研究発表

コンパクトな処理系が可能な連想情報モデルについて

井田昌之*

経営管理における重要な課題の一つとして多様化する事務情報の効果的な管理運用がある．本論文では連想子と称する基本単位に基づく連想情報モデルについて考察を行ない，さらに本モデルの電子計算機上でのコンパクトな実現方法を示し，小規模ないしは実験的なデータベースの構築手法を導いた．また筆者らが8ビットマイクロコンピュータを用いて作成したLispマシンALPS／I上で実現された連想情報モデルとその応用例も示した．連想子は単一の記号情報ないしそれらの構造体間の写像を，ハッシングを利用して高速参照性を与えた記憶単位であり，連想子をリンクリストとして結合することにより任意の構造を実現できる．

A Proposal of an Associative Information Model and Its Compact Implementation on Computer

Masayuki IDA

To efficiently manage business information which is heterogeneous is one of the important tasks in business management field. Database has growing popularity as a solution to it. But it is expensive for a small application or an experimental use. This paper presents an Associative Information Model which is based on associator. Database-like features can be compactly obtained with this model. Associators are the units to storing mappings among simple symbolic informations or their structures with hash function for quickness. Furthermore, by linking associators as linked list, arbitrary information structure can be built on them. This mechanism can be commonly used. And it is easy to implement it. As an example, this model has been built on the Lisp machine named ALPS/I, our original computer, and some applications are shown in this paper.

1. はじめに

企業体内における事務情報は複雑をきわめ，その管理と運用は経営管理における重要な課題の一つとなってきている．しかし，その推進役であるEDPは多分に手工業的なものであった．

近年ようやくソフトウェア工学とよばれる分野も形成されつつあり，ソフトウェア生産体制やおのおのプログラム・データ構造に関する研究〔1〕～〔6〕も実際の適用が試みられはじめている．

現在普及しつつあるデータベース〔4〕は，そのなかで対象となる事務システムで使用する情報自身に着眼しそれを統一的に管理する．その結果個別的なプログラム作成のむだを省くことが可能となり，EDP部門のシステム改善を支援できる．

データベース的な考え方では多少複雑であっても情報の高速参照能力やその整合性・完備性への対処をシステム側にもたせ，情報の検索・維持・格納等に対する要求を簡潔化しようとする．

しかし既存のデータベースシステム自身は，大容量のディスクメモリを対象とする大規模なプログラムであり，小規模な応用ないしは実験的な導入においてはデータベースは高価すぎるといえよう．

データベースの本質を事務情報の効果的な蓄積と参照機構の提供とみなしたとき，これらを比較的小さいプログラムで実現できるならばその適用分野も広げられよう．

本研究は以上のような認識に立ち，連想情報モデルを提示し，かつその簡潔（コンパクト）な実現アルゴリズムを示す．提示する連想情報モデルは一連の情報に対して有向グラフ構造を用いて考察する場合および個々の要素の検索効率も高めたい場合に有効である．またこれは適用分野および使用機種に依存せず小さく作成できる（例として筆者らがハード・ソフトともに自作したALPS／I〔7〕上に実現したが，約500ステップ程度の大きさで済んでいる）．

2.より順に，1)情報構造がもつべき性質について，2) 1) を満たす構造についての導入，3)連想情報モデルの定義とその実現アルゴリズム，4)筆者らの計算機ALPS／Iにおける実現方法，5)応用例について述べる．

2. 情報構造がもつべき諸性質

モデルを提示する前に次のような諸性質を考え判断の材料とする．順編成においてはこれらいずれの性質も存在しない．

*青山学院大学（Aoyama Gakuin University）
昭和52年度春季研究発表会にて発表
受付：昭和53年7月26日

①**フィールド（項目）識別性**：レコード中の項目を正しく識別できる能力.

②**構造の柔軟性**：フィールドの新規追加・削除，大きさ・内容の変更に対する対処能力．これらの操作が情報構造全体に影響せずにできることが望ましい.

③**情報の任意性**：各フィールド中の情報の属性の任意性．数・文字列・集合・グラフなどを受け入れることが可能か否か．定性的情報も簡単に組み込めることが望ましい.

