ICOT Technical Report: TM-0013

TM-0013

Prologによる
ビットテーブルの扱い

石井　暁

August, 1983

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Prologによる
ビットテーブルの扱い

石井　暁

概要

Prologでプログラム作成，実行し，参照速度等を評価することによりビットテーブルの扱いの面から論理型言語を考察した．ここでビットテーブルの使用目的は，ある文字列が辞書にあるか否かの判定を想定した。

手法を変えた4通りのプログラムおよびFortranプログラムによる評価の結果，ユニファイと"arg"による要素のとり出しを行うPrologプログラムの性能はかなり良く，Fortranと比較して速度，容量共1/3の性能を持っていることが解った．

目　次

1．はじめに

ビットテーブルの扱いにおけるデータ構造の表現能力，参照速度等の面から論理型言語を考察した。論理型言語としてPrologを用い，実際のプログラムの作成，実行により実証的な考察とした。

ここでのビットテーブルとは，ビット列を定義し，各ビットが1か0かによって何らかの状態を表わすためのものである。例えばOSにおいてディスクのあるセクタが使用中か否かを記憶するのに用いるものである。

この評価実験においてビットテーブルの使用目的として自然言語において，ある文字列が辞書にあるか否かの判定を想定する。[3] 例えばZエディタのスペルチェッカ[2]はウェブスター辞書の25000語をビットテーブルの500000ビットの1/0に対応させており，ある文字列がウェブスター辞書にあるかを，500000ビットのうちの最大10ビットの1/0を調べることによって判断している。（Zエディタの方法は辞書にない文字列を辞書にあると判断することもある）

Zエディタの方法では，ビットテーブルの大きさは辞書の語数の10から数10倍必要であり，数10万ビットの大きさが必要となる。[1]

また，ビットの参照の速度が充分速いことが必要となる。例えば1行に10単語，10行に1か所のミススペルと仮定し，1単語のチェックに10ビットを参照し，1ビットの参照に0.1m秒かかるとすると，対話型システムにおいて1つのミススペルの処理が終ってから次のミススペルがみつかるまでの時間は

$$0.1^{\text{m秒}}/_{\text{参照}} \ast 10^{\text{参照}}/_{\text{単語}} \ast 10^{\text{単語}}/_{\text{行}} \ast 10^{\text{行}}/_{\text{エラー}}$$
$$= 0.1\ \text{秒}/\text{エラー}$$

となり，この程度の速度は必要であろう*．
なお，ビットの書きかえの速度はそれ程要求されないので，本評価実験において考えていない．

使用した計算機はICOTのdec 2060である．Prologは Prolog-20 version 1.0であり，全ての述語にmode指定を付けてある．

---

* 各単語について参照される全ビットがばらばらにはなれていると仮定している．

2. プログラム

次の5通りのプログラムにより評価した。

- Prolog（ユニファイ・ターム方式）
- Prolog（タームのターム方式）
- Prolog（ユニファイ・リスト方式）
- Prolog（リストのリスト方式）
- Fortran

以下に各々の説明をする。

2.1 Prolog（ユニファイ・ターム方式）

ビットテーブルを表現する際，第1図に示す様に複数のアサーションに分解し，各アサーションにおいてはコンパウンドタームとして表現するものである。ここで a_{ij} は整数であり，最大18ビット使えるが，本実験では右側10ビットのみ用いている。また，$m = 50$，$n = 300$ とした。

nビット目のビットの値を知るアルゴリズムは自明であるが，次のステップで行われる。

(i) どのアサーションの中のタームにそのビットがあるかを計算し，ユニファイによりそのタームを取り出す。
(ii) そのタームの何番目の要素に求めるビットがあるかを計算し，組み込み手続き"arg"によりその要素を取り出す。
(iii) その要素の何番目のビットが求めるものかを計算し，シフトによりそれを取り出す。

プログラムの大きさは約20行である。

bit(1, b(a_{11}, a_{12}, ------ $a_{1\,m-1}$, a_{1m})).

bit(2, b(a_{21}, a_{22}, ------ $a_{2\,m-1}$, a_{2m})).

⋮

bit(n, b(a_{n1}, a_{n2}, ----- $a_{n\,m-1}$, a_{nm})).

