入れ子構造を許す言語処理で用いる名前表の一構成法とその解析†

中村良三†† 牛島和夫†††

ブロック構造を許すプログラム言語のコンパイラやインタプリタで用いる名前表は，名前の探索だけでなく，格納，削除が混在する動的な状況に対処する必要がある．見出し探索法の一つである分散記憶法では，探索に要する路の長さは名前の多少に関わらず表占有率によって決まり，平均的に数回の探索回数でよい．したがって，分散記憶法は探索効率の点から名前表の処理に有効な方法である．しかし，この方法では，ブロックの開閉に伴う名前の格納，削除が混在する状況に対しては効率がよくない．本稿では，名前の格納，削除が混在する動的な状況に対処できるように，名前の探索に有効な分散記憶法と入れ子構造の処理に簡便なスタック法を組み合わせたワンパス・コンパイラ向きの名前表の構成法を提示し，この名前表が入れ子構造を許すプログラム言語 Pascal での名前の処理に有効であることを具体的に示す．次に，この名前表を用いた場合の探索路長をブロック間での名前の参照頻度を考慮したもとで解析し，探索路長の評価式を導出する．そして，この評価式を用いて，探索路長を評価するとともに単純にスタックのみで名前表を構成した場合の探索路長とを具体的な数値例を用いて比較し，ここで提示した名前表を用いれば，探索路長が著しく短くなることを示している．

1. まえがき

コンパイラやインタプリタにおける名前表の構成には従来種々の方法が提案されている．分散記憶法を用いる方法では，探索路長は名前の多少に関わらず表占有率によって決まり，平均的には数回の探索回数で十分であり，探索効率がよい．しかし，ブロックの入れ子構造を許す Pascal 型の言語では，名前の操作は名前の有効範囲規則に従った名前の探索だけでなく，ブロックの開設，閉鎖に伴う名前の格納，削除が混在する．したがって，このような動的状況に対応する名前表の構成に分散記憶法だけで対処することは，名前を削除する際に効率がよくない．一方，単純に名前表をスタックのみで構成すると，名前を削除する際の効率はよいが，しかし，多重入れ子構造で名前が広域的に使用される場合，名前の探索効率はよくない．

本稿では，名前の探索に有効な分散記憶法と入れ子構造の処理に簡便なスタック法を組み合わせたワンパス・コンパイラ向きの名前表の構成法を提示し，この名前表が Pascal 型言語での名前の処理に有効であることを具体的に示す．次に，この名前表を用いた場合の探索路長をブロック間での名前の参照頻度を考慮したもとで解析し，探索路長の評価式を導出する．この評価式を用いて，単純にスタックのみで名前表を構成した場合とこの場合との探索路長を具体的な数値例を用いて比較し，提示した名前表を用いれば，探索路長が著しく短くなることを示す．

2. ブロック構造と名前の有効範囲

Pascal 型言語のプログラムの構造は宣言・定義部（以下，宣言定義部と書く）と実行文部に大別される．この宣言定義部の中にある手続き・関数宣言部はまた宣言定義部と実行文部から構成される入れ子構造になっている．このような入れ子構造では，宣言定義された名前の有効範囲はどこかということが問題になる．Pascal ではブロックの先頭で宣言された名前はそのブロックの区域の内側で有効となる．ここで，ブロックの先頭で宣言定義された名前とは，

1) 仮パラメータ部の名前
2) 定数定義，型定義，変数宣言部に現れた名前
3) 手続き宣言，関数宣言の頭部に現れる手続き名および関数名

となる．

ここで，ブロックの区域とは，仮パラメータ部，宣言定義部，実行文部を合わせたものである．

このとき，名前がどのブロックで宣言定義されたものかということは，ブロックを内から外に探してゆくことで見つけられる．上述したブロックの区域の入れ子構造に関する名前の有効範囲規則は定数名，型名，

† An Organization of Symbol Table for a Language with a Nested Block Structure and Its Analysis by RYOZO NAKAMURA (Faculty of Engineering, Kumamoto University) and KAZUO USHIJIMA (Faculty of Engineering, Kyushu University).