④**参照の高速性**：レコード中の各フィールドは極力高速にかつ簡潔（余分なオーバーヘッドなし）に参照できることが望ましい.

3. リンク構成と連想構成による情報の構造化

2.でとりあげた性質すべてを満足するモデルおよび処理系の作成は高価であると思われており，大規模なデータベースにおいてのみ考察されてきた〔4〕．しかし，それらは必ずしも高価ではなく，以下に示す二つの構成方法の併用により上述の性質を満足することが可能である.

3.1 リンクリスト構成と構造の柔軟性

リンクリスト構成は理論的には有向グラフ$\vec{G}=(A, N, \phi)$に帰着する（**図1**）．これは明示的にポインタを用いて情報の関係を規定する.

その定式化はGuha〔14〕などにより試みられている.

リンクリスト構成はポインタのつけかえにより情報間の関係を変更できるので構造の柔軟性がある.

3.2 フィールド識別性と情報の任意性の付加

J. McCarthyにより開発されたLisp〔10〕はリンクリストを中心とする記号処理（文字列として表わされる非数値情報の処理）用言語であり，他の計算機用言語とは著しく異なった特徴をもっている．人工知能関連プログラムのほとんどはLisp言語を用いている．その特徴は次のようなものである.

① 処理手続きとデーターの区別がなく，すべて二進のリンクリストで統一され表現される（**図2**）.

② リンクリストの基本単位はセルとよばれ，すべて同一の大きさで扱われる．さらにリンクリスト，数，文字列などはそれらすべてが格納された場所へのポインタで統一的に処理される．この結果，種々の属性をもつ情報の扱いが容易となっている．属性の判別はポインタに与えられた公理的順序を用いて行なう〔7〕.

③ 処理手続きはすべてリンクリストに対して働き，リンクリストを値とする関数として表わされる.

④ 関数の再帰的定義が許され，またシステム組込み

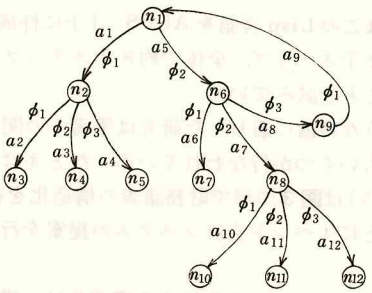

図1 リンクリスト構成$\vec{G}=(A, N, \phi)$
（ただしA：アーク（ポインタ），
N：ノード，ϕ：リンクラベル）

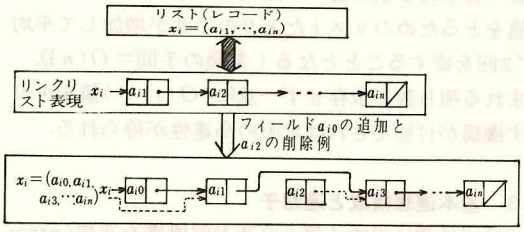

図2 ファイル$X=\{x_1, \cdots, x_i, \cdots, x_m\}$
のLisp型リンクリスト表現

の基本関数の有用性が高い．使用者は他の言語の場合と比べてかなり簡潔に（小さく）処理手順を記述できる．その結果処理全体の見通しがよくなる．たとえば**図2**に示したx_iの集合$x=\{x_1, \cdots, x_m\}$（いわゆるファイルに相当）に対する変更は次の定義を実行させればよい.

additem〔x；a〕＝mapcar〔x；λ〔〔i〕；progn〔
i：＝nconc〔list〔car〔a〕；car〔i〕〕；cddr
〔i〕〕；a：＝cdr〔a〕；i〕〕〕
（$a=\{a_{i0}, \cdots, a_{im}\}$とする）

説明：ファイルxのすべての要素x_iに次のことを行なえ（mapcar〔x；λ〔〔i〕⋯〕〕）

① a_{i0}を追加する（list〔car〔a〕；car〔i〕〕）

② a_{i2}を削除する（nconc〔list〔⋯〕；cddr
〔i〕〕）

③ 次への準備処理

また項目をすべて数値としたとき累計表$y=\{y_1, \cdots, y_n\}$，$y_j=\sum a_{ij}$を更新する手続きは次のようになる.