第1図　ユニファイ・ターム方式のビットテーブル

（ビットテーブルそのものおよびコメント行を除く）

2.2 Prolog（タームのターム方式）

ビットテーブルを表現する際，第2図に示す様に1つのアサーション中のタームのタームとして表すものである．その他は前のものと同じである。ただし，n＝100とした。

nビット目のビットの値を知るアルゴリズムは前のユニファイ・ターム方式と同様であるが，(i)が異り，

(i) ターム中の何番目のタームにそのビットがあるかを計算し，"arg"によりそのタームを取り出す。

となる。

プログラムの大きさはユニファイ・ターム方式と同様である．

この方式では，ビットテーブルが大きくなると，エディタで参照している時，その何行目を扱っているかが解りにくくなる．第1図の方式においては，述語bitの第1アーギュメントにより，かなりその点で解り易くなる。大きなプログラム作成時には考えておくべき特徴であろう。

2.3 Prolog（ユニファイ・リスト方式）

ビットテーブルを表現する際，第3図に示す様に複数のアサーションに分解し，各アサーションにおいてはリストとして表現するものであ

$$
\begin{aligned}
bit(\ bi(\ & b(a_{11}, a_{12}, \cdots\cdots\ a_{1\,m-1}, a_{1m}), \\
& b(a_{21}, a_{22}, \cdots\cdots\ a_{2\,m-1}, a_{2m}), \\
& \vdots \\
& b(a_{n1}, a_{n2}, \cdots\cdots\ a_{n\,m-1}, a_{nm}))).
\end{aligned}
$$

第2図　タームのターム表式のビットテーブル

bit (1, [a_{11}, a_{12}, ····· , $a_{1\,m-1}$, $a_{1\,m}$]).

bit (2, [a_{21}, a_{22}, ····· , $a_{2\,m-1}$, $a_{2\,m}$]).

⋮

bit (n, [a_{n1}, a_{n2}, ····· , $a_{n\,m-1}$, $a_{n\,m}$]).

第3図　コニファイ・リスト方式のビットテーブル

る．ユニファイ・ターム方式と比較するとコンパウンドタームがリストに変わったものであり，その他は同様にしてある．

nビット目のビットの値を知るアルゴリズムはユニファイ・ターム方式と同様であるが，次の様である．

(i) どのアサーションの中のリストにそのビットがあるかを計算し，ユニファイによりそのリストを取り出す．

(ii) そのリストの何番目の要素に求めるビットがあるか計算し，その要素をとり出す．

(iii) その要素の何番目のビットが求めるものかを計算し，シフトによりそれを取り出す．

プログラムの大きさは，リストの要素を取り出す手続き（2～3行）分だけユニファイ・ターム方式より多い．

2.4 Prolog（リストのリスト方式）

ビットテーブルを表現する際，第4図に示す様に1つのアサーションのリストのリストを用いる．その他はユニファイによるものと同様である．ただし，n=100とした．

nビット目の値を知るアルゴリズムは，ユニファイ・リスト方式のプログラムと比較して(i)のみが異り，

(i) リスト中の何番目のリストにそのビットがあるかを計算し，そのリストを取り出す．

bit([[a_{11}, a_{12}, ... $a_{1\,m-1}$, $a_{1\,m}$],
[a_{21}, a_{22}, ... $a_{2\,m-1}$, $a_{2\,m}$],
⋮
[a_{n1}, a_{n2}, ... $a_{n\,m-1}$, $a_{n\,m}$]]).

第4図　リストのリスト方式のビットテーブル

となる．
プログラムの大きさについてはユニファイ・リスト方式のプログラムと同様である．

2.5 Fortran

ビットテーブルとして整数の1次元配列を用いる．Fortranでは整数は36ビットであるが，符号ビットを除いた35ビットを用いた．本実験では配列の大きさを10,000（350,000ビット）とした．
Nビット目の値を知るアルゴリズムは自明である．ただし，整数に対してビットのシフトを行う関数，サブルーチンが用意されていない様なので，それを2の何乗かの数との演算により陽に書いた．
プログラムサイズは20行余りで，Prologの場合と大差はない．（付録のプログラには使われていない部分や，ビットに値をセットするサブルーチンも含まれており，もっと長い）