†† 熊本大学工学部電子工学科
††† 九州大学工学部情報工学科

変数名，手続き名，関数名あるいは仮パラメータ名に適用される．

一方，レコード変数の内部データを扱うにはレコード変数名のうしろにピリオドとフィールド名を書く．レコード型変数のフィールド名はレコードに関係づけてその有効範囲を規定される．このフィールド名の指定を簡単に書く便利な方法として下記の with 文がある．

with 〈レコード変数〉{,〈レコード変数〉}
　　do 〈文〉;

〈文〉の中ではフィールド名が通常の変数名と同じ取扱をされるとともにフィールド名以外の名前も使用できる．したがって，レコード型のフィールド名を含めた名前の有効範囲規則を考慮しなければならない．

3. 分散記憶法とスタックを併用した名前表の構成

ブロック構造を許す言語の処理で，名前表に分散記憶法を用いる方法はブロック番号と名前を組にしたものを分散関数の引数として取り扱う方法も考えられるが[1]，ブロックを閉鎖する際，このブロックに属する名前の削除を行うときには手間がかかる．

ここで提示する名前表の構成法は，名前の有効範囲規則に従った名前の探索とブロックの閉鎖に伴う名前の削除とを効率よく行えるように，分散記憶法とスタックを併用する．このとき，分散記憶法の衝突処理としては分離連鎖法を用い，分離あふれ領域をリストとして形成する．そして，分散表の上で同じ指標となる名前は同一のリストとして表現し，新しい名前はリストの先頭に格納する．スタックは主としてブロックの開設および閉鎖に伴う問題を処理する．この名前表の構成を**図1**に示す．この方法では，スタックにはブロックの入口の印と名前（たとえば，図1の XYZ）が格納されるリストを指し示す分散表の指標（図1の K）とを格納する．分散表にはリストの先頭を指す番地が格納され，リストの各要素，すなわちセルは名前とその属性のほかに，次の要素を指すポインタからなり，逐次動的に割付けられる．ところで，この名前表はスタックと分散記憶法とを混成（ハイブリッド）して用いるので，以下この方法による名前表を混成法による名前表と称する．

理解を助けるため，はじめに，単純なデータ型からのみなる名前を処理する場合について，この名前表の処理アルゴリズムの概要を示す．

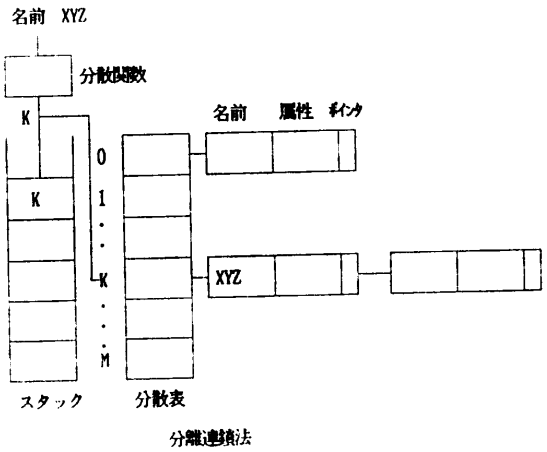

図1 分離連鎖法とスタックを併用した名前表の構成
Fig. 1 The structure of a symbol table combined separate chaining with stack.

A 1.〔ブロックの開設〕
スタックにブロックの入口の印を積む．

A 2.〔名前の格納〕
ブロックの先頭で宣言定義された名前を分散関数によって指標に変換し，この値をスタックに積む．新しい要素を入れるセルを動的に生成し，名前とその属性をそこに格納する．そして，分散表のこの指標が指すリストの先頭にこのセルを挿入する．

A 3.〔名前の探索〕
使用される名前は分散関数によって分散表の指標に変換し，分散表のこの指標が指すリストの先頭から始めて名前が一致するまで探す．一致するものがあれば，その属性が参照される．なければ，その名前は未定義となる．

A 4.〔ブロックの閉鎖〕
スタックのいちばん上から指標を取り出し，分散表のこの指標が指すリストの先頭の要素を削除する．この操作をブロックの入口の印が来るまで繰り返す．これによって，名前の削除に際しても，スタックを用いる場合と同じように効率よく処理できる．

次に，構造化されたデータ型からなる名前，とくにレコード型のフィールド名を含めた名前，を考慮した場合の名前表の構成と処理アルゴリズムを示す．

はじめに，名前表の構成の変更について述べる．構造化されたデータ型からなる名前を処理するためには，分離連鎖法のリスト上のセルを拡張してポインタ用のフィールドを一つ付加する．そして，構造型の要素，たとえば配列型の要素あるいはレコード型のフィールド名，を構造型のサブリストとして構成し，こ

```
type alfa  = packed array(.1..16.)  of char ;
     jyear = record era : ( Meiji,Taisho,Showa ) ;
                    year: integer ;
             end ;
     person= record name : alfa ;
                    birth: jyear;
                    sex  : ( male,female )
             end ;
var clerk : person ;
```

図 2 型定義と変数宣言の例
Fig. 2 Example of type definition and variable declaration.

