19. HLISPのImplementationと使用経験

後藤英一，寺島元章，金田康正（東京大学・理学部）

ＨＬＩＳＰ（Hash－LISP）はＬＩＳＰ1.6にHash 符号を利用する連想機構を追加したＬＩＳＰであり，追加機構を使わない通常のＬＩＳＰプログラムはほとんどものがそのまま実行できるようになっている。ＨＬＩＳＰに関し，１９７４年夏のシンポジウム以降に得られた結果と計画について述べる。

1. Ｆ－２３０／７５（理研），Ｈ－８８００／８７００（東京大学大型計算機センター）にimplementされているＨＬＩＳＰの下で作動する，数式処理system HLISP—REDUCEの動作状況については学会で発表した。[6)]

2. Garbage Collector（GBC）と仮想記憶化機構の改良については学会で発表した。[7)] 現在のＨＬＩＳＰではFree－storageのrelocationは行なわず，回収された廃品（garbage）はlistにつなげて使っている。それは，システム全体として速度が最も速いからである。しかしながら，１語に２個以上のpointerを含むblockの様なdata構造を導入する可能性とpage方式の仮想記憶を主記憶に使った場合のdataの局所性の保存（ＭＩＴのＭＡＣＬＩＳＰ[14)]はこの例）などに備えてcompactifying GBCについての下記のテストと考察を行なった。compactifying[11)]とは，記憶装置のaddressの順序を変えることなく，途中の廃品の部分を除去して詰めることを言い，compacting[12), 13)]とは，listのcdr部を取り去った一連のblockにすることを一般にさしている。Wegbreit[11)]も示唆しているbinary treeを利用する方法を実際にin core中の廃品になる部分にrelocation用の数を，高さのほぼ等しいbinary tree（Knuth[10)]の言うheap）にしてrelocationの速度を速くした。廃品がＭケ所に飛々に出来た場合，$\log_2 M$に比例する時間でrelocationが出来ることを確認した。我々のalgorifhmでは，binary treeは，その枝のcar cdr部のpointerの情報のみならず，その枝のおかれているaddress（廃品のblockを代表する）も利用したものである。これを利用しないと，廃品が１cellづつ飛々に出来た場合には，relocation用の情報を入れる場所が不足する。

Hansenのcompacting GBC[13)]は，recursiveなalgorithmであるのに対して，Cheney は論文[12)]の表題にnonrecursiveとうたっている。これは，stackを使わないということが大きな利点であるという一般に考えられていることを示している。しかし，そのCheneyのalgorithmは，stackのかわりに，本質的にはqueueを用いたものである。１個のqueueをもつ機械でTuring machineをシミュレートできるのに対して，１個のstackをもつオートマトンでは，それが不可能であることをみても，queueの方が強力な手段である。したがって，機構も一般に複雑になる。この様に，より強力で複雑な機構を使っているのにかかわらず，stackを使わないと言うことをうたうのは，当を得ていない。我々は，virtual tape，virtual headというdata構造の重要性を指摘し，その実現を研究[8) 9)]している。それによるとstackは１tape １headalgorithmで実現できる機構である。これに対し，Cheneyのalgorithmに必要なものは，queueのために１tape ２headsを加えてさらにtapeの先頭をさすhead １個とである。るものである。

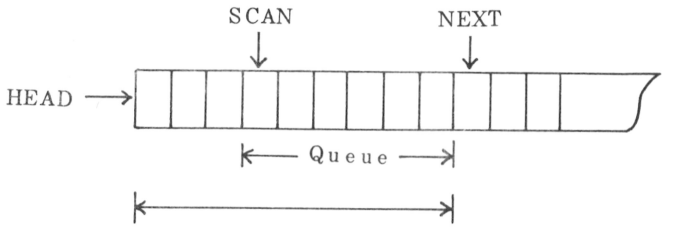

Fig. 1 Virtual tape Virtual head によるCheney の Algorithm のQueueのシミュレート

3. 佐々（東工大）は，block，virtual tape 等の新しいdata typeをＨＬＩＳＰに導入することを検討中であり，板野，Mateev（東大・理）らによって，virtual tape の hard ware 化も検討されている。[8) 9)]

4. ＨＬＩＳＰの応用として最初に得られた，実際に意味のある結果として，佐々木（理研）によるFeynmangraphの個数の計算がある。（計画は夏のシンポジウムに発表[1)]。結果は理論物理学関係誌に発表予定。）