3. 評価

評価は定義し得るビットテーブルの大きさと速度について行われた。

3.1 大きさの評価

(1) Prolog（ユニファイ・タ-ム方式）

一応本実験では15,000整数で150,000ビット（フルに使えば270,000ビット）で行った。しかし，本実験で直接の限度となったのは，Prologにおける整数の値の上限である。即ちビットテーブルの"何番目"のビットかを表わすのに整数を用いたが，それが18ビットなので約130,000の数までしか表現できないことがネックになった。（それ以上の数を扱う拡張もされているが，それが全ての場合に思い通りには動作をしない様である）

ビットテーブル自体はコンパイラを通すとき

90,000整数で900,000ビットは可
95,000整数で950,000ビットは不可

また，コンパイラを通さなければ

95,000整数で950,000ビットは可

である．

(2) Prolog（タ-ムのタ-ム方式）

本方式においては上の数字よりかなり減り，

コンパイラを通すとき

7,500 整数で 75,000 ビットは可
10,000 整数で 100,000 ビットは不可

であり，コンパイラを通さないとき

5,000 整数で 50,000 ビットは可
7,500 整数で 75,000 ビットは不可

であった。コンパイラを通す方が多くのビットを定義できるのは，これのみであった。なお，この方式の速度の評価は 50,000 ビットを定義して行った。

この方式や後に述べるリストのリスト方式の様な1つのアサーションでビットテーブルを表わす方式において，上記の制約は1つのアサーションについての制約であり，アサーションを複数にすることにより，更に大きなビットテーブルを定義することができる。

(3) Prolog（ユニファイ・リスト方式）

ユニファイ・ターム方式と同様な条件で行った。直接の上限が Prolog の整数が 18 ビットであることで決まっている点も同様である。

ビットテーブル自体はコンパイルする時，しない時共，

45,000 整数で 450,000 ビットは可
50,000 整数で 500,000 ビットは不可

であった。

(4) Prolog（リストのリスト方式）

コンパイラを通す場合

5,000 整数で 50,000 ビットは可
6,000 整数で 60,000 ビットは不可

であった。整数の18ビットをフルに使えば，約100,000ビットまで扱える。なお，コンパイラを通さなければ少くとも

7,000整数で70,000ビットは可

であった。
速度の評価の実験は50,000ビットを定義して行った。

(5) Fortran

本実験は10,000整数で350,000ビットを扱ったが，更に大きなテーブルを扱えて

130,000 整数で 4,550,000 ビットは可
140,000 整数で 4,900,000 ビットは不可

であった。

3.2 速度の評価

基本的には速度は1ビットを参照するのに必要な時間で評価される。しかし，リストを用いた場合には，参照したいビットがリストのどの位置にあるかによって必要な時間が変化すると思われる。

そこで，本実験においては，ビットテーブルの大きさ（ビット数Sで表す）を変化させ，各々の場合についてビットテーブルの全ビットを順に1ビットずつ参照するのに必要なCPU時間を測定することとした。*

これは，Prologにおいては"statistics"によって得られる。ただし，時間がxwd表示になる（129.071秒以上になる）場合には，表示に不明な点があるため，測定を行わなかった。

Fortranにおいて，このための適当なサブルーチンが見当らなかったので，runコマンドを実行するのにかかった時間を用いた。

これは測定したい時間の他に多くの時間を含むが，それは測定値の解析の時に考慮した。

全てのケースにおいて測定値は3回同条件で測定した値の平均である。また，Prologによる場合，コンパイルしての実行を中心に扱う。マシンの混み方などによる測定値のバラツキは，100 ms以上の値においては3～4%程度におさまっている様である。

＊ 実験は，ビットテーブルの大きさは固定し，その一部の全ビット参照を行った。これによる誤差は特に問題にならない。また，この状況は実際の応用システムでの使用状況とは異なるかも知れない。注意が必要である。