れを上で付加したポインタで指す．したがって，リストの要素を表すセルを可変部付きのレコード型に改訂して，ポインタが一つの場合と二つの場合が取り扱えるようにする．

ここで，**図 2** に示すプログラム片（型定義と変数宣言）を後述のアルゴリズムによって処理し，格納したときの分散表とリストとの構成を**図 3** に示す（ただし，図 3 の分散表の順序は見やすくするために並べ変えている）．図 3 を用いて，構造化された型からなる名前を処理した場合の名前表の構成を具体的に示す．たとえば，レコード型の型名 person では，その要素すなわちフィールド名 name, birth, sex をサブリストとして構成し，それを person が格納されているセルの付加したポインタで指している．また，名前が標準型名以外の型名を用いて宣言定義されている場合にも，構造型に準じた取扱いを行い，その型名を探し出し，それをサブリストとして連結する．たとえば，図 3 で，変数名 clerk と型名 person で示されるように clerk のセルから person のセルを指させている．

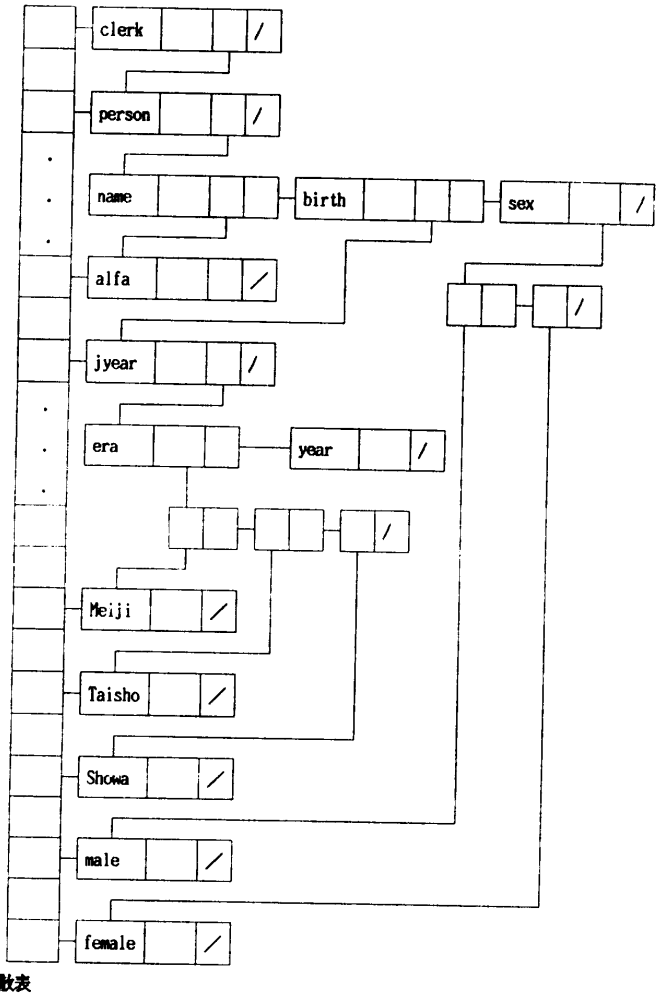

図 3 図 2 の宣言定義に基づく名前表の構造
Fig. 3 The structure of symbol table according to Fig. 2.

次に，アルゴリズムの手直しについて述べる．前述のアルゴリズムでは，すべての名前が分散関数の処理対象になっていたが，ここではレコード型のフィールド名のみは，その有効範囲の規定を考慮して，分散関数の処理対象からはずさねばならない．

A 1.〔ブロックの開設〕の場合，次のようなアルゴリズムが付加される．

レコード型変数に用いる with 文で with を認識したときには，ブロックの開設に準じた取扱を行い，スタックにブロックの入口の印を積む．

A 2.〔名前の格納〕の場合は次のことを付加する．

構造化されたデータ型からなる名前を処理する場合には，ポインタを二つもつセルが動的に生成され，構造型の各要素は，このセルのサブリストとして再帰的に構成される．このとき，分散関数の処理対象になる名前はリストのレベル 1 に格納されるが，分散関数の処理対象にならないレコード型のフィールド名はサブリストに格納される．図 3 では，レコード型のフィールド名 name, birth, sex, あるいは，era, year は分散関数の処理対象にならず，それらはそれぞれレコード型の型名 person あるいは jyear のサブリストに格納されていることに注意されたい．

