロードして、実行可能とする。
- LSトランスレータ：LS言語で書かれたプログラムをマイクロプログラムのアセンブラ・コードに変換する。
- マイクロコード生成コンパイラ：ユーザLISP関数をコンパイルして、LS言語のオブジェクトプログラムを生成するコンパイラ。

図1　AIM1.0の構成

2.2 AIM-1.5の概要

AIM-1.5は実験システムAIM-1.0の開発経験を踏まえて開発中であり、[256Kセル, 2Kマイクロ語のWCS]で動作する実用システムである。AIM1.5のシステム構成も図1に示したAIM-1.0と基本的には同じであるが、AIM1.5のAIM1.0との主たる違いは次のような点にある：

(1) セル領域を256Kセルにする事
(2) マイクロ化LISP関数に対するマイクロプログラミング環境を高度化する事
(3) ユーザに対するLISPプログラミング環境を整備する事

AIM1.0からの主な機能的な変更点に留意しながらAIM-1.5の設計方針について説明する。

(1) LISPインタプリタ：AIM-1.0ではインタプリタ関数を全てマイクロプログラム化したが、この結果がミニコンのOS環境下での機能拡充の点からは融通性のないものとなった。そこでAIM1.5ではインタプリタへのエントリーはアセンブラベースで行いインタプリタに使われる関数をマイクロプログラム化して実行する形式も検討している。またWCSをマイクロ化ユーザLISP関数に最大限に開放するために、アセンブラで書かれたインタプリタも用意しておく。

(2) マイクロプログラム実行支援環境：AIM1.0のマイクロモニタはシングルユーザのみを対象としていたがマルチユーザ用に拡張すること、および2.1で述べたマイクロモニタのオーバーヘッドを少なくするためにLISPプログラムの解析機能を具備させ、その結果をマイクロモニタが利用して効率のよい実行管理を行う。(3.1参照)

(3) マイクロプログラム作成支援環境：AIM-1.0ではLS言語とそのプログラミング, LSトラン

スレータ，マイクロオブジェクト生成コンパイラ(MGEN)によってユーザ定義LISP関数のマイクロプログラム化を支援して来た。AIM1.5では，いわゆるプログラム変換(program transformation)あるいはプログラム演算(program manipulation)の研究成果や考え方を用いた変換，翻訳を行い，LISPプログラムの実行効率の改良を取り入れることにした。(3.2参照)

(4) LISPプログラミング環境の整備：AIM1.0はLISP1.5を支援する実験システムであり，プログラミング環境が貧弱であったが，AIM1.5ではデバッギング機能の拡充，メモリ管理機能の拡充，プログラム解析・変換・演算機能，および例題系列・ドット系列記法を用いたプログラミングスタイル(EXPOL プログラミング)の埋め込みなどを行う。

3. AIM-1.5におけるマイクロプログラム実行・作成支援環境

AIM1.0においてマイクロプログラム実行支援環境としてマイクロモニタとマイクロローダ，マイクロプログラム作成支援環境としてLS言語とLSトランスレータ，マイクロオブジェクト生成コンパイラ(MGEN)を用意した。AIM1.5では

① LISPプログラム解析ルーチンを活用したWCSの効率の良い管理機能

② プログラム変換・演算ルーチンを用いた効率の良いマイクロオブジェクト生成機能

が装備されている。

3.1 LISPプログラム解析ルーチン

マイクロモニタによるWCSの動的管理によってユーザのWCS領域を超えるような大規模なマイクロ化LISP関数でもマイクロプログラム直接実行することができる。しかしWCSのロードには無視できないオーバーヘッドがあり，WCSの効率的実行管理を行う必要がある。このためにユーザ定義LISP関数(群)の振舞を解析してマイクロ化LISP関数のグループ分けを行い，ローディングによるオーバーヘッドを少くすることが考えられる。AIM1.5には，このような目的でLISPプログラム解析ルーチンLISDAPを具備させることが計画された。作製したLISDAPの機能を図2に示す。