(1) Prolog（ユニファイ・タ－ム方式）

大きさSビットのビットテーブルの全ビットの参照にかかるCPU時間Tを推定する．
ビット列の任意のビットを参照するには

・アサーションのユニファイ
・argによる要素のとり出し

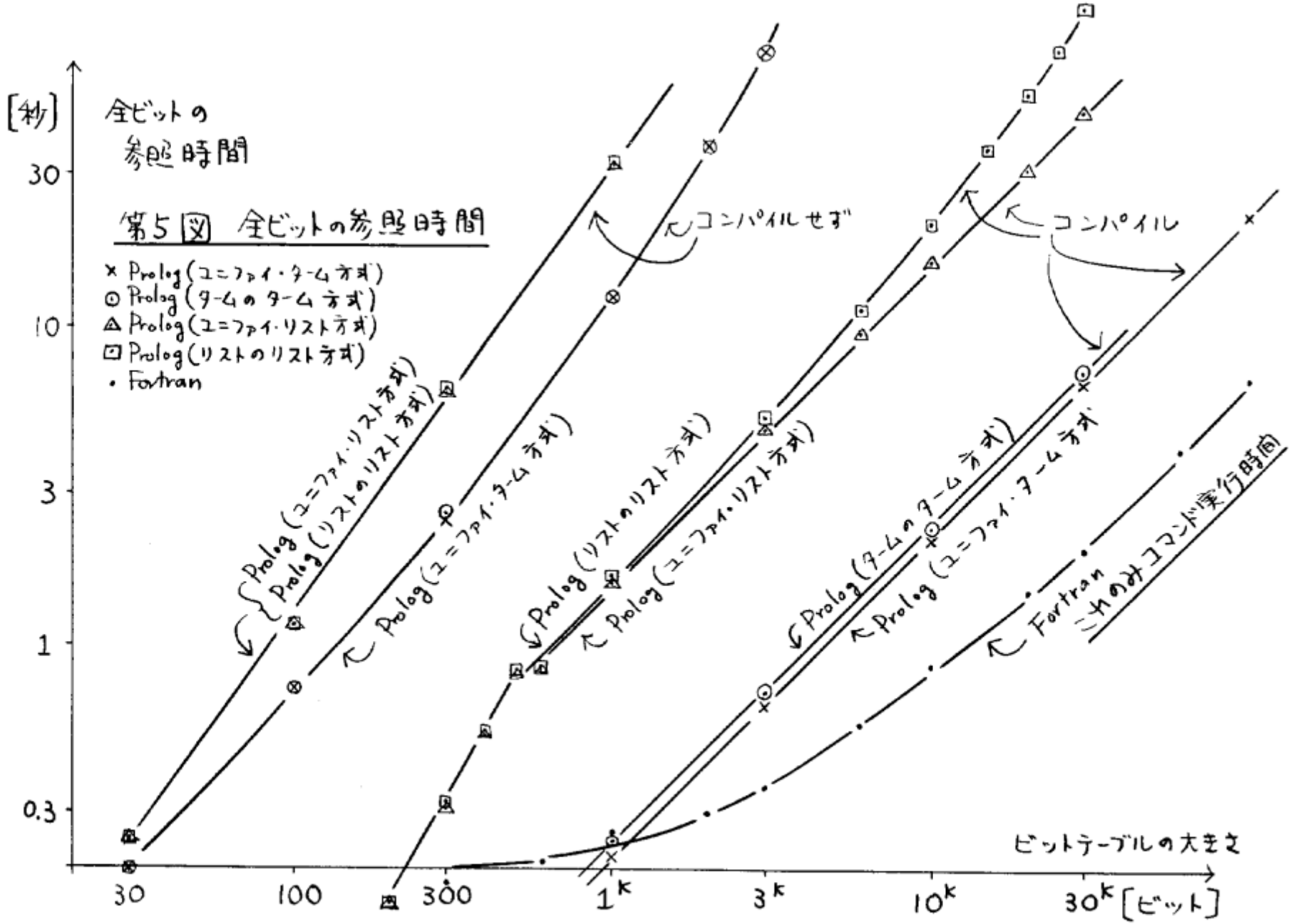
の2つの処理が主に問題になる．前者は処理系の工夫次第であるが，例えばハッシングなどによってどのアサーションがユニファイされる場合でも一定の時間ですむと考えられる．
後者についても，同様にどの要素がとり出される場合でも一定の時間ですむと考えられる．結局，

TはSに比例

と考えられる．
実測値によるグラフを第5図に示す．コンパイルした時は上記の議論が成り立ち，

T は $S^{1.0}$ に比例

となっている．
一方，コンパイルをしない場合，Sが100以下付近では上記議論の通りであるが，Sがそれ以上になると，それ以上の時間がかかる様になる．次に述べるユニファイ・リスト方式ではこの様な傾向が見られないので，これは"arg"の処理方式がコンパイルする場合の様に工夫されていないためと考えられる．
上記をまとめると，コンパイルによるとこの方式では1ビットの参照には



