記 号 処 理 55－1
（1990．6．4）

Ｐｒｏｌｏｇによる
将棋プログラムのデータ構造

飯田弘之 ・ 瀬野訓啓 ・ 吉田武俊 ・ 小谷善行

（東京農工大学工学部 電子情報工学科 情報工学講座）

Ｐｒｏｌｏｇのデータ表現としてどのような構造を選択するかという問題について、将棋という具体例から、検討する。Ｐｒｏｌｏｇプログラムでの将棋の局面や着手に関するさまざまなデータ構造について、効率を測定した。将棋の局面は９×９の格子であり、そのほかに持ち駒がある。局面は手を指すごとに変化し、また優劣評価の対象となる。これらの作業に対して、主として、データを述語の引数とするかグローバルな事実とするか、および、将棋盤に対応したデータにするか駒の属性表に対応したデータにするか、という観点から実験を行なった。

Data Structure of
Shogi Playing Programs in Prolog

Hiroyuki Iida　Kunihiro Seno
Taketoshi Yoshida　Yoshiyuki kotani

Department of Electronic and Information Science
Tokyo University of Agriculture and Technology

We discuss the problem of what kind of data structure should be chosen as data description in Prolog programs. We measured performance of various data structure of positions and moves of Shogi game (Japanese chess). Any Shogi position consists of 9x9 board and possessed pieces in hand, is changed by a move and is evaluated. These operations are measured at the viewpoint of whether position data is stored in arguments of predicates or in global Prolog facts, and of whether it corresponds to a Shogi board or to a property table of Shogi pieces.

はじめに

Ｐｒｏｌｏｇのプログラムにおけるデータ表現をどうするかは重要な問題である。しかし、伝統的なプログラム言語の場合に比べて、Ｐｒｏｌｏｇはまだ歴史が浅く、データ表現に関する議論はまだ煮詰まっていない。ここでは、われわれが作成しつつある将棋システムの局面および手（とくに局面）のデータ表現の方法を通じて、この問題を検討する。

Ｐｒｏｌｏｇで将棋をプレイするシステムを記述す際、はじめに考慮しなければならないことは局面を表すためのデータ構造についてである。そこでどのように、局面を表すための効率的（と思われる）なデータ構造を見つけるかについて報告する。

一般に、局面を表すのに、

（１）引き数としてもつ方法

（２）グローバルな事実としてもつ方法

の二つに分類されるだろう。さらに、それぞれの場合に考えられる種々のデータ構造について議論する。

局面を表すためのデータ構造、その場合に考えられる局面の更新、指し手の生成などを考慮しながら考察してみる。

ここで局面の処理の基本述語を述べる。この測定で関係するのは次の３つのものである。

(1)　tsugino_kyokumen

一つの局面と一つの手とから、その手を指した後の局面を求める（局面の更新）

(2)　maeno_kyokumen

手を指した後の局面とその手とから、その手を指す前の局面を求める（局面の復帰）

(3)　kanouna_te

一つの局面からそこで指せるすべての手を生成する（指し手の生成）

このほかに、実際のプログラムでは、局面に対する（静的）評価関数が重要である。しかしこれはその中身により計算時間が異なるので、測定に含めていない。末端のノードの数からこれを計算できる。

これらの述語のアリティは測定するデータ構造に依存するものである。

指し手は局面を進めるだけでなく戻すこともできるように配慮し、

te(背番号１,移動前,移動後,生成１,
　背番号２,生成２)

背番号１：動く（または打つ）駒の番号
移動前　：動く駒の元の場所
移動後　：動いた駒が行った場所
生成１　：動く駒が生・成っている・今成った
背番号２：取られる駒の番号
生成２　：取られる駒が生だった・成だった

と表す。いずれの場合でも局面は現在の手番を含んでいる。

１.測定するデータ表現

測定する局面のデータ構造を説明する。全部で１２種類のものを用意している。次の２種類にわけられる。

1.1から1.9　：　引き数としてもつ場合
2.1から2.3　：　グローバルにもつ場合

である。

まず、データの具体的形とその性質をを示す。さらに対応するシステムの実現について述べる。

リスト　テストデータの形式

形式　1.1

```
position(Teban,
    ban(背番号,駒,11,所有者,裏表),
    ban(背番号,駒,12,所有者,裏表),
            ...,
    ban(背番号,駒,99,所有者,裏表),
    [mochogoma(sente,背番号,駒,),
     mochogoma(gote ,背番号,駒,)]
)
```