LISDAPを用いて実際にかなり大きなLISPプログラムの解析を行なった例を図3に示す。解析したプログラムは(筧着等の研究室の重永哲君が開発した) SL-resolutionに基ずく定理の証明プログラムであり，約700行のLISPプログラムである。

まずリストアップ機能を用いて変数，関数の静的出現頻度を調べる。(a)では5回以上出現するもののみを示した。実際にプログラムを実行し各関数の実行回数を調べたのが(c)である。次に関数間の呼出関係を解析する。(b)にはトップレベルの関数 PROVER! が直接呼出す関数を呼出頻度と共に表示した例を示す。また(d)は各関数のマイクロプログラム化した時の概算ステップ数を示している。以上の解析をもとに，全体を4つのグループにわけ，グループを生成する。生成したグループの中で入力部であるグループAを構成する関数を(e)に示した。この時各グループの統ステップ数は(f)のようになる。このグループ構成で実行した時のグループ間の呼出関係を示し

	機能
(1) 関数定義のリストアップ機能	ユーザ定義関数中に現われる関数名，変数名，頻度を表示。WCSに常駐化すべきシステム関数の選定やタイプミスのチェックにも用いる。
(2) 実行回数の測定機能	ユーザ定義関数が何回実行されたかを測定する機能。入力部・前処理部・実行部・出力部の各部分における実行回数を調べたり，特定関数から呼び出される関数を調べたりできる。
(3) 関数呼出関係解析機能	ユーザ定義関数間の呼出関係を解析する機能。表示としては，①ある関数から直接呼び出される関数のリスト ②間接に呼び出されるものも含めたリスト，③ある関数を直接呼び出す関数のリスト，④間接も含め呼び出す関数のリスト がある。 また実際のデータを用いた実行において，呼出回数を測定するという動的解析を行なうこともできる。
(4) マイクロステップ数概算機能	S式で書かれているユーザ定義関数から，その関数をマイクロプログラム化した場合，どの位の大きさのマイクロプログラムになるかを概算する機能。再帰，条件文，prog文，map関数，関数の呼出等に重みをつけその合計をその関数のステップ数としている。また実際のマイクロ化した時のステップ数がわかっていれば，それも設定することができる。
(5) グループ生成機能	(1)~(4)の機能を用いて大まかなグループ分けができたならば，グループ生成機能を用いてグループを生成し，マイクロ化した時のシミュレーションを行なう事ができる。そのために，グループの生成，消去，追加，削除及び構成している関数の表示機能がある。
(6) グループ統ステップ数表示機能	グループを構成する関数のマイクロ化した時のステップ数を合計して表示する機能。MS-50のWCSは2Kマイクロ語であり，モニタ及びLISPシステム常駐部をのぞけば各グループを1.2Kマイクロ語程度におさめる事が望ましい。
(7) グループ間呼出関係解析機能	(3)の関数の呼出関係解析機能において同一グループ上での呼び出しは調べないようにしたもので，このグループ間の呼び出しが実際のWCSエリアへのプログラムのロードに対応する。
(8) マイクロプログラム実行ステップ数計算機能 (S-eval)	WCSの動的管理とは直接関係ないが，関数の計算の手間を調べるための機能として，その関数をマイクロ化インタプリタで評価してときのマイクロプログラムステップ数を計算する