第5図　全ビットの参照時間

約

$0.22m$ 秒

かかる．また，コンパイルの効果はSにより一定ではないが，第5図のグラフの範囲内（$30 < S < 3.000$ 程度で）は

大体 28 ～ 110 倍の効果

である．また，コンパイルの時間は約30秒で．コンサルトする場合の7割増程度である．

(2) Prolog（タームのターム方式）

前のユニファイ・ターム方式と同様の結果であり，それとの相違を中心に述べる．
ビット列の任意のビットの参照には

・ argによる要素の取り出し

が2回行われる。これはどの要素についても一定と考えられるので，大きさSビットのビットテーブルの全ビットの参照にかかるCPU時間Tについて

TはSに比例

となる。
第5図の実測値によると，コンパイルをした時は

TはS$^{1.0}$に比例

となっている．しかし，前のユニファイ・ター

ム方式の場合と比較すると約1割多く時間がかかっており，1ビット参照に約

0.24m秒

必要である．

コンパイルしない場合は，前のユニファイ・ターム方式と全く同じ結果になっており．特に付け加えて説明する点はない。

コンパイルの効果は第5図のグラフの範囲内（30 < S < 3,000 程度で）は

大体 25 ～ 100 倍の効果

である。また，コンパイル時間は約10秒で，コンサルトする場合の3割増程度である。

(3) Prolog（ユニファイ・リスト方式）

大きさSビットのビットテーブルの全ビットの参照にかかるCPU時間Tは，次の様になると考えられる．

(i) $0 << S \leq 500$ のとき

ビット列を表現している複数のアサーションのうち，1つのみで処理が終る．そのためアサーションをユニファイする時間は特に問題にならない。一方，求めるビットが，そのアサーションの中のリストのどの位にあるかの影響は，リストの先頭からの位置に比例した時間がかかると考えられる。そこで．

T は S^2 に比例

となる。

(ii) $S \gg 500$ のとき

(i)とは逆にリストの中からビットをさがす時間は平均すれば一定と考えてよい。一方ユニファイするための時間は，処理系の工夫の結果，どのアサーションがユニファイされる場合でも一定の時間であると考えられる。そこで

T は S に比例

となる。

実測値によるグラフを第5図に示す．コンパイルした場合には，ほぼ上記の議論が成り立っている。即ち

(i) $0 \ll S \leqq 500$ のとき

グラフから

T は $S^{1.8}$ に比例

となる。べき数が2まで達しないのは S に比例の項などが影響しているためと思われる．

(ii) $S \gg 500$ のとき

グラフから

T は $S^{1.0}$ に比例

となる。

(iii) $500 < S < 1000$ 付近

(i)と(ii)の部分がなめらかにはつながらないことが読みとれる．

一方コンパイルをしない場合.

T は $S^{1.4}$ に比例

となっている. これは 300 < S < 1.000 程度のちょうど(i)と(ii)の中間で, 上記議論が成り立たない範囲のためと考えられる.

上記をまとめると, コンパイルした場合数万ビットのビットテーブルにおいて, 1ビットの参照にほぼ一定の時間

1.6 m秒

かかることが解る。

コンパイルの効果はSが30から1000程度の範囲で

大体 20 ～ 24倍

である。なお, コンパイル時間は1分余りであり, コンサルトするのに比較して約3倍の時間がかかる.

このプログラムで, リストAのB番目の要素をCに取り出す述語を

select (A, B, C)

で呼ぶ様にしてある. これを

select (B, A, C)

とアーギュメントの順を変えることにより, コンパイルする時, 3割～4割の速度の向上がある. 処理系のやり方を知ってプログラムを書くか, 書かないかにより, 大きな差が出ることが解る. なお, コンパイルしない場合の速度の向上は, 無視し得る(値のバラツキの)範囲内である。

(4) Prolog（リストのリスト方式）

(i) $0 \ll S < 500$ のとき

(3)に述べたユニファイ方式と同様の議論が成り立ち

T は S^2 に比例

と考えられる．

(ii) $S \gg 500$ のとき

小さいリストからビットをさがす時間は平均すれば一定と考えてよい．しかし，該当する小さいリストを得るためには，リストの先頭からの位置に比例した時間がかかると考えられるので，(i)と同様に