次に，with 文は，A 1.〔ブロックの開設〕の手直しで述べたように，ブロックに準じた取扱をされるから，with 文で指定

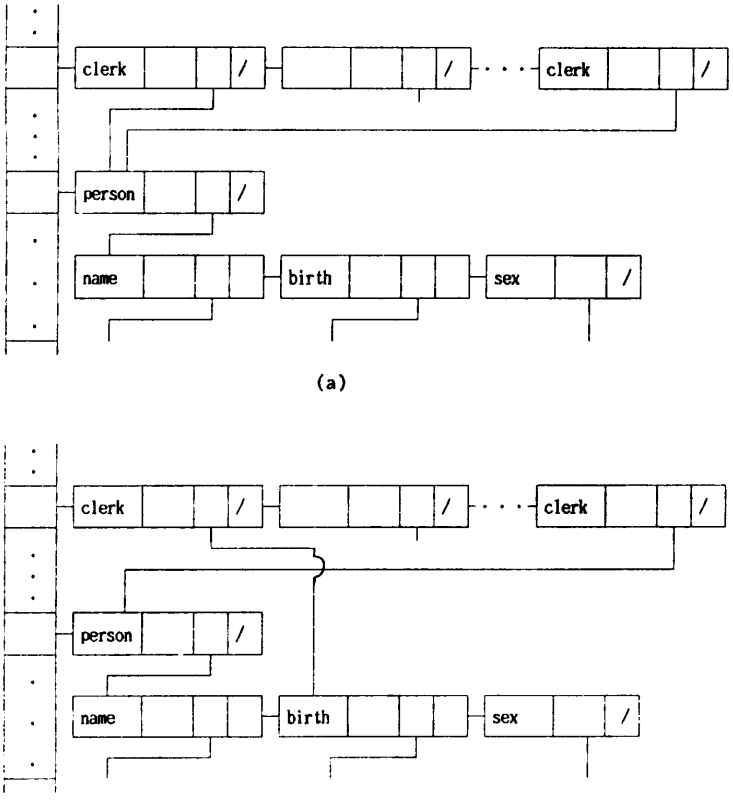

図 4　with 文におけるレコード変数の処理
Fig. 4　Processing of record variable using the with statement.

したレコード変数名は名前の宣言とみなした処理をする．すなわち，レコード変数名は分散関数によって指標に変換し，この値をスタックに積む．そして，新しいセルを動的に生成し，分散表のこの指標が指すリストの先頭にこのセルを連結する．このとき，このレコード変数名はすでに宣言されているはずであるから，このレコード変数名をこのリストのなかから探し出し，そのレコード変数名が格納されているセルの内容をリストの先頭に挿入したセルに複写し，そのフィールド名の有効範囲を規定する．

たとえば，次の with 文，

with clerk do 〈文〉;

を処理した場合のリストの構造を図 4 (a)に示す．このとき，with 文の clerk はリストの先頭に挿入され，それはこのリストにすでに格納されている clerk が指していると同じ person を指す．

また，次の with 文，

with clerk. birth do 〈文〉;

を処理する場合には，clerk. を認識した時点で，上述した図 4 (a)の場合と同様な処理をし，次の birth に対しては，型名 person のフィールド名のなかから birth を探し出し，図 4 (b)で示すように連結する．

A 3.〔名前の探索〕の場合は次のことを考慮する．レコード変数のフィールド名は，そのレコード変数の型に関係づけて有効範囲を規定される．このような場合，フィールド名はそのレコード変数名が格納されているセルのサブリストに格納されているので，このセルから直接辿ることによって探される．このとき，フィールド名がまた構造型になっていても（たとえば，図 3 のフィールド名 birth），リスト構造を順次辿ることによって容易に探すことができる．

一方，with 文の do に続く〈文〉での名前の探索は，前述のアルゴリズムと少し異なる．つまり，do に続く〈文〉での名前は，まず，名前が with 文で指定したレコード変数のフィールド名であるかどうか調べるため，スタックのいちばん上から取り出した分散表の指標が指すリストの先頭の要素（そのレコード変数が格納されているセル）から，図 4 (a)，(b)で例示したように，ポインタを辿りそのサブリストを探す．このとき，一致するフィールド名がなければ，次の指定したレコード変数のフィールド名かどうか調べるため，この操作を with 文の入口の印が来るまで繰り返す．もし，この操作によっても名前がフィールド名に対応しなければ，探索は通常の名前の探索アルゴリズムに従う．その結果，フィールド名はそのレコード変数に所属したものが探されるとともにフィールド名でない名前はブロックの区域の先頭で宣言定義された名前が順次探される．

A 4.〔ブロックの閉鎖〕の修正は，上述したアルゴリズムの手直しから次のようになる．

with 文が終了したときには，ブロックが閉鎖した場合の処理と同じようにできる．すなわち，スタックのいちばん上から順次取り出した指標によって，with 文で用いられた各レコード変数名がどのリストに格納されているかがわかる．したがって，それぞれのリス

トの先頭のセルを削除すればよい．この操作を with 文の入口の印がくるまで繰り返す．

なお図3で示すように，(Meiji, Taisho, Showa), (male, female) などのスカラ値の名前は文法上一意性を必要とし，定義のあと定数名として使用されるので，分散関数の処理対象となる．しかも，スカラ値の名前はスカラ型に属する値を指示するための識別名として列挙され，その列挙順序が意味をもつ．したがって，セルの中の属性を表すフィールドにもポインタをもたせ，それらの順次関係をサブリストとして形成している．