summerize〔x；y〕＝mapcar〔y；λ〔〔j〕；progn
〔x：＝mapcar〔x；λ〔〔i〕；progn〔k：＝sum
〔k；car〔i〕〕；cdr〔i〕〕〕〕；j〕〕〕

説明：累計表yのすべての要素y_jに次のことを行なえ（mapcar〔y；λ〔〔j〕⋯〕〕）

① ファイルxの各要素x_iに対してa_{ij}をy_jにたし込む（x：＝mapcar〔x；λ〔〔i〕⋯〕〕）

② 次への準備処理

筆者らはこのLisp言語をALPS／I上に作成し(処理系は約2千ステップ，全体で約8千ステップ)実用に供することを試みている．

このような特徴に着目した研究は筆者らの関連分野においてもいくつか行なわれている．たとえばLieberman〔5〕は**図3**の形で財務諸表の構造化を行ない，これをもとにイベント会計システムの提案を行なっている．

しかしリンクリストのみによる構造化は，構造の柔軟性などの性質は満足するが各フィールドの参照はリストをたどっていかねばならず，本質的に低速である．つまり含まれる項目数nに比例して求めるフィールドの値をとるためのリストたどりの手間が増加して平均$n/2$回を要することとなる(参照の手間$=O(n)$)．含まれる項目数に依存せず一意に($O(1)$で)値をとりだす機構が付加できれば参照の高速性が得られる．

3.3 基本連想構成と連想子

本論では鍵に対する値の高速参照機構を連想(association)とよぶ．**図4**に示すように連想は集合間の写像(mapping)の実現といえる．この写像は通常全単射的(bijective)に作成するが，応用によっては全射性も単射性も要しない場合もある．

(定義1) 基本連想構成をKからVへの写像f：$K\rightarrow V$で定義する．ここでK：鍵集合，V：値集合とする．

(定義2) 写像関係をもつ$k\in K$, $v\in V$の対(k,v)を連想子(associator)とよび，構成の基本要素とする．

さらにkおよびvに対して文(文字列および文字列のリンクリスト表現)を直接用いられるように計算機上に実現する．このことによりたとえば「子供」→「かわいい」，「来月の生産可能台数」→「1,500台以下」などの情報をそのままの形で計算機上に格納できる．

連想子は(k_i, v_i)を$2\times n$の表にいれることにより擬似的に格納できる．単純な記号表はこの好例である．またLisp 1.5〔10〕におけるa-list(変数束縛リスト)は連想子のリストによる擬似表現の例である．これらの表現における実際上の問題はPfaltz〔11〕などにより報告されているが，これらの方法はあくまで擬似的であり連想子の特徴となるべき高速参照能力はない．

これに対し高速性をもつ次のような機構が考えられる．Vを配列として保持し非数値である鍵に1対1に対応する数値を計算する．そしてその数を配列への添字として直接使用することにより連想子数nに関係なく高速に(1回で)参照できる(**4.2**で詳述する)．