T は S^2 に比例

と考えられる．

第5図の実測値によると

(i) $0 \ll S < 500$ のとき

グラフでは，(3)のユニファイ・リスト方式と重なり，

T は $S^{1.8}$ に比例

となる．

(ii) $S \gg 500$ のとき

かなり予想と異った結果である．これは，実は

$T = S^2$ に比例する項（小さなリスト

を選ぶ操作）
＋Sに比例する項（小さなリストからビットを選ぶ操作など）

であり，図では，両方の項を考えなければならない範囲を表わしているためと考えられる。Sに比例する項はユニファイ・リスト方式での時間Tとほぼ等しいと考えられるので，その差をとると

差は$S^{2.0}$に比例

となった。

図のSが10Kビットの点でも，小さなリストには50の要素があるのに，大きなリストには20の要素しかないため，$S > 500$ではあるが，$S \gg 500$ではなく，上記の様になると考えられる。

上記から，この方式では数万ビットのビットテーブルにおいて，1ビットの参照には約

2.2m秒

かかることが解る。

コンパイルの効果はSが30から1000の範囲で

大体20～24倍の効果

であり，ユニファイ・リスト方式と同様である。なお，コンパイル時間は約1分であり，コンサルトするのに比較して約8倍の時間がかかる。

余談になるが，このプログラムで，リストAのB番目の要素をCに取り出す述語select(A, B, C)を

```
select ([C|_], 1, C) :- !.
select ([_|D], B, C) :- BM is B-1, !,
                        select (D, BM, C).
```

としてある．この2番目の文を

```
select(A,B,C):- second(A,D),BM is B-1,
                !, select(D,BM,C).

second([_|D],D).
```

とすると，実行時間が2～3割多くかかる．何度も呼ばれる述語ではあるが，予想以上の影響が出てくる．

また，selectのアーギュメントの順による影響は前のユニファイ・リスト方式の場合と同様である．（もっと大きなビットテーブルではより大きな影響があると思われる．

(5) Fortran

ビットテーブルをリストではなく配列にしていること，およびCPU時間は参照の時間のみではなく，runコマンド全体の実行時間を測定したため，

$$T \text{ は } S \text{ に比例} + \text{ゲタ}$$

と考えられる．第5図に示す様に実測値は，その様になっている．

Sが小さい時のTの値からゲタを推定することにより

$$T = 0.067\,S + 0.20 \text{〔m秒〕}$$

となる．

4. おわりに

Prologによりビットテーブルを参照するプログラムを作成し，動作させて評価し，論理型言語を考察した．評価にあたり，4種のデータ構造のPrologプログラムおよびFortranによる同機能のプログラムの比較によった。

その結果，Prologでも，ユニフィケーションと"arg"による要素の取り出しによる方法ではかなり効率がよく，コンパイルすれば

- 速度はFortranの3倍程度遅く．
- ビットテーブルは一応百数十万ビット（Fortranの場合の約1/3）程度まで定義できる。ただし，ビットの"何番目"を整数で表すと，それが18ビットであることが直接のネックとなり，工夫なしでは13万ビット（Fortranの場合の数十分の1）程度までとなる。

  なお，大きなビットテーブルを定義する際，コンサルトされた語数としてマイナスの数が表示されたりする．正しく動作しているかは，更に確認する必要があるだろう。

リスト構造はPrologにむいたデータ構造であると言われている。しかし，本文で述べた様なせかなりの量のデータに対してランダムなアクセスをする応用については速度，記憶容量の点から，なるべくリストを避けなければならないということが定量的に明らかになった。この種の"論理"に関係ない処理も容易に記述できる様な改良（従来からの言語機能を取り込むこと）を希望したい。本実験では扱えなかったビットテーブルへの書き込みにおいては，その種の改良が一層望まれるであろう．

文献

〔1〕 Carter, L. et al, "Exact and Approximate Membership Testers", Proc. of 1978 Tenth Annual Symposium on Theory of Computing, ACM, pp. 59~65 (1978)

〔2〕 Wood, S., "Z - the Last Word in a Series of Single Letter Yale Editors", Yale Univ., pp. 34~35 (1980)

〔3〕 川合, "英文綴り検査法", 情報処理 vol. 24, no. 4, pp 507~513 (1983)