5. ＨＬＩＳＰ system の portability の実現。

１９７４年１０月２１日現在，Ｈ－５０２０（東大情報科学研究施設），Ｆ－２３０／４５Ｓ（東工大・情報科学科），Ｆ－２３０／６０（北大・大型計算機センター），Ｈ－８８００／８７００（東大・大型計算機センター）の各計算機の下でＨＬＩＳＰは作動しており，Ｎ－２２００／７００（東北大・大型計算機センター）に対してはimplementationの作業中である。

6. 学生のＬＩＳＰ実習として９月より，ＨＬＩＳＰ（Ｈ－５０２０にimplementされている）の使用を開始したが，実習に使用することによって，system のbugsが多数とれた。現在ＦＯＲＴＲＨＮで書いたprivate monitor の下で脱線〔ＦＯＲＴＲＡＮのrun － time －errov 例えは，invalid computed go to index，drum area over，core 上のarrayのillegal addressing，等のエラーが生ずること。〕せずに作動している。

7. 応用関数として作成したＳＹＮＴＡＸ（syntax を checkする関数，作成者玉木）をＤＥＦＩＮＥの代りに使用することによって，学生job の throughput は大幅に改善され，指導が非常に楽になった。ＳＹＮＴＡＸは関数の定義におけるエラーチェック（ＦＯＲＴＲＡＮでいえはdeck 毎のエラーチェック）のみならず，関数の相互の呼びあいにおける引数の数のチェックを行なっているために，プログラムの作成上きわめて有効な関数である。（Fig. 2）ＳＹＮＴＡＸを今後さらに強化し，functional argument，ＦＥＸＰＲ等の変数束縛の大部分の問題を，ＳＹＮＴＡＸの責任で解決するようにしたい。この方法は，実行時に特別なbinding mechnnismを利用する方法に比べて速度を向上させる。具体的には，＊で始まる変数名は汎関数のλ変数に専用する，という簡単なものである。これによって実際に現れる大多数のＬＩＳＰプログラムについて，関数引数の使用に伴う変数束縛の混乱の問題が解決される。例えはリストＬ中で条件Ｐが真となる要素のみを選択的に残す汎関数ＭＡＰＫＥＥＰ

```
I   SYNTAX (2)
N          1 1

N
I   (FACTORIAL (LAMBDA (N)
N   1          2       3 3

N
I                  (COND ((LESSP N 2) N)
N                  3     45         5  4

N
I                        (T (TIMES N (FACTORIAL (SUB1 N))))))))))))))
N                        4  5        6           7      7654321000000

N
I   (COMB (LAMBDA (N M)
N   1     2       3   3

N
I               (DIVIDE (FACORIAL N) (FACTORIAL N N-M))))))))))))))))
N               3       4            4 4                 4321000000000000

N
I   END
N

N
P   NO SYNTACTIC ERROR IN FACTORIAL
N
P  5  *WARNING* UNDEFINED FUNCTION FACORIAL
N
P  6  2 ARG TO 1 ARG FUNCTION FACTORIAL
N
P   ==THESE ERRORS FOUND IN COMB
N
     (1   COMB (2   LAMBDA (3   N M) (4   DIVIDE (5   FACORIAL N)
N    1         2           3       3 3            4                4
P  (6   FACTORIAL N N-M))))
N  4                   4321
P  SKIP TO NEXT HLISP JOB
N
```

= means the value of evalquote doublet.

I means the copy of input card.

P means the output by pseudo-function print.

N means parentheses nesting level count.

Fig.2 Example of SYNTAX

は次のように定義するようにする。

```
SYNTAX(2)
 (MAPKEEP(LAMBDA(*L *P)(COND((NULL *L)NIL))
          ((*P(CAR *L))(CONS(CAR *L)
                            (MAPKEEP(CDR *L) *P)))
          (T(MAPKEEP(CDR *L)*P)))))))
END
```

SYNTAXは*Pの出現によりMAPKEEPは汎関数であると判定する。汎関数の定義式中のλ変数が*L，*Pのように*で始まっていなければ，SYNTAXはwarning を出すとともに，これを*付変数（例えはL，P⇒ *L，*P）に自動的に変更するようにする。逆に汎関数の定義式以外の場所に*付変数を使った場合には，SYNTAXはwarning のmenageを出すようにする。これはSYNTAXをMode 2の状態で働かせた場合で，*付変数への変換は行なわないMode を持つようにする。