図2 LISDAPの機能説明

図3 LISDAP の使用例

(a)

```
***** VARIABLE *****        ***** FUNCTION *****
CL2             =  88.      SETQ            =  179.
NIL             =  79.      CDR             =  106.
CL1             =  64.      QUOTE           =  98.
L1              =  63.      RETURN          =  95.
F               =  57.      GO              =  92.
I               =  45.      AND             =  85.
L               =  44.      PRINC           =  79.
CL              =  44.      OR              =  74.
/               =  41.      CAR             =  62.
*CENTER         =  39.      PROG            =  37.
E               =  36.      TERPRI          =  36.
L2              =  35.      CONS            =  31.
X               =  30.      CAAR            =  27.
T               =  30.      RPLACA          =  25.
PS1             =  25.      RPLACD          =  25.
MGU             =  25.      EQ              =  24.
BS              =  24.      CDAR            =  24.
M               =  20.      MAPC            =  17.
V               =  19.      ATOM            =  15.
*TREE           =  19.      PROG2           =  13.
BL              =  17.      CDDR            =  13.
*ITH            =  15.      COPY1           =  13.
PS              =  15.      CADR            =  12.
BCL             =  13.      TIME            =  11.
Y               =  13.      CADAR           =  11.
N               =  12.      PLUS            =  9.
U               =  10.      SUBSTC1!        =  9.
LAMBDA          =  10.      MERGE1          =  8.
L3              =  9.       DIFFERENCE      =  8.
*CLIST          =  6.       ADD1            =  6.
TREATING-TIME   =  6.       MAP             =  6.
L0              =  6.       CADDAR          =  6.
LN1             =  6.       SUB1            =  6.
MSEC            =  6.       SUBSTC1         =  5.
REF-TIME        =  6.       LESSP           =  5.
LL              =  6.       NULL            =  5.
L4              =  5.       CDDAR           =  5.
S               =  5.       PTREE1          =  5.
*DEPTH          =  5.       CAAAR           =  5.
*POSI           =  5.       UNIFICATION1    =  5.
*NEGA           =  5.       VAR1            =  4.
Q               =  5.       PROGN           =  4.
I1              =  5.       ZEROP           =  4.
/ +             =  5.       EQUAL           =  4.
```

(b)

```
? (CALL 'PROVER1 T)
PROVER1          ->
                 -> COPY1              38. TIMES
                 -> TIMER1             18. TIMES
                 -> RESOLUTION1        18. TIMES
                 -> ASSIGN1            18. TIMES
                 -> DELETE1            15. TIMES
                 -> RENAME1             6. TIMES
                 -> PAIR1               6. TIMES
                 -> LENGTH1             5. TIMES
                 -> INPUT1              1. TIMES
                 -> ARRAY1              1. TIMES
                 -> PRINTTIME1          1. TIMES
                 -> SEPARATE1           1. TIMES
                 -> DATA1               1. TIMES
                 -> OUTPUT1             1. TIMES
                 -> MODIFY1             1. TIMES
                 -> PRINTTREE1          1. TIMES
```

(c)

```
***** COUNT PRINT *****
"*** A-PHASE ***"
SUBSTC2!        =  1034.
SUBSTC1!        =  509.
NOTIN1!         =  265.
NOTIN1          =  191.
VAR!            =  146.
COPY1           =  110.
RENAME2!        =  107.
UNIF1!          =  89.
REVAR1          =  80.
OUTPUT5!        =  73.
NOTINSIDE1      =  66.
OUTPUT3!        =  55.
RENAME1!        =  47.
PTREE!          =  44.
PTREE1!         =  44.
UNIFICATION1    =  40.
SUBSTU1!        =  36.
INPUT3!         =  35.
OUTPUT4!        =  29.
ASSIGN1         =  24.
RESOLUTION1     =  18.
RESOLVE1        =  18.
SUBSTU1         =  18.
TIMER1          =  18.
DELETE1         =  15.
SUBSTC1         =  13.
TWICE1          =  12.
TAUTOLOGY1      =  12.
MERGE1          =  12.
INPUT2!         =  11.
OUTPUT2!        =  8.
PAIR1!          =  8.
FACTOR1         =  7.
RENAME1         =  6.
PAIR1           =  6.
LENGTH1         =  5.
REDUCE1         =  5.
UTREE1          =  4.
SEPARATE1!      =  4.
DATA1           =  4.
OUTPUT1!        =  4.
INPUT1!         =  4.
LTREE1          =  4.
MTREE1          =  4.
MODIFY1!        =  4.
SUBST1!         =  2.
PROVER1         =  1.
INPUT1          =  1.
SEPARATE1       =  1.
OUTPUT1         =  1.
MODIFY1         =  1.
PRINTTREE1      =  1.
PRINTTIME1      =  1.
ARRAY1          =  1.
```