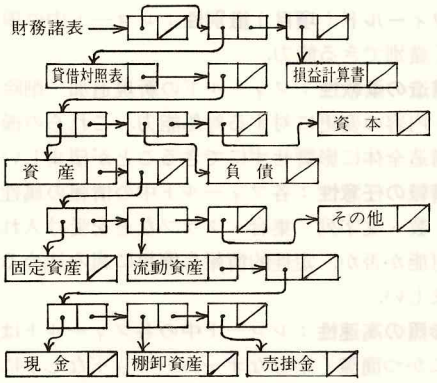

図3 Lisp型リンクリストによる財務諸表構造化の一例〔5〕

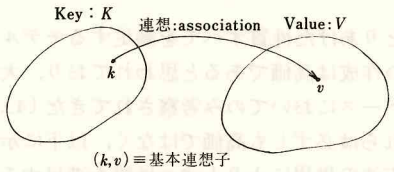

図4 連想構成

4. 連想情報モデルとその実現

4.1 連想情報モデル

連想の定義域Kを吟味することによりいわゆるファイルレコードを連想子として保持することが可能となる．

(定義3) 連想情報モデルを基本連想構成の拡張として$A\times O\rightarrow V$と定義する．

ここでA：属性(Attribute)，O：対象(Object)，V：値(Value)の集合とする．

(定義4) 連想子とは(a, o, v)のことをいう．ここで$a\in A$, $o\in O$, $v\in V$とする．

たとえばある従業員に対する入社年度ないし基本給の格納は連想の概念を用いて次のように表現する．

「青山太郎」$\xrightarrow{\text{「入社年度」}}$「1970年」，

「青山太郎」$\xrightarrow{\text{「基本給」}}$「12万円」

$A=\{$入社年度，基本給$\}$，$O=\{$青山太郎$\}$，

$V=\{$1970年，12万円$\}$

このように$a(o)=v$の関係を保つようなa, oを定める．このとき各値の検索要求は

(入社年度, 青山太郎, ?), (基本給, 青山太郎, ?)

と表現でき，(a, o)を鍵とする連想高速検索により値をうることができる(実際の計算機上の実現においても，それらはコード化されず記号のままで格納される)．

4.2 ハッシュ技法を利用した連想情報モデルの実現

連想子をハッシュ技法を利用し実現することにより，高速参照能力のある連想情報モデルを構成できる．

連想子の値を求める手続き find (attribute, object, value) を次のように定義する．

```
procedure find (attribute, object, value ) ;
function addr ; integer i ;
sparse-array vtable〔m〕" m>=n" ;
begin i :=addr〔attribute, object〕;
  if i=NIL then value :=NIL
    else value :=vtable〔i〕
end ;
```

vtable を疎な配列とする．位置ぎめ関数 addr は非数値である文字列からハッシュ技法を用いて数を対応させる関数とする．ハッシュ技法はその名のとおり文字列を「切り刻みまぜ合わせる」ことにより，定められた区間内の整数に対応させる技法である．その整数はハッシュ領域とよばれる表のアドレスとして用いられる．概要は Morris〔13〕などにより報告されている．ハッシュ関数としては高速性を重んじ文字コードに対する単純な四則だけで済ませることが実用上一般的である (付録Ⅱ)．このため文字列と整数との間の写像の単射性は保証されず関数値の衝突が生じる．衝突の対策としてはチェイン法，リハッシュ法などが知られているが，その高速性から後者を採用する〔7〕．

リハッシュ法は値の衝突時にその値をもとに再ハッシュを行なうものである．衝突の確率はハッシュ領域中の連想子総数の比率 (ロードファクタ ρ) により定められる〔13〕．筆者らの方法による平均探索回数 (参照値が発見されるまでのハッシュ回数 prob の平均 E (prob)) は

$$E(\text{prob}) = -\frac{1}{\rho}\log(1-\rho)$$

によって与えられ，ρ が80%であっても約2回で済むことが確率的にもまた実測上もわかっている．たとえば 10,000 個の情報も 15,000 語ほどのハッシュ領域に保持させれば各項目を他の手法と比較して高速に平均約2回の探索で参照でき，それは要素数に依存しない．またハッシュ領域は少量 (数万程度まで) ならば主記憶上に，多量ならばディスク等の補助記憶にわりあてることにより，このモデルの定量的制限をはずすことができる．

なお関数 addr は次のように定義できる．

```
function procedure addr〔a, o〕;
sparse-array hash-area〔m〕;
begin i :=hash〔a〕+hash〔o〕;
  while occupied〔hash-area〔i〕〕do
    begin if equivalent-key〔hash-area〔i〕;
      a ; o〕 then return〔i〕 ;
      i :=rehash〔i〕
    end ;
      return〔NIL〕 end ;
```

occupied〔x〕はハッシュ領域中の要素 x に連想子がはいっているなら真，さもなければ偽を返す関数，hash は付録Ⅱに示す関数，equivalent-key〔x ; y ; z〕は要素 x 中の鍵が (y, z) と等しいならば真，さもなければ偽を返す関数とする．この addr 関数は find 手続中で使用されるだけでなく，情報の登録時の手続においても使用される．

ALPS／I においてこのプログラム部分は約 200 ステップ程度で実現できている (他の計算機でもその程度で作成できる)．さらに一意性を必要とする情報をすべて連想子として扱う．これによりシステム全体の統一性が保たれコンパクトに実現できる．ALPS／I では，1) いわゆる変数識別名称の文字列を鍵とする素記号連想子とよばれる基本連想子，2) ハッシュ化配列連想子とよばれる連想情報モデルの中核をなす連想子，および3) 連想計算連想子〔付録Ⅰ〕の3種がある〔7〕．