付録1. Prolog（ユニファイ・フレーム方式）ソースリスト（一部）

```
:-public test/2.
:-mode test(-,-).
test(T1,T2):-write('Loop kaisuu. (< 130000~140000) '),read(Loop),
    statistics(runtime,T1),for(I,Loop),
    /* write('test  '),write(I),nl, */
    check(I,X),     /* I banme is 1 --> X is 1 (ima, -1), else 0 */
    /* write('kekka   '),write(X),nl, */
    next(I),statistics(runtime,T2).

:-mode check(+,-).
check(I,X):-J is I + 499, K is J/500,       /* k banme no TERM ni ari */
    /* write(I),bl,write(J),bl,write(K),nl, */
    bit(K,Llist),
    L is K - 1, M is L * 500, N is I - M,   /* Llist no N bit me */
    JJ is N + 9, KK is JJ/10,       /* KK banme no atom ni ari */
    /* write(L),bl,write(M),bl,write(N),bl,write(JJ),bl,write(KK),nl, */
    arg(KK,Llist,Atom),
    LL is KK - 1, MM is LL*10, NN is N - MM,        /* Atom no NN bit me */
    Hidari is 8+(NN-1),B is Atom << Hidari,
    /* write(LL),bl,write(Hidari),nl, */
    X is B >> 17,!.

:-mode for(-,+).
for(N,N).
for(M,N):- K is N - 1,!,for(M,K).
:-mode next(+).
next(1).

bl:-write('   ').

:-mode bit(+,-).
bit(1,b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5)).
bit(2,b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5)).
bit(3,b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5)).
bit(4,b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5)).
bit(5,b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5)).
```

付録2. Prolog（タームのターム方式）ソースリスト（一部）

```
:-public test/2.
:-mode test(-,-).
test(T1,T2):-write('Loop kaisuu. (< 130000~140000) '),read(Loop),
    statistics(runtime,T1),for(I,Loop),
    /* write('test  '),write(I),nl, */
    check(I,X),     /* I banme is 1 --> X is 1 (ima, -1), else 0 */
    /* write('kekka   '),write(X),nl, */
    next(I),statistics(runtime,T2).

:-mode check(+,-).
check(I,X):-J is I + 499, K is J/500,       /* k banme no TERM ni ari */
    /* write(I),bl,write(J),bl,write(K),nl, */
    bit(Bit),
    arg(K,Bit,Llist),
    L is K - 1, M is L * 500, N is I - M,   /* Llist no N bit me */
    JJ is N + 9, KK is JJ/10,        /* KK banme no atom ni ari */
    /* write(L),bl,write(M),bl,write(N),bl,write(JJ),bl,write(KK),nl, */
    arg(KK,Llist,Atom),
    LL is KK - 1, MM is LL*10, NN is N - MM,        /* Atom no NN bit me */
    Hidari is 8+(NN-1),B is Atom << Hidari,
    /* write(LL),bl,write(Hidari),nl, */
    X is B >> 17,!.

:-mode for(-,+).
for(N,N).
for(M,N):- K is N - 1,!,for(M,K).
:-mode next(+).
next(1).

bl:-write('   ').

:-mode bit(-).
bit( bi(
      b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5),
      b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5),
      b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5),
      b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5),
      b(5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5),
```

付録3. Prolog（ユニファイ・リスト方式）ソースリスト（一部）

```
:-public test/2.
:-mode test(-,-).
test(T1,T2):-write('Loop kaisuu. (< 130000~140000) '),read(Loop),
    statistics(runtime,T1),for(I,Loop),
    /* write('test  '),write(I),nl, */
    check(I,X),     /* I banme is 1 --> X is 1 (ima, -1), else 0 */
    /* write('kekka   '),write(X),nl, */
    next(I),statistics(runtime,T2).

:-mode check(+,-).
check(I,X):-J is I + 499, K is J/500,       /* k banme no LIST ni ari */
    /* write(I),bl,write(J),bl,write(K),nl, */
    bit(K,Llist),
    L is K - 1, M is L * 500, N is I - M,   /* Llist no N bit me */
    JJ is N + 9, KK is JJ/10,       /* KK banme no atom ni ari */
    /* write(L),bl,write(M),bl,write(N),bl,write(JJ),bl,write(KK),nl, */
    select(Llist,KK,Atom),
    LL is KK - 1, MM is LL*10, NN is N - MM,        /* Atom no NN bit me */
    Hidari is 8+(NN-1),B is Atom << Hidari,
    /* write(LL),bl,write(Hidari),nl, */
    X is B >> 17,!.

:-mode for(-,+).
for(N,N).
for(M,N):- K is N - 1,!,for(M,K).
:-mode next(+).
next(1).

bl:-write('   ').

:-mode select(+,+,-).
select([C|_],1,C):-!.
select([_|D],B,C):-BM is B - 1,!,select(D,BM,C).

:-mode bit(+,-).
bit(1,[5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
       5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]).
bit(2,[5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
       5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]).
bit(3,[5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
       5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]).
bit(4,[5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
       5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]).
bit(5,[5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
       5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]).
```