形式　1.2

```
position(Teban,
    komalist(1,駒,位置,所有者,裏表),
    komalist(2,駒,位置,所有者,裏表),
            ...,
    komalist(40,駒,位置,所有者,裏表)
)
```

形式　1.3

```
position(Teban,
    [ban(背番号,駒,位置,所有者,裏表),...],
    [mochogoma(sente,背番号,駒,),
     mochogoma(gote ,背番号,駒,)]
)
```

形式　1.4

```
position(Teban,
    [komalist(背番号,駒,位置,所有者,裏表),
    ...]
)
```

形式　1.5

```
position(Teban,
    [komalist(1,駒,位置,所有者,裏表),
     komalist(2,駒,位置,所有者,裏表),
     ...,
     komalist(40,駒,位置,所有者,裏表)]
)
```

形式　1.6

```
position(Teban,
    [                                /*先手*/
    komalist(背番号,駒,位置,裏表),...],
    [                                /*後手*/
    komalist(背番号,駒,位置,裏表),...]
)
```

形式　1.7

```
position(Teban,
    [                                /*先手*/
    komalist(背番号,駒,位置,裏表),...],
    [                          /*先手持ち駒*/
    mochogoma(背番号,駒),...],
    [                                /*後手*/
    komalist(背番号,駒,位置,裏表),...],
    [                          /*後手持ち駒*/
    mochogoma(背番号,駒),...]
)
```

形式　1.8

```
position(Teban,
    [komalist(背番号,駒,位置,所有者,裏表),
    ...],                            /*段1*/
    [komalist(背番号,駒,位置,所有者,裏表),
    ...],                            /*段2*/
        :
    [komalist(背番号,駒,位置,所有者,裏表),
    ...],                            /*段9*/
    [mochogoma(背番号,駒),...],
                          /*先手持ち駒*/
    [mochogoma(背番号,駒),...]
                          /*後手持ち駒*/
)
```

形式　1.9

```
position(Teban,
    [komalist(背番号,駒,位置,所有者,裏表),
    ...],                            /*筋1*/
    [komalist(背番号,駒,位置,所有者,裏表),
    ...],                            /*筋2*/
        :
    [komalist(背番号,駒,位置,所有者,裏表),
    ...],                            /*筋9*/
    [mochogoma(背番号,駒),...],
                          /*先手持ち駒*/
    [mochogoma(背番号,駒),...]
                          /*後手持ち駒*/
)
```

形式　2.1

```
teban(Teban)
komalist(背番号,駒,位置,所有者,裏表).
...
komalist(背番号,駒,位置,所有者,裏表).
```

形式　2.2

```
teban(Teban)
ban(背番号,駒,位置,所有者,裏表).
...
ban(背番号,駒,位置,所有者,裏表).
mochogoma(sente,背番号,駒,).
mochogoma(gote ,背番号,駒,).
```

形式　2.3

```
teban(Teban)
komalist(背番号,駒,位置,所有者,裏表).
...
ban(位置,背番号).
...
sentemochigoma(背番号).
...
gotemochigoma(背番号).
...
```

表1　テストデータの性質対照表

データ構造番号	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.1	2.2	2.3
駒属性表		o								o		o
盤形式ﾃﾞｰﾀ	o										o	o
ﾘｽﾄ形式駒属性表				o	o	o	o					
ﾃﾞｰﾀを先後で分割						o	o					
ﾘｽﾄ形式の盤ﾃﾞｰﾀ			o									
ﾘｽﾄ形式の持ち駒	o		o				o				o	o
ソートされている	o	o			o							

まず、データ構造１．１を例にして、実現と処理の概略を述べる。さらにそのあと、個々のデータ構造に対する実現の方式を述べる。

指し手を生成するためには、まずその局面がどちらの手番であるか知らなければならない。そのためには、arg/3を用い

arg(1,Position,Teban)

で現在の手番を参照する。次にTeban側の駒を参照するため、

arg(M,Position,ban(S,X,K,Teban,F))

ただし　(1<M<83)

を用い、それぞれの駒に応じた可能な指し手を生成すればよい。その時、駒の動きはバイアスで与えられるようになっていて、例えば金は

'KI'(先手,(0,1))