4.3 ハッシュ化配列機構

連想情報モデルにおける連想機構は ALPS／I ではハッシュ化配列 (hashed array) として組み込まれている．「属性」をハッシュ化配列名とよび，「対象」はその引数 (添字) として表記する．これは疎な配列の実現方法を意識した名称である (たとえば 10,000×10,000の疎な行列も連想子表現により実際に値をいれられた要素分だけの記憶域の使用で済む)．ハッシュ化配列は連想子であるのでその添字 (「対象」) はいうまでもなく数字，文字列，リンクリストのいずれであってもよい．

ALPS／I に用意された基本機能は次のとおりである．

- array〔$a_1 \cdots a_n$〕　a_i をハッシュ化配列名として宣言
- dearray〔a〕　ハッシュ化配列名 a とそれを属性部にもつ連想子をすべて除去
- seta〔a ; o ; v〕　属性部に a，対象部に o，値部に v をもつ連想子の作成
- deleta〔a ; o〕　属性部に a，対象部に o をもつ連想子を除去
- a〔o〕　鍵に (a, o) をもつ連想子の値 v を返す

ファイルをハッシュ化配列を用いて格納することにより各項目の高速参照が可能となる．さらにその連想

子をリンクリストで結合することにより連想子情報を階層的に構造化できる.

たとえば(学生氏名, 英語得点, 数学得点)からなる成績ファイルの宣言は,

array〔ENGLISH ; MATHEMATICS〕

ある学生の英語得点のセットは

seta〔ENGLISH ; name ; grade〕

その参照は

ENGLISH〔name〕

を実行させればよい.

5. 連想情報モデルの応用例

文献〔6〕に示されている時間給制による従業員給与(payroll)システムを一例にとり, それに対してモデルを適用し有効性を確認する.

対象となるファイルは従業員名簿としてまとめられる. 各員に対しては時間給および日々の労働時間が入れられる. **図5**にその構成を示す. 各項目はハッシュ化配列連想子として格納される.

なおこのシステムは連想子の応用を例示するために極力小さく作成されている. これをさらに現実的にするには, いくつかの項目の追加といくつかの手続きの追加が必要であるが本質的な変更は不要である.

図5の構成は文字列を添字とする二次元の表といえるが, 要素はすべて満たされなくてもよく疎な入力でも記憶効率がよい.

このシステムの単位操作と実際のコマンドとの対応を**表1**に示す. この単位操作に基づく会話型処理系のプログラムリスト(初期化部分を除く)を**図6**に, その実行例を**図7**に示す.

またこの会話型システムに対して「仮想データ」〔4〕を導入できる. 連想計算機構〔付録Ⅰ〕を指定することにより「一度手続き的に計算した値は覚えておく」ことができる. これにより使用者は計算手続きを書くだけでよい. 二次的なデータの格納はシステムにより自動的に行なわれ, 値の2度目以降の計算は実際に生ぜず連想子検索に自動的に切りかえられる.たとえば「1月1日に労働を行なった者のリストアップ」が必要ならば, この手続きを連想計算指定すると2度目以降の参照は1度目に自動的に作成された連想子の検索に自動的におきかえられ手続き実行の手間が省かれる.

表1 payrollシステムにおける単位操作

	単位操作	コマンド	機能
1	従業員の追加	ADD-EMP	従業員名簿への追加と時間給の設定
2	従業員の削除	REMOVE-EMP	従業員名簿からの削除
3	新日付の設定	NEWDATE	新しい日付を追加
4	労働時間の格納	SET	(日付, 作業員名, 労働時間)の連想子の作成
5	連想子の参照	DISPLAY	指定された連想子の表示
6	連想子の除去	ERASE	設定ミス等に対応して連想子を除去
7	給与表の作成	PRINT	各従業員の給与の印刷

また連想子は, 1) 数式処理言語ALPS-Reduce〔9〕におけるフラグ・属性の処理. 2) 箱入娘パズルの解法を例とするbreadth-first型探索における帰路の処理〔8〕などに利用されている.