(d)

```
MICRO STEPS (RECKONED)
ARRAY1          =  15.
ASSIGN1         =  4.
COPY1           =  4.
DATA1           =  3.
DELETE1         =  22.
FACTOR1         =  200.
INPUT1          =  8.
INPUT1!         =  34.
INPUT2!         =  20.
INPUT3!         =  32.
LENGTH1         =  29.
LTREE1          =  46.
MERGE1          =  101.
MODIFY1         =  9.
MODIFY1!        =  52.
MTREE1          =  76.
NOTIN1          =  24.
NOTIN1!         =  19.
NOTINSIDE1      =  2.
OUTPUT1         =  4.
OUTPUT1!        =  30.
OUTPUT2!        =  34.
OUTPUT3!        =  5.
OUTPUT4!        =  15.
OUTPUT5!        =  37.
PAIR1           =  10.
PAIR1!          =  296.
PRINTTIME1      =  146.
PRINTTREE1      =  17.
PROVER1         =  159.
PTREE1          =  18.
PTREE1!         =  32.
REDUCE1         =  109.
RENAME1         =  4.
RENAME1!        =  8.
RENAME2!        =  25.
RESOLUTION1     =  48.
RESOLVE1        =  76.
REVAR1          =  27.
SEPARATE1       =  4.
SEPARATE1!      =  103.
SUBSTC1         =  5.
SUBSTC1!        =  10.
SUBSTC2!        =  25.
SUBST1!         =  6.
SUBSTU1         =  4.
SUBSTU1!        =  25.
TAUTOLOGY1      =  60.
TIMER1          =  58.
TWICE1          =  54.
UNIF1!          =  119.
UNIFICATION1    =  44.
UTREE1          =  85.
VAR1            =  6.
```

(e)

```
(PGROUP 'GRPA)
***** LIST UP FUNCTION OF GROUP *****

GRPA CONSIST OF
DATA1
INPUT1
INPUT1!
INPUT2!
INPUT3!
PTREE1
PTREE1!
SEPARATE1
SEPARATE1!
```

(f)

```
? (GSTEP)
***** GROUP STEPS COUNT FACILITY *****

GRPA  =  254. STEPS
GRPB  =  376. STEPS
GRPC  =  1112. STEPS
GRPD  =  408. STEPS
```

(g)

```
? (GCALL 'GRPC T)
PROVER1          ->
                 -> RENAME1            6. TIMES
                 -> PAIR1              6. TIMES
                 -> SEPARATE1          1. TIMES
                 -> PRINTTIME1         1. TIMES
                 -> MODIFY1            1. TIMES
                 -> OUTPUT1            1. TIMES
                 -> ARRAY1             1. TIMES
                 -> PRINTTREE1         1. TIMES
                 -> DATA1              1. TIMES
                 -> INPUT1             1. TIMES
```