((0,1)の最初の引数は飛び効きとの区別するものである）のように表されている。その駒の現在位置にこれを加えて移動先を決めることになる。そしてルール判定をする。移動先の位置が盤の中であって、行き所があって、しかもそこに味方の駒がいないことを確認しなくてはならない（厳密にはその駒の動きによって自王が取られないかどうかもチェックする必要があるが、本稿ではこれに関しては配慮していない）、はいずれの局面の表し方でも同じである。このために移動先をTXとすると、TXの位置に関する引き数は

（TX-8-TX/10）番目にあることがわかっているので,これをNとすれば

arg(N,Position,ban(_,_,TX,Teban1,_))

によってTeban1とTebanとが等しくないことを確認すればよい。

さらに、持ち駒を打つときは

arg(83,Position,HandList)

によって持ち駒のリストを取り出し、その中から駒を取り出し打てるかどうか調べればよい。ただし、いずれの局面の表し方でも同様に、持ち駒の種類による重複をなくさなければ非常に効率の悪いプログラムになる。これはsetof/3を使うことにより実現している。あとは盤上の位置１１から９９までその駒がルール上打てるかどうかを調べればよい。このあとは、やはりarg/3を用いてその位置が空であり、駒の種類によって禁手（二歩と行き所の無いところへ駒を打つことと打ち歩詰め、ただし、本稿では打ち歩詰めに関しては配慮していない）のチェックをすることになる。

次に指し手による局面の更新は、N番目の引き数を取り出してそれを更新してさらにM番目の引数を取り出してそれを更新するといった操作が必要となる。ここではchange_arg/5を使ってこれを行っている。すなわち

change_arg(N,Position,X,Position1,Y)

によってPositionをPosition1に更新する。ただし、XはPositionのN番目の引数であり、XをYに入れ換えたものがPosition1である。これを適宜使えば局面の更新を行うことができる。

1.1の場合

盤上の一つ一つのマス目に関する情報は、ファンクタの形式、

ban(背番号,位置,駒の種類,所有者,駒の状態)

で表される。

局面データの第１引数に手番、第２引数以下に

上記のものが位置順（１１から９９）に並んでいる。この場合、持ち駒に関する引数が必要で、そのために

[mochigoma(所有者,背番号,駒の種類),....]

として持ち駒をリストで表すことにする。したがって、局面は８３個の引数の複合項の構造で表される。ただし、任意の位置に駒がいない場合は背番号は0,駒の種類と所有者と駒の状態はそれぞれnilとする。背番号とは一つ一つの駒を指定する数値であり、個々の駒に１から４０までの数が対応している。

1.2の場合

それぞれの駒に関する情報が

komalist(背番号,位置,駒の種類,
所有者,駒の状態)

で表され第１引数に手番、第２引数以下はkomalist/5が背番号順に並んでいる。この場合は持ち駒のための引数は持たない。したがって、局面は４１引数の複合項の構造で表されてる。

1.1と大きく違っているのは、まず指し手を生成する際のルール検定に関してである。移動先をTXとすると盤の中であることを知るために

TX>10,TX<100,TX mod 10 =¥= 0

を実行するようになる。

1.3の場合

1.1の場合と似ているがban/5がすべてリストの要素となっている。それゆえ、引き数３の複合項の構造で表されている。２番目の引き数は常に長さが８１のリストである。

指し手を生成するには、局面がposition(Teban,Ban,HandList)で表されているので

member(ban(J,K,X,Teban,F),Ban)

でTeban側の駒をさがし、移動先TXに味方の駒がいないかを確認するにはTXは２番目の引き数の(TX-9-TX/10)番目にあるのでこれをNとして、N番目の要素を取り出すためにnth1/3[1]を使って

nth1(N,Ban,ban(TJ,TK,TX,Teban1,TF))

で移動先のデータを取り出しTeban1とTebanが等しくないことを確認すればよい。

局面の更新では、ここでX1からX2へ移動したと仮定するとリストの中の(X1-9-X1/10)番目の要素と(X2-9-X2/10)番目の要素のそれぞれを適切に置き換えればよい。これを実現するためにN番目の要素をBNとすると

append(Pre,[BN|Bs],Ban)

として分割しその後BNをBN1に置き換えてBanをBan1に更新する。すなわち、

append(Pre,[BN1|Bs],Ban1)