4. 探索路長の解析

提示した混成法で名前表を構成した場合とスタックのみで名前表を構成した場合について，ブロック間での名前の参照頻度を考慮に入れ，その探索路長の解析を行い，その評価式を導出する．次に，この導出した評価式を用いて，プログラム構造が現実に即した場合と仮想的な場合をモデルにして，混成法とスタック法における名前表の探索路長を評価比較する．

解析に先立って次のような記号を導入する．ブロックの入れ子の深さを i ($i=1,2,\cdots,h$, ただし，h は現在処理中のブロックの入れ子の深さとする) で表す．次に，入れ子の深さ i のブロックで宣言定義された名前の数を n_i とし，この名前の数が名前の総数 N ($N=\sum_{i=1}^{h} n_i$) に対する割合を w_i ($w_i=n_i/N$) で表す．また，p_{ij} ($i=1,\cdots,h$, $j=1,\cdots,n_i$) は入れ子の深さ i のブロックで先頭から j 番目に宣言定義された名前が現在処理中のブロック，すなわち，深さ h のブロックで参照される確率とし，この確率の和を P_i で表す．

このとき，$P_i=\sum_{j=1}^{n_i} p_{ij}$, $\sum_{i=1}^{h} P_i=1$ が成立する．ところで，探索路長の平均，分散の評価式の導出に際しては，入れ子の深さ i のブロックで宣言定義された名前は，すべて等しい確率 ρ_i で参照されると仮定し，$\rho_i=P_i/n_i$, ($i=1,\cdots,h$) とする．

次に，探索が成功するときの平均探索路長を S_N，分散を V_N，不成功のときのそれらをそれぞれ $\bar{S}_N$, $\bar{V}_N$ とする．

また，成功時の探索路長は，ブロック間での参照頻度を考慮した次の四つの場合について解析評価する．ただし，仮想的なプログラム構造を用いた a)，b)，c) の場合は各ブロックで宣言定義された名前の数は等しく n ($n=n_i, i=1,\cdots,h$) と仮定し，現実に即したプログラム構造を用いた d) の場合は，実際のプログラム例から統計的に算出した P_i, w_i を用いる．

a) 名前が参照される頻度が入れ子の深さに関係なく一様な場合

このとき，ρ_i は次のようになる．

$$\rho_i=1/nh, \quad (i=1,\cdots,h) \qquad (1)$$

b) 名前が参照される頻度が入れ子の深さに依存し，現在処理中のブロックを囲むブロックを内から外に行くに従いブロック間での参照頻度が逐次半減する場合

このとき，ρ_i は次のようになる．

$$\rho_i=\rho_h/2^{h-i}, \quad (i=1,\cdots,h)$$
$$\rho_h=2^{h-1}/n(2^h-1) \qquad (2)$$

c) 名前が参照される頻度が入れ子の深さに依存し，現在処理中のブロックを囲むブロックを内から外に行くに従いブロック間での参照頻度が調和減少する場合

このとき，ρ_i は次のようになる．

$$\rho_i=\rho_h/(h-i+1), \quad (i=1,\cdots,h)$$
$$\rho_h=1/nH_h \qquad (3)$$

ただし，$H_h=1+1/2+\cdots+1/h$ とする．

d) 現実に即したプログラム構造として，Pascal で記述された PL/0 コンパイラのプログラム例[2]を用いて，入れ子の深さに伴うブロック間での名前の参照頻度 P_i と名前の数の比率 w_i を統計的に算出した場合

このとき，P_i, w_i ($i=1,\cdots,h$) はプログラムの構造に伴い次のようになる．

$h=2$ のとき，

$P_1=0.4$, $P_2=0.6$

$w_1=0.75$, $w_2=0.25$

$h=3$ のとき，

$P_1=0.7$, $P_2=0.1$, $P_3=0.2$,

$w_1=0.75$, $w_2=0.15$, $w_3=0.1$,

$h=4$ のとき，

$P_1=0.7$, $P_2=0.05$, $P_3=0.05$, $P_4=0.2$

$w_1=0.85$, $w_2=0.05$, $w_3=0.05$, $w_4=0.05$,

$h=5$ のとき，

$P_1=0.7$, $P_2=0.0$, $P_3=0.0$, $P_4=0.0$, $P_5=0.3$,

$w_1=0.95$, $w_2=\varepsilon$, $w_3=\varepsilon$, $w_4=\varepsilon$, $w_5=0.05$

ただし，ε はゼロに近い非常に小さい値．

この算出データから，入れ子が深くなれば，深さ1のブロックで宣言定義された大域的な名前の数の割合

やそれが参照される頻度が大きくなる傾向が強いことが示される.