FEXPRの場合には上と全く同様にして／に始まる変数名をFEXPRの専用にするようにする。

このように特別な記号で始まる名前を特定の目的に専用する方法はすでに広く使われており，例えはMACSYMAではuser の与えた名前にはシステム内部では$を頭に付けて混乱を回避している。FORTRANにおけるI，J，K，L，M，Nの6文字で始まる変数を整数視することも類似の例といえよう。

8. 現在S式を，実行速度が多少はやい，編集されたS式に変換するSCOMPILer を作成して，性能試験を行なっている。さらに，LISPで書かれたProgramme をMachine Independent なFORTRAN Programme に変換するFCOMPILer のLISP Programme）の作成を検討中である。まず実例として，夏のシンポジウムにおける課題の一つ，SEQUENCE（Sort Programme）を手でFCOMPILe してみた結果，実行速度が10～15倍になった。これはCompiler の必要性を如実に示すものである。ただし我々は現在，我が国における大型計算機にすべてCompatible を Systen を作成することに目標をおいているので，Compiler は普通のLISP Compiler のようなmachine dependentでは困る。それ故，machine independent（objectがFORTRANであり，しかも能率が良いもの）なCompiler をいかにして作るかを検討している段階である。

9. HLISPに使われているHash CodingにはHash table からの削除が非常に多く，これを能率よく行なう方法についての理論的解析が完了した。（後藤，群司 未発表）この結果を利用して，Hash coding 機構の高速化を検討している。

10. 関数の定義については仮想化の実現により，Free — storage の領域の数倍の大きさのLISP Programme を実行させ得ることはすでに実証されている。data については，補助記憶を直接操作する命令を使うことなく，仮想化によってデータ領域をふやす例題として，箱入り娘 puzzleを試みている。

11. F－230／75，H－8800／8700のようなキャッシュ・レジスターを持つ計算機に対して，実行速度を上げる試みている。具体的にはCAR部とCDR部，重要な system 変数，等を隣接した記憶位置におくよう

にする。

12. ＬＩＳＰのＰＲＯＧ feature におけるＧＯのevalvationには，Hash codingによる連想的検索を行なっているが，その方法を改良することにより，計算型ＧＯを高速化した。
（計算型ＧＯはＬＩＳＰ1.6の文法に従った。）

13. Proparty － list 関係の命令には連想機能を用いて，高速化を図った。

14. ＨＬＩＳＰのsystem 内部では連想機能を利用して，１２，１３のように実行速度を上げているが，記号処理，数式処理，人工知能等の問題に対して，直接連想機能を導入し，algorithm の高速化を図ることはこれからの残された研究課題の一つである。

参考文献

1）１９７４年プログラミングシンポジウム（夏）報告集〝記号処理と数式処理〟
2）J. McCarthy etal. :〝LISP 1.5 Programmer's manual〟, MIT Press (1962)
3）L. H. Quam :〝Stanford LISP 1.6 Manual〟Stanford Artificial Intelligence Laboratory Operating Note №. 28.4
4）A. C. Hearn :〝REDUCE—2 USER'S MANUAL〟2—rd ed. UTAH Univ. Mar. 1973
5）後藤英一：「ＬＩＳＰ入門」bit，共立出版，1974，1月号より連載中。
6）金田康正，後藤英一：「数式処理言語ＲＥＤＵＣＥ－２」情報処理学会第１５回大会予稿集
7）寺島元章，後藤英一：「ＨＬＩＳＰの仮想記憶とGarbage Collectorの改良」
同上
8）佐々政考，後藤英一：「Block と Virtual tape を持つＬＩＳＰシステム」
同上
9）E. Goto, K. Itano and L. Mateev :〝Propositions for Virtual Multihead Multitape Processor〟同上
10）D. E. Knuth :〝The Art of Computer Programming〟vol 1, vol 3, Addison－Wesely (1973, 1968)
11）B. Wegbreit :〝A Generalised Compactifying Garbage Collector〟The Computer Journal vol 15 no 3 (1971)
12）C. J. Cheney :〝A Nonrecursive List Compacting Algorifhm〟CACM vol. 13 no. 11 (1970)
13）W. J. Hansen :〝Compact List Representation : Definition, Garbage collection and System Implementation.〟CACM vol 12 no 9 (1969)
14）R. R. Fenichel and J. C. Yochelson :〝A LISP Garbage Collector for Virtual Memory Computer Systems〟CACM vol 12 no 11 (1969)