付録4. Prolog（リストのリスト方式）ソースリスト（一部）

```
:-public test/2.
:-mode test(-,-).
test(T1,T2):-write('Loop kaisuu. (<= 50000) '),read(Loop),
    statistics(runtime,T1),for(I,Loop),
    /* write('test  '),write(I),nl, */
    check(I,X),     /* I banme is 1 --> X is 1 (ima, -1), else 0 */
    /* write('kekka   '),write(X),nl, */
    next(I),statistics(runtime,T2).

:-mode check(+,-).
check(I,X):-J is I + 499, K is J/500,       /* k banme no LIST ni ari */
    /* write(I),bl,write(J),bl,write(K),nl, */
    bit(Bit),
    select(Bit,K,Llist),
    L is K - 1, M is L * 500, N is I - M,   /* Llist no N bit me */
    JJ is N + 9, KK is JJ/10,        /* KK banme no atom ni ari */
    /* write(L),bl,write(M),bl,write(N),bl,write(JJ),bl,write(KK),nl, */
    select(Llist,KK,Atom),
    LL is KK - 1, MM is LL*10, NN is N - MM,        /* Atom no NN bit me */
    Hidari is 8+(NN-1),B is Atom << Hidari,
    /* write(LL),bl,write(Hidari),nl, */
    X is B >> 17,!.

:-mode for(-,+).
for(N,N).
for(M,N):- K is N - 1,!,for(M,K).
:-mode next(+).
next(1).

bl:-write('   ').

:-mode select(+,+,-).
select([C|_],1,C):-!.
select([_|D],B,C):-BM is B - 1,!,select(D,BM,C).

:-mode bit(-).
bit( [
       [5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5],
       [5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5],
       [5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5],
       [5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5],
       [5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5],
```

付録5. Fortran ソースリスト

```
      dimension ibit(10000)
c      do 10 i= 1,10000
c10    ibit(i)=0
c
c
      go to 1
c
c
      do 20 i = 1,10
      write(5,99)
99    format(1h ,3hset)
      read(5,100) j
100   format(i6)
20    call setbit(ibit,j)
1     write(5,199)
199   format(17h check(<= 350000))
c      read(5,100)i
c
       read(5,100) imax
      call time(x,y)
      write(5,900) x,y
      do 1000 ii=1,imax
c      x=ran(0) * 350000.0
c      i=x
       i=ii
      if(i .le. 0) i=1
      if(i .ge. 350000) i=350000
      call refbit(ibit,i,j)
c      write(5,200) i
c      write(5,200) j
200   format(1h ,i10)
1000  continue
c      go to 1
      call time(x,y)
      write(5,900) x,y
900   format(1h ,2a)
      stop
      end
      subroutine setbit(ibit,i)
      dimension ibit(10000)
      k=(i+34)/35
      l=ibit(k)
      kk=i-(k-1)*35
      m=0
      if (kk .ne. 1) m=(l/(2**(35-kk+1))) * (2**(35-kk+1))
      n=mod(l,(2**(35-kk)))
      ibit(k)=m+n+2**(35-kk)
c      write(5,100) i,k,l,kk,m,n,ibit(k)
100   format(1h ,2i8,o,i,3o)
      return
      end
      subroutine refbit(ibit,i,j)
      dimension ibit(10000)
      k=(i+34)/35
      l=ibit(k)
      k=i-(k-1)*35
      j=l/(2**(35-k))
      j=mod(j,2)
      return
      end
```