ここでは, ρ_i は次のようになる.

$$\rho_i=P_i/Nw_i,\ (i=1,\cdots,h) \qquad (4)$$

4.1 スタックを用いた名前表

この方法では登録および削除が簡単であり, また名前表の使用領域も少なくてできるが, プログラムが多重入れ子構造で名前が広域的に使用されるならば, 探索効率は良くない.

はじめに, 成功時の探索路長の評価式の導出に先立ち, 仮想的なプログラム構造を用いた場合, 各名前が参照される確率からなる母関数を $G(z)$ とする. このとき, 母関数 $G(z)$ は,

$$\begin{aligned}G(z)=&\rho_h(z^{n+1}-z)/(z-1)\\&+\rho_{h-1}(z^{n+1}-z)z^n/(z-1)+\cdots\\&+\rho_1(z^{n+1}-z)z^{(h-1)n}/(z-1)\end{aligned} \qquad (5)$$

となる.

ここで, a), b), c)の各場合の ρ_i はそれぞれ前式(1), (2), (3)で与えられているので, それらを(5)に代入すると, 各場合の母関数はそれぞれ次のようになる.

a)の場合の母関数

$$G(z)=(z^{nh+1}-z)/(nh(z-1))$$

b)の場合の母関数

$$G(z)=(z^{n+1}-z)(z^{nh}-2^h)/(n(2^h-1)(z-1)(z^n-2))$$

c)の場合の母関数

$$\begin{aligned}G(z)=&(z^{n+1}-z)(1+z^n/2+z^{2n}/3+\cdots+z^{(h-1)n}/h)\\&/(nH_h(z-1))\end{aligned}$$

これらの母関数に微分法を用いれば, 成功探索路長の平均 S_N, 分散 V_N は

$$S_N=G'(1)$$

$$V_N=G''(1)+G'(1)-G'(1)^2$$

となる. よって, それぞれの探索路長の評価式は次のようになる.

a) 名前の参照頻度が一様な場合

$$S_N=(nh+1)/2 \qquad (6)$$

$$V_N=((nh)^2-1)/12 \qquad (7)$$

b) 名前の参照頻度がブロックが浅くなるに従い半減する場合.

$$S_N=(3n+1)/2-nh/(2^h-1) \qquad (8)$$

$$\begin{aligned}V_N=&(25n^2-1)/12-(nh)^2/(2^h-1)\\&-(nh/(2^h-1))^2\end{aligned} \qquad (9)$$

c) 名前の参照頻度がブロックが浅くなるに従い調和減少する場合.

$$S_N=nh/H_h-(n-1)/2 \qquad (10)$$

$$\begin{aligned}V_N=&(n^2-1)/12+((nh)^2+n^2h)/2H_h\\&-(nh/H_h)^2\end{aligned} \qquad (11)$$

d) PL/0 コンパイラのプログラム例を用いた場合

算出した P_i, w_i $(i=1,\cdots,h)$ をもとに(4)から ρ_i を求め, 探索路長の平均, 分散を評価する.

一方, 探索が不成功のときには, a), b), c), d)のいずれの場合においても探索路長は次のようになる.

$$\bar{S}_N=nh+1 \qquad (12)$$

$$\bar{V}_N=0 \qquad (13)$$

4.2 混成法を用いた名前表

この混成法を用いた名前表において, 名前を探索するときには, スタックは用いないで分離連鎖法を用いて行う. この分離連鎖法における探索路長の基本的な解析はすでに試みられている[3),4)]. したがって, ここではその解析法を応用する.

はじめに, 探索路長の評価式の導出過程で用いる記号を次のように導入する.

まず, すべての名前は大きさ M の分散表に一様に分配され, そのとき, 表の占有率 (load factor) を $\alpha(N/M)$ とする.

q_{Nk}: 任意のリストに k 個の名前が登録される確率とする.

このとき, q_{Nk} は

$$q_{Nk}={}_NC_k(1/M)^k(1-1/M)^{N-k},\ (k=0,\cdots,N) \qquad (14)$$

となる. また, この q_{Nk} の母関数 $Q(z)$ を

$$Q(z)=(1-(z-1)/M)^N$$

とする.

r_{kj}: k 個の名前が連結された任意のリストで, リストの先頭から j 番目の名前が探される確率.

ここでは, 分散表の上で同じ指標となる名前は登録される順序に従いリストの先頭に順々に挿入されるとするから, N 個の名前のうち j 番目に登録された名前が, k 個の名前からなるリストの先頭から j 番目に配置される確率は

$${}_{N-i}C_{j-1}\ {}_{i-1}C_{k-j}/{}_{N-1}C_{k-1}$$

となる.