6. ま と め

連想情報モデルの提示とそのALPS/Ⅰ上での実現例について述べた.

連想は文字列で示される記号からすでに格納された値を高速に参照することから発している. 筆者の示した連想方式は二つの集合の直積空間上の点に種々の値を与える機構を簡潔に比較的小さなプログラムで作成できる点に特長がある.

図5 連想子によるpayrollシステムの構成(破線内が連想子で疎にちらばっている)

```
 PROGN(
(ARRAY RATE COMMAND)
(CSETQ DATES NIL)                    初期化部分
(CSETQ EMPLOYES NIL)
(SETAQ COMMAND(QUOTE SET)(QUOTE(SETA(GETITEM)(GETEMP)(GETVALUE))))
(SETAQ COMMAND(QUOTE ERASE)(QUOTE(DELETA(GETITEM)(GETEMP))))
(SETAQ COMMAND(QUOTE DISPLAY)(QUOTE(PROGN(PRINI(EVAL(LIST(GETITEM)(LIST(QUOTE QU
OTE)(GETEMP)))))(PRINT (QUOTE $$$ IS IN IT$)))))
(SETAQ COMMAND(QUOTE NEWDATE)(QUOTE(PROGN(GETDATE)(EVAL(LIST(QUOTE ARRAY)D))(CSE
TQ DATES(CONS D DATES)))))
(SETAQ COMMAND(QUOTE PRINT)(QUOTE(OUTPUT(MAKE-WAGE-LIST))))
(SETAQ COMMAND(QUOTE ADD-EMP)(QUOTE(PROGN(CSETQ EMPLOYES(CONS(GETEMP)EMPLOYES))(
GETRATE))))
(SETAQ COMMAND(QUOTE REMOVE-EMP)(QUOTE(DELETE EMPLOYES(GETEMP))))
(SETAQ COMMAND(QUOTE FIN)(QUOTE(RETURN(QUOTE PAYROLL-END))))


DEFINE((
(MAKE-WAGE-LIST(LAMBDA( )(MAPCAR EMPLOYES(QUOTE(LAMBDA(K)
      (CONS K (MULT(RATE K)(TOTAL-HOURS K))))))))
(TOTAL-HOURS(LAMBDA(X)(PROGN (LOCAL I J) (SETQ I 0)
      (MAPC DATES (QUOTE (LAMBDA(Y)(COND((SETQ J(ERRSET(Y X)T))(SETQ I (SUM I J))
)))))
      )))
(PAYROLL(LAMBDA( )                    ┐
 (PROG  (D COM)                       │
LOOP   (TERPRI)(TERPRI)               │
       (PRINI (QUOTE $$$COMMAND ? $))  │   会話処理部分
       (SETQ COM (READ))              │  (各コマンドごとの手続きも連想子として
       (EVAL (COMMAND COM))           │   初期化部においてセットされている)
       (GO LOOP)                      │
)))                                   ┘
))
```

図6　payroll システム処理系全プログラム

```
EVALQUOTE ENTERED, ARGUMENTS...
PAYROLL NIL

COMMAND ? ADD-EMP
 EMPLOYE-NAME ? TOMY
 RATE ? 5

COMMAND ? ADD-EMP
 EMPLOYE-NAME ? JOHN
 RATE ? 4

COMMAND ? NEWDATE
 DATE ? S53-06-06

COMMAND ? SET
 ITEM ? S53-06-06
 EMPLOYE-NAME ? TOMY
 VALUE ? 6

COMMAND ? NEWDATE
 DATE ? S53-06-07

COMMAND ? SET
 ITEM ? S53-06-07
 EMPLOYE-NAME ? TOMY
 VALUE ? 5

COMMAND ? SET
 ITEM ? S53-06-07
 EMPLOYE-NAME ? JOHN
 VALUE ? 7

COMMAND ? DISPLAY
 ITEM ? RATE
 EMPLOYE-NAME ? TOMY 5 IS IN IT


COMMAND ? PRINT

     WAGE LIST

     JOHN = 2800 YEN
     TOMY = 5500 YEN

TOTAL 2 EMPLOYES


COMMAND ? FIN

END OF EVALQUOTE, VALUE IS..
PAYROLL-END
```