のようにする。これを適宜行えば局面の更新ができる。

1.4の場合

1.2の場合と似ているがkomalist/5がすべてリストの要素となっている。それゆえ、引き数２の複合項の構造で表されている。２番目の引き数は常に長さが４０のリストである。

指し手を生成するには、局面がposition(Teban,KomaList)で表されているので

member(komalist(J,K,X,Teban,F),
KomaList)

でTeban側の駒をさがし、移動先TXのルール判定をするには、KomaListにはkomalist/5がランダムにあるので1.2の場合のように盤の中であるかどうかを確認してから、TXの位置に味方の駒がないかどうか判定するために

memberchk(komalist(TJ,TK,TX,TB,TF),
KomaList)

を使って実現している。

局面の更新では、リストの中味がランダムなのでselect/3を使ってこれを実現している。

1.5の場合

1.4の場合でkomalist/5が背番号順に並んでいる場合である。引き数２の複合項の構造に変わりはない。２番目の引き数はやはり長さが４０のリストである。

1.4との違いはおもに局面の更新のときにある。すなわち、常に背番号順に並べるために1.3のときに使った方法を使ってこれを実現するようになる。背番号Ｊの駒を動かすことを仮定すると、２番目の引き数のＪ番目の要素を置き換えることになるからである。

1.6の場合

1.4の場合で２番目の引き数を先手と後手の２つに分けた場合である。したがって、局面は引き数３の複合項の構造で表されている。この場合、２番目と３番目の引き数はリストであるがそれぞれ

の長さの和は常に４０であることは明らかである。

先手と後手の場合分けを適切に行えば1.4の場合とほぼ同じように指し手の生成と局面の更新を行うことができる。

1.7の場合

1.6の場合で持ち駒を後手方と先手方とに分けた場合である。したがって、局面は引き数５の複合項の構造で表されている。

1.8の場合

将棋盤を横から見てそれぞれの行に分けた場合である。したがって、２番目のリストは長さが９である。この場合、持ち駒は先手と後手をそれぞれリストで表す。これによって、局面は引き数４の複合項の構造で表されている。２番目のリストのそれぞれの中味は、その行に駒があるかどうかによってkomalist/5がランダムに入れられている。

指し手の生成は、移動の場合はそれぞれの行の味方の駒をさがせばよい。持ち駒は別にリストになっているので自明である。ただこの時、移動先あるいは打とうとする位置TXについて判定するためには、２番目のリストをSL2とすると M=TX mod 10 としてSL2のM番目の要素Yを取り出して、リストYの中にTXの位置にいるkomalist/5をさがすことになる。

局面の更新は、２番目のリストの各要素を指し手によって置き換えることが中心になる。これは1.3の場合とほぼ同様にして記述した。

1.9の場合

将棋盤を縦に見てそれぞれの列に分けた場合である。1.8の場合に M=TX mod 10 としたのを M=TX / 10 とすればほぼ同様にできる。

2.1の場合

このあとは局面をグローバルにもつ場合である。これまでと大きく違う点の１つとして局面を戻す述語をここで定義しなければならないことである。ミニマックス法によって木の探索をする際、これが必要になる。これまでさほど気にしなかったが、ここではkomalist/5の５つの引き数の順番がかなり影響を及ぼしてくる。これは、ユニフィケーションの動作に関連している。第一番目の引き数はインデキシングにより迅速にこれを行えるよう、たいていのPROLOGの処理系はできている。それゆえ、第一番目の引き数に何をもってくるかが特に重要な問題で、ここでは背番号を第一番目とした。あとの2.3の場合と比較するのに良いと思うからである。

これまでとの記述上の違いとして、グローバルにもつ場合はretract/1, assert/1を使って局面の更新を行い、指し手の生成ではそれらに適切にユニフィケーションさせることによって実現できる。例えば、手番を替えるのは

```
retract(teban(Teban))
```

として現在のteban(Teban)を消して、TebanをTeban1に変えて（先手なら後手に）

```
assert(teban(Teban1))
```

とやればできる。

この場合、一個のteban/1と４０個のkomalist/5で合計４１個の事実で局面を表していて、本稿にて局面をグローバルにもつ場合で一番少ない個数である。

2.2の場合

この場合、一個のteban/1と８１個のban/5そしてmochigoma/3のそれぞれの事実で局面を表している。ここではban/5の引き数の順番を座標位置を第一とした。