したがって, このリストの先頭から j 番目にすべての名前が配置される可能性を考慮すると, r_{kj} は次のようになる.

$$r_{kj}=\sum_{i=1}^{N}{}_{N-i}C_{j-1}\ {}_{i-1}C_{k-j}\rho_{i^*}/{}_{N-1}C_{k-1} \qquad (15)$$

---

* ρ_i の添字 i^* は $i<n_1+\cdots+n_i,\ i=1,\cdots,h$ を満たす最小の値とする.

このとき，成功探索路長の平均，分散は次のように表現される．

$$S_N=\sum_{k=1}^{N} k \sum_{i=k}^{N} r_{ik}q_{Ni} \Big/ \sum_{k=1}^{N} q_{Nk} \qquad (16)$$

$$V_N=\sum_{k=1}^{N} k^2 \sum_{i=k}^{N} r_{ik}q_{Ni} \Big/ \sum_{k=1}^{N} q_{Nk}-S_N{}^2 \qquad (17)$$

上の式から，各場合の探索路長の評価式は次のようになる．

a） 名前の参照頻度が一様な場合

(1)と(15)から，$r_{ik}=1/i$ となる．よって，探索路長の平均，分散は次のようになる．

$$\begin{aligned} S_N&=\sum_{k=1}^{N} k \sum_{i=k}^{N} 1/i q_{Ni} \Big/ \sum_{k=1}^{N} q_{Nk} \\ &=(Q'(1)+Q(1)-Q(0))/(2(1-Q(0)) \\ &=(1/2)(N/M)/(1-(1-1/M)^N)+1) \\ &\fallingdotseq(\alpha/(1-e^{\alpha})+1)/2 \end{aligned} \qquad (18)$$

$$\begin{aligned} V_N&=\sum_{k=1}^{N} k^2 \sum_{i=k}^{N} 1/i q_{Ni} \Big/ \sum_{k=1}^{N} q_{Nk}-S_N{}^2 \\ &=(1-Q(0)+5Q'(1)+2Q''(1))/(6(1-Q(0)) \\ &\quad -S_N{}^2 \\ &=(N(N-1)/M^2+N/M)/(3(1-(1-1/M)^N) \\ &\quad -((N/M)/(2(1-(1-1/M)^N))^2-1/12 \\ &\fallingdotseq(\alpha(\alpha+1)/(3(1-e^{-\alpha})) \\ &\quad -\alpha^2/(4(1-e^{-\alpha}))^2-1/12 \end{aligned} \qquad (19)$$

b） 名前の参照頻度がブロックが浅くなるに従い半減する場合．

このとき，(2)と(15)から，r_{ik} は

$$r_{ik}=\sum_{j=1}^{M} {}_{N-i}C_{k-1}\ {}_{i-1}C_{i-k}2^{h-1} \Big/ (n(2^h-1)(2^h-\lceil j/n \rceil^{*})_{N-1}C_{i-1}$$

となる．したがって，これを(16)，(17)に代入し，探索路長の平均，分散を評価する．

c） 名前の参照頻度がブロックが浅くなるに従い調和減少する場合．

このとき，(3)と(15)から，r_{ik} は

$$r_{ik}=\sum_{j=1}^{N} {}_{N-j}C_{k-1}\ {}_{j-1}C_{i-k} \Big/ nH_n\left(h-\left\lceil \frac{j}{n} \right\rceil^{*}+1\right)_{N-1}C_{i-1}$$

となる．したがって，これを(16)，(17)に代入し，探索路長の平均，分散を評価する．

d） PL/0 コンパイラのプログラム例を用いた場合

算出した P_i, w_i $(i=1,\cdots,h)$ をもとに(4)から ρ_i を求める．これを(15)に代入し，(16)，(17)から，探索路長の平均，分散を評価する．

表 1 スタックを用いた名前表における探索路長の比較

Table 1 Comparisons of the search length in case of using stack.

Number of nested blocks (h)	Total number of names (N)	Successful search							
		a) Uniform		b) Half		c) Harmonic		d) PL/0 Compiler	
		S_N	V_N	S_N	V_N	S_N	V_N	S_N	V_N
1	10	5.5	8.2	5.5	8.2	5.5	8.2	5.5	8.2
2	20	10.5	33.2	8.8	30.5	8.8	30.5	7.0	32.7
3	30	15.5	74.9	11.2	61.3	11.8	67.7	14.5	83.7
4	40	20.5	133.2	12.8	94.5	14.7	119.6	17.2	160.7
5	50	25.5	208.2	13.9	125.0	17.4	185.7	18.9	274.1

表 2 提案する名前表における探索路長の比較

Table 2 Comparisons of the search length in case of using the proposed symbol table.