(h)

```
SAPPLY [ APP ; (NIL (B1)) ; NIL ] = 281.

SAPPLY [ APP ; ((A1) (B1)) ; NIL ] = 804.  +
    (PLUS  )
       (SAPPLY (QUOTE
                   (LAMBDA
                      (X Y)
                      (COND
                         ((NULL X) Y)
                         (T (CONS (CAR X) (APP (CDR X) Y))))))
               (QUOTE (NIL (B1)))
               (QUOTE ((Y B1) (X A1)))))

SAPPLY [ REV ; ((A1 A2 A3 A4 A5)) ; NIL ] =  822.  +
    (PLUS)
       (SAPPLY (QUOTE
                   (LAMBDA
                      (X)
                      (COND
                         ((NULL X) NIL)
                         (T
                            (APP
                               (REV (CDR X))
                               (CONS (CAR X) NIL))))))
               (QUOTE ((A2 A3 A4 A5)))
               (QUOTE ((X A1 A2 A3 A4 A5))))
    (SAPPLY (QUOTE
                (LAMBDA
                   (X Y)
                   (COND
                      ((NULL X) Y)
                      (T (CONS (CAR X) (APP (CDR X) Y))))))
            (QUOTE ((A5 A4 A3 A2) (A1)))
            (QUOTE ((X A1 A2 A3 A4 A5)))))
```

注)

(a) 関数・変数リストアップ機能.

(b) 関数呼出関係解析機能

(c) グループ生成機能

(f) グループ総ステップ数表示機能

(g) グループ間呼出関係解析機能

(c) 実行回数測定機能

(d) マイクロ・ステップ数概算機能

(h) マイクロプログラム実行ステップ数計算機能

たのが(9)である。

3.2 LISPプログラム変換ルーチン

AIMではユーザ定義LISP関数をマイクロ化直接実行する事により高速処理を実現している。LISP関数をマイクロ化するためにマイクロコード生成コンパイラMGENがAIM1.0に用意されている。MGENは構文主導型コンパイラであり、最適化ルーチンがあるとは言え、アルゴリズムの中に立入った最適化には限界がある。プログラム変換・演算の研究結果や考え方を利用することにより、極めて効率の良いマイクロオブジェクトを生成できる場合があり、AIM-1.5にはこのような目的のためにプログラム変換ルーチンが用意されつつある。例えば

rev[x] ⇐ if null[x] then NIL
else append[rev[cdr[x]]; cons[car[x]:NIL]]

は、リスト長nの入力に対してconsを$O(n^2)$呼ぶのに対し、

rev[x] ⇐ R[x:NIL]
R[u:v] ⇐ if null[u] then v
else R[cdr[u]; cons[car[u]:v]]

はconsを$O(n)$呼ぶだけである。このようなプログラム変換を行った後にマイクロ化した方がはるかに効率のよいオブジェクトが生成できる訳である。プログラム変換・改良の方法として、