指し手の生成と局面の更新の記述は、2.1の場合と似たものとなる。指し手の生成のときなど移動先TXの情報をban(TX, Sebango, Koma, Teban, Nari)としてユニフィケーションさせることによって迅速に知ることが可能である。このとき明らかに、盤の中であればいずれかにユニフィケーションするがそうでない場合は失敗する。

2.3の場合

この場合、１個のteban/1と４０個のkomalist/5と８１個のban/2そしてsentemochigoma/1とgotemochigom/1のそれぞれの事実で局面を表している。事実として常に１２２個以上存在することになる。komalist/5の引き数は背番号を第一番目とし、ban/2の引き数は座標位置を第一番目とした。

これによって2.1と2.2のそれぞれの長所が活かされると考えられる。例えば、位置Xについての情報を知りたいとしよう。これは指し手の生成などでよくあることでまず、

```
ban(TX, Sebango)
```

でban/2にユニフィケーションさせる。このときそこにどちらかの駒があればSebangoは１から４０ま

での数となり、そうでないときは0となるようにしてある。ユニフィケーションに失敗することは盤の中でないことを意味する。そこで次に、Sebangoが0でないとき

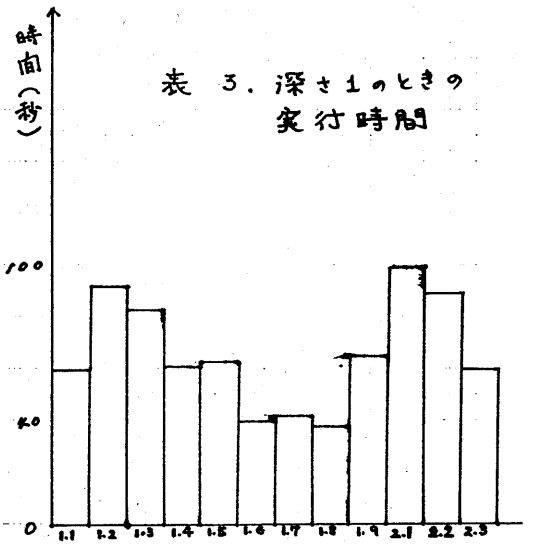
komalist(Sebango,Koma,TX,Player,Nari)