図7　payroll 実行例（コマンドは表1に示す．下線部が入力）

さらに自作のハードウェア ALPS／I のホスト言語として製作した Lisp の上にこの方式を実現することにより，連想子をリンクリストで結合できる．

この連想子とリンクリストとの融合は高速参照能力と構造の柔軟性が兼ね備えられており，現在データベースの分野で話題になりつつあるユニバーサル構造〔15〕に発展する可能性も考えられ，この方面への今後の研究を行なうことが要点であると思われる．

この研究に対して平素ご指導いただいている青山学院大学理工学部間野浩太郎教授，ならびにALPS／I作成に協力を仰いだ同研究室の諸兄に感謝の意を表する．

［付録　Ⅰ］　連想計算機構

連想計算は Goto〔12〕により階乗の計算の高速化を例として提案されている．筆者はこの機構を ALPS／I に組込み実用に供している．連想計算を行なう関数は次の手順により処理される．

①　与えられた実引数に対して関数値がすでに求められているかを判定する．すでに存在すればその値をただちに返す．

②　存在しなければ関数手続きを呼び出し実行する．その値を返すと同時に連想子として格納する．

Goto〔12〕によれば再帰的に構成された n ! の計算の後の m !（$m \leq n$）の計算は実際に行なわれず，ただ1回の連想子検索のみで値を求めることが述べられている．

［付録　Ⅱ］　ハッシュ技法による写像例

'ABCDE' という文字例から区間［0，4095］の点

への写像をなすハッシュ関数addrは次のような手順をとる.

① 'ABCDE' を表わす文字コード4142434445(16進)の折り重ね(folding)→4142+4344+4500=8986を得る.

② 上位4ビットを切りおとし区間[0, 4095]に入れる. →986(16進)すなわち2438(10進)を得る.

この2438が'ABCDE'のハッシュ値となる.この手法はALPS/Iにおいて用いられている.

参考文献

〔1〕 井田昌之:"事務処理プログラム作成の体系化とFAST 1", 日本経営工学会誌, pp. 417-422, Vol. 28, No. 4, (1978)

〔2〕 IBM: "Chief Programmer Teams Principles and Procedures," FSD Report No. FSC 71-5108, (1971)

〔3〕 Baker, F.: "System Quality through Structured Programming," *Proc. of FJCC*, pp. 339-343, Dec., (1972)

〔4〕 Martin, J.: Computer Database Organization, Prentice-Hall, (1975)

〔5〕 Lieberman, A.Z. and Whinston, A.B.: "A Structuring of an Events-Accounting Information System," *Account. Rev.*, pp. 246-258, April, (1975)

〔6〕 Haseman, W.D. and Whinston, A.B.: "Design of Multi-dimensional Accounting System," *ibid.*, pp. 65-79, Jan., (1976)

〔7〕 井田昌之:"マイクロプロセッサを用いたLispマシンALPS/I", 情報処理, pp. 113-121, Vol. 20, No. 2, (1979)

〔8〕 井田昌之:"ALPS/Iの性能評価", 情報処理学会記号処理研究会77-2-(1), (1977)

〔9〕 小林茂男:"ALPS-Reduceのインプリメンテーション", 青山学院大学修士論文, (1977)

〔10〕 McCarthy, J.: Lisp 1.5 Programmer's Manual, MIT Press, (1963)

〔11〕 Pfaltz, J.L.: Computer Data Structure, McGraw-Hill, (1977)

〔12〕 Goto, E.: "Monocopy and Associative Algorithms," Tokyo Univ. Tech. Rep., (1974)

〔13〕 Morris, R.: "Scatter Storage Techniques," *C. of ACM*, pp. 35-38, Vol. 11, No 1, (1968)

〔14〕 Guha, R. and Yeh, R.: "A Formalization and Analysis of Simple List Structures," in R. Yeh ed., Applied Computation Theory, Prentice-Hall, Chapter 5, pp.150-182, (1977)

〔15〕 小林 要:"汎関係データベースモデルと拡張集合プロセッサ", 情報処理学会第19回全国大会, pp. 905-906, (1978)