①スキーマによる変換法
②変換ルールを組み込んだシステムを構成する方法

がある。②はHeuristicな変換システムとなるが、反復形のような効率のよいものへの変換が困難なときでも、スタックやカウンタを用いたプログラムに変換し、再帰計算を効率化することができ、全てのLISP関数を扱うことができるという利点がある。AIM1.5では①と②を共に用意しつつある。図4にはスキーマによるプログラム変換の一例を示した。これは文献[8]のアルゴリズムを実現した例である。このスキーマで上述のrev(x)や指数的なFibonacciを線形Fibonacciに変換することができる。この例では変換ルーチンはマッチングを試み、可能な全ての置換集合を表示し、同時にその置換が満たすべき公理を示す。公理が満たされれば反復形式またはtail recursionに変換し結果を出力する。

```
  (TRANS)
PROGRAM FILE NAME = REV
****************** PROGRAM ******************
(DE REV (X)
   (C
      (NULL X)
      NIL
      (APPEND (REV (CDR X)) (CONS (CAR X) NIL))))
*********************************************
OK ? (YES/NO)
? YES
TEMPLATE = T1
****************** SCHEMA ******************
(LAMBDA
   ($F $X)
   (C ($A $X) ($B $X) ($H ($D $X) ($F ($E $X)))))
(LAMBDA
   ($F $X)
   (C
      ($A $X)
      ($B $X)
      (PROG
         ($Y)
            (SETQ $X ($E $X))
            (SETQ $Y ($D $X))
            (WHILE
               (NOT ($A $X))
               DO
               (SETQ $X ($E $X))
               (SETQ $Y ($H1 $Y ($D $X))))
            (SETQ $Y ($H $Y ($B $X)))
            (RETURN $Y))))
*********************************************
OK ? (YES/NO)
? YES

"MATCHING SUCCEEDS !"
3. SOLUTIONS EXIST
-----------------------------
   ($D LAMBDA ($X1) (CONS (CAR $X1) NIL))
   ($E LAMBDA ($X1) (CDR $X1))
   ($H LAMBDA ($X1 $X2) (APPEND $X2 $X1))
   ($B LAMBDA ($X1) NIL)
   ($A LAMBDA ($X1) (NULL $X1))
-----------------------------
   ($D LAMBDA ($X1) (CAR $X1))
   ($E LAMBDA ($X1) (CDR $X1))
   ($H LAMBDA ($X1 $X2) (APPEND $X2 (CONS $X1 NIL)))
   ($B LAMBDA ($X1) NIL)
   ($A LAMBDA ($X1) (NULL $X1))
-----------------------------
   ($D LAMBDA ($X1) $X1)
   ($E LAMBDA ($X1) (CDR $X1))
   ($H LAMBDA ($X1 $X2) (APPEND $X2 (CONS (CAR $X1) NIL)))
   ($B LAMBDA ($X1) NIL)
   ($A LAMBDA ($X1) (NULL $X1))
****************** AXIOM ******************
(APPEND (APPEND $Z $Y) $X)  =  (APPEND $Z ($H1 $X $Y))
(APPEND * $X)  =  *
*********************************************
SATISFY ?
? ($H1 LAMBDA ( X Y ) (APPEND Y X))
****************** AXIOM ******************
(APPEND (APPEND $Z $Y) $X)=(APPEND $Z (APPEND $Y $X))
(APPEND * $X)  =  *
*********************************************
SATISFY ?
? YES
****************** RESULT ******************
REV =
(LAMBDA
   ($X)
   (C
      (NULL $X)
      NIL
      (PROG
         ($Y)
            (SETQ $X (CDR $X))
            (SETQ $Y (CONS (CAR $X) NIL))
            (WHILE
               (NOT (NULL $X))
               DO
               (SETQ $X (CDR $X))
               (SETQ
                  $Y
                  (APPEND (CONS (CAR $X) NIL) $Y)))
            (SETQ $Y (APPEND NIL $Y))
            (RETURN $Y))))
*********************************************
"END OF TRANSLATION"
```

図4 プログラム変換の例

文献 ①伊藤・庄内・岸本, 信学会計算機研究会資料(昭57-1) ②伊藤・藤本・田村・岸本, 信学会計算機研究会資料(昭57-5) ③伊藤, 本学会第24回大会B-4 (昭57-3) ④伊藤・庄内, 本学会第24回大会B-5 (昭57-3) ⑤伊藤・岸本他, 本学会第24回大会B-6 (昭57-3) ⑥伊藤・藤本・田村, 本学会第24回大会 (昭57-3) ⑦J. Mc Carthy etal, LISP 1.5 Manual ⑧G. Huet and B. Lang, Acta Inf. 11, 31 (1978) ⑨伊藤, 情報処理, 第19巻10号 (昭53-10)

謝辞 AIM-1.0の評価実験, MS-50上のLISPコンパイラによる評価実験に協力頂いた庄内守君, 藤本正樹君; SL-resolution proverのLISPによる解析に協力頂いた重永哲君に深謝の意を表する。

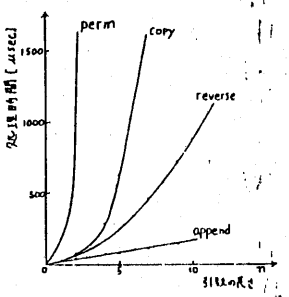
付録. AIM 1.0 の評価実験結果 (単位 μsec)