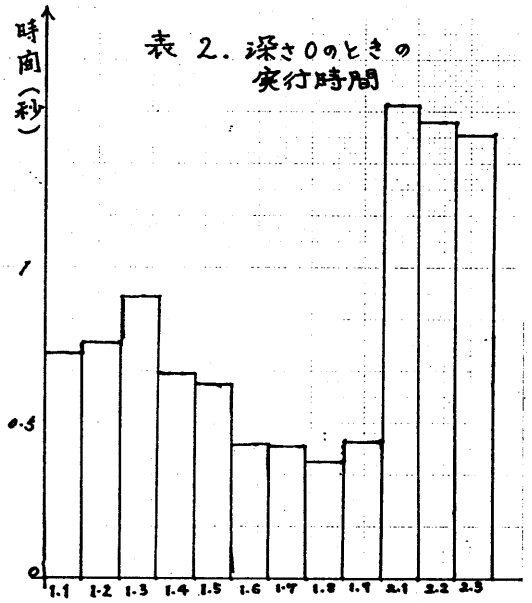
にユニフィケーションさせる。これらの動作によって迅速に目的は達せられる。

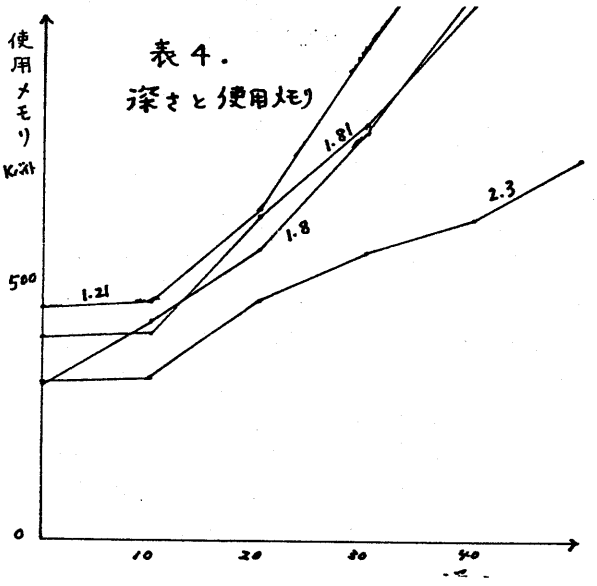
2.効率測定実験

データ構造の形式1.1から2.3までの局面の表し方のそれぞれの場合について以下のように実験する。つまり、minimax法（局面内部の並び方に依存しない）を使って指し手を求めるモジュールを作り、深さを変えながらそれぞれの場合について指し手を求める時間とメモリー使用について計測する。テストの局面として3局面を用意し局面に依存し過ぎることないように平均をとって考察する。

実験の結果を表2と表3に示す。







３．実験で得られる知見

それぞれの局面の表し方における、実行時間を表したものである。ただし、表２は深さ０の場合であり、表３は深さ１の場合である。

当初の予想では1.1の場合が一番多く実行時間を要し、2.3が一番少ないのではないかと思っていた。筆者のこれまでのPROLOGでのプログラム記述の経験からこのように予想していた。ところが、実際は表にあるようにだいぶ違った結果が得られた。

1.1と1.3の結果を比べるとリスト処理をしている1.3の場合が実行時間が多い。これによって1.1と1.2のかなり多く引き数に持つ場合でも有力な局面の表し方であると考えられる。また、1.1の方が1.2よりも実行時間が少ないのは、指し手の生成の際、移動先や持ち駒を打とうとするマス目が空であるか味方の駒がいないか等の判定をする時に1.2が効率のよくないところが原因であると考えられる。そこで、1.2の場合に加えてban/2をグローバルにもつ複合的な局面の表し方（1.21とする）も有力であると思われる。

1.3から1.9まではそれぞれにリスト処理が用いられているが、全体的に1.8の場合が良い結果を示している。これは４０個の要素を１つのリストでもつより９個のリストでもつ方が効率がよいからであろう。1.8の場合にban/2をグローバルにもつ複合的な局面の表し方（1.81とする）でどう変わるか興味のあるところである。

2.1から2.3の場合は表２から明らかに良い結果は得られなかった。局面を戻さなければならないので実行時間がよけいかかっているのだろう。ただし、表３から2.3の場合はかなり期待のもてる局面の表し方であると考えられる。深さが大きくなればさらに、良い結果が望めそうである。

以上のことから、1.8の場合を筆頭にいくつか有力であることがわかった。そこでさらに吟味するために、1.21、1.8、1.81、2.3の４つについて深さとメモリーについて実験を行った。探索木の１つの末端でのメモリーの様子を深さを０から５０まで変えながら観測してみた。その結果は表４である。この表から明らかに、深さが１０以上の先読みをする場合は2.3が最も効率がよい局面の表し方であると断定できる。但し、将棋の性質上先読みは必要だがそれほど深く読むことは常に良い戦略とはいえない。実際は４－５手の先読みで十分なことが多いからである。

５．まとめ

Ｐｒｏｌｏｇで思考問題や思考ゲームなどを記述する際、局面を引き数としてもつことは「よいプログラム表現」であると思われる。この類のプログラムでは先読みが重要であるが、実験から明らかなように、局面を戻さなくてよいのは大きなメリットの１つである。それよりも上げられる利点はＰｒｏｌｏｇを思考の道具として考えた場合、局面を引数としてあたかも変数の１つとして扱うことができるのでプログラミングがしやすいのである。しかし、引数として局面を表すことが可能かどうかは先読みの深さとの兼ね合いで決めることになるだろう。

Ｐｒｏｌｏｇにおける配列の問題も大事なことである。一般に配列のN番目の要素にアクセスする際、Nの大きさに比例して時間がかかる。しかし、本稿では1.1の場合のようにarg/3が組込み述語でかなり速く動作するので、この問題については特別な対処はしていない。

なお、本稿では1.1から2.3のそれぞれの場合について議論したが、そのほかにもいろいろな（例えば引数とグローバルの複合）形式があって、良い結果が得られるかも知れない。さらに、記述上の問題で、本実験よりも効率よく動作することも考えられるので、そのことをここに付記する。

参考文献

The Art of Prolog Programming （松田利夫訳、Ｐｒｏｌｏｇの技芸、共立出版）Leon Sterling Ehud Shapiro.