Number of nested blocks (h)	Total number of names (N)	Load factor (α)	Successful search							
			a) Uniform		b) Half		c) Harmonic		d) PL/0 Compiler	
			S_N	V_N	S_N	V	S_N	V_N	S_N	V_N
1	10	0.2	1.047	0.05	1.047	0.05	1.047	0.05	1.047	0.047
2	20	0.4	1.102	0.11	1.084	0.09	1.084	0.09	1.139	0.148
3	30	0.6	1.160	0.17	1.113	0.12	1.120	0.13	1.177	0.185
4	40	0.8	1.222	0.25	1.134	0.15	1.156	0.18	1.259	0.298
5	50	1.0	1.286	0.33	1.151	0.17	1.192	0.23	1.210	0.523

* $\lceil x \rceil$ は x 以上の最小の整数をあらわす．

一方，探索が不成功のときの探索路長は次のようになる．

$$\bar{S}_N=\sum_{k=0}^{N}(k+\delta_k)q_{Nk} \tag{20}$$

$$\bar{V}_N=\sum_{k=0}^{N}(k+\delta_k)^2q_{Nk}-\bar{S}_N{}^2 \tag{21}$$

ただし，

$$\delta_k=\begin{cases}1,(k=0)\\0,(k\neq 0)\end{cases}$$

4.3 数値例

ここでは，4.1，4.2 節で導出した探索路長の評価式を用いて具体的な数値例を示す．

現実のプログラムである d）の場合，入れ子の深さはたかだか5程度で，それ以上深くなることはきわめて少ない．したがって，この数値例では，入れ子の深さ h を $h=1,2,\cdots,5$ とし，それに伴う名前の総数 N を $10*h$ とした場合の探索路長を計算した．

このとき，名前表にスタックを用いた場合の探索路長を**表1**に，混成法を用いた場合の探索路長を**表2**にそれぞれ掲げる．

この数値例から明らかなように単純にスタックのみで名前表を構成したときに比べ，混成法を用いた名前表では著しく探索路長が短縮されることが数量的に把握できる．また，近隣のブロックで宣言定義された名前がより多く参照されると仮定した b），c）の場合には，a），d）の場合に比べ，表1，表2のいずれの場合においても，探索路長が短くなることがわかる．さらに，現実に即したプログラム構造を用いた d）の場合，表1では入れ子の深さに伴い平均探索路長は単調に増加しているが，表2では入れ子の深さが4および5のときの平均探索路長は 1.259 および 1.210 となる．このことから，混成法を用いた名前表では，入れ子が深くしかも名前の総数が多くなっているにもかかわらず，平均探索路長が，かえって短くなることがありうることに注意されたい．

5. むすび

ブロック構造を許す Pascal 型言語向きの名前表の一構成法を提案し，ブロック間での名前の探索頻度を考慮したときの探索路長の評価式を導出し，その評価式を用いて，スタックを用いた名前表と，提示した混成法を用いた名前表における探索路長を具体的に評価した．その結果，この混成法による名前表が，入れ子構造を許す Pascal 型の言語処理に有効であり，とくに，名前の探索だけでなく削除の際にも効率がよいことを示した．また，導出した探索路長の評価式から現実の現象にあった探索路長が評価でき，適切な名前表の選択が可能になることを示した．

本稿では主として探索路長に関して解析を行ったが，名前表の構成には，名前に関する情報を格納する領域と管理に要する領域とを合わせた所要記憶領域をもあわせて検討する必要がある．すなわち，探索路長の短縮と管理に要する記憶領域の増加とのかねあいで名前表の構成を評価する必要がある．

提示した混成法を用いた名前表では，スタックと分離連鎖法で用いるポインタとを管理領域として必要とするが，この名前表は，情報を格納しているセルの大きさに対するポインタなどの管理領域の割合が小さいほど有効となる．

なお，この名前表は筆者らが作成した Pascal 風言語を処理するプリプロッサの中で使用した．このとき，このプリプロセッサを Pascal を用いて記述したところ，この名前表の構成ならびに処理アルゴリズムを簡潔に表現することができた．

謝辞　本稿をまとめる際，九州大学の藤村直美，荒木啓二郎両氏に貴重なご意見をいただいた．ここに記して謝意を表す．

参考文献

1) Gries, D.: *Compiler Construction for Digital Computers*, pp. 212-230, John Wiley & Son, New York (1971).
2) Wirth, N.: *Algorithms+Data Structures=Programs*, pp. 337-347, Prentice-Hall, Englewood Cliffs, N. J. (1976).
3) Knuth, D. E.: *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, pp. 506-518, Addison-Wesley, Reading, Mass. (1973).
4) 中村，松山：分散記憶法における探索頻度を考慮した探索路長とその評価，情報処理学会論文誌，Vol. 24, No. 1, pp. 125-130 (1983).

(昭和59年8月10日受付)
(昭和60年2月21日採録)