	MS-50 AIM-1.0			ACOS-1000 EPICS			ACOS-1000 OLISP			ACOS-900 EPICS			ACOS-900 OLISP	
	LS	mp	Inter	Inter	Comp	Sys	Inter	Comp	Sys	Inter	Comp	Sys	Inter	Sys
append	188 (1.0)	176 (0.94)	4,550 (24.2)	619 (3.29)	60 (0.32)	28 (0.15)	989 (5.26)	72 (0.38)	60 (0.32)	1,605 (8.54)	157 (0.84)	88 (0.49)	2,403 (12.8)	163 (0.87)
reverse	270 (1.0)	259 (0.96)	7,615 (28.2)	1,013 (3.75)	94 (0.35)	---	1,653 (6.12)	98 (0.36)	---	2,696 (9.99)	230 (0.85)	---	3,980 (14.7)	---
copy	312 (1.0)	312 (1.0)	8,432 (27.0)	1,269 (4.07)	101 (0.32)	---	1,848 (5.92)	1250 (0.40)	---	3,230 (10.4)	256 (0.82)	---	4,530 (14.5)	---
permu tation	4,478 (1.0)	---	98,600 (22.0)	12,942 (2.89)	1,303 (0.29)	---	20,906 (4.67)	1,548 (0.35)	---	33,246 (7.42)	3,474 (0.76)	---	51,371 (11.5)	---

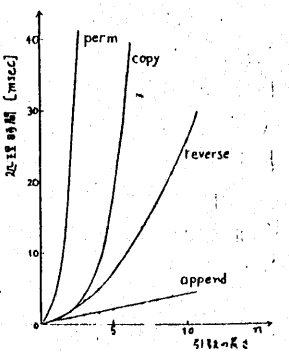
注) LS…LSコーディングによるマイクロ化直接実行. mp…ハンドコーディングによるマイクロ化直接実行. Inter…インタプリタによる実行, Comp…コンパイルされた関数による実行, Sys…システム関数による実行.
append: 第1引数のリスト長10, reverse: リスト長5, copy: 深さ4の完全2進木, permutation: 重複要素なしの長さ4のリスト

図5 AIM 1.0の実験評価

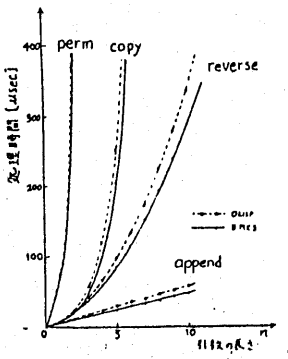
AIM 1.0の評価実験結果を図5と図6に与える. 図5では各関数の実行時間及びLSコーディングを1.0とした時の比を示す. 図6は同じ関数に対し, AIM 1.0 (a,b), 大型計算機Acos 1000上のLISP処理系 (c,d), ミニコンMS 50上のLISPコンパイラ (e) において, 引数のリスト長をかえて, 実行時間の測定を行なった結果を示す.



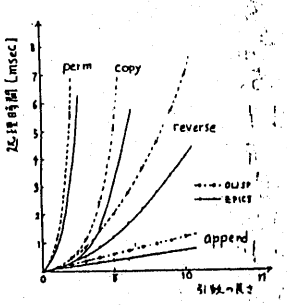
(a) AIM 1.0 LSコーディング



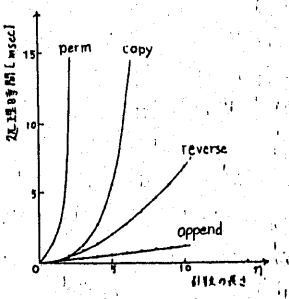
(b) AIM 1.0 インタプリタ実行



(c) 大型計算機 コンパイルされた関数の実行



(d) 大型計算機 インタプリタ実行



(e) ミニコン上の機械語 LISP コンパイラの実行

図6 引数長を変化させての実験