BASICによるPROLOGシミュレーション

吉村　卓

PROLOG Simulation System on BASIC Language

Takashi YOSHIMURA

（平成3年10月30日受理）

1. はしがき

第5世代コンピュータ用プログラミング言語としてPROLOGが採用されて以来，人工知能研究用言語としてPROLOGは非常に多くの分野で活用されつつある。PROLOGの持つデータベースの作成と検索機能，規則の表現と書き換え機能などが，論理的機能とあいまってAI研究の強力な道具となり得るからである。

しかし今まで，逐次処理向けプログラミング言語としてのFORTRANや，BASIC, C, PASCALなどを扱ってきた者にとって，PROLOGの動作原理は理解しにくい面がある。それ故，PROLOGシステムの構造や動作原理の理解のために，BASICによるそのシミュレーションソフトを作成して，入門段階の学習に利用してみよう。

2. PROLOGの概要

2. 1. 項

PROLOGのデータ構造はすべて項と呼ばれる。項には単純項と複合項がある。単純項には定数（定項ともいう）と変数（変項ともいう）がある。

定項はまたアトムとも呼ばれる。アトムはPROLOGのデータの最小単位であり，任意の長さの文字列から成る。ただし，数値（整数および実数）として扱われる数字文字列には接頭辞として引用符“’”をつける。引用符のない数字文字列は単なる文字列とみなされる。

変数は，接頭辞として星印“*”で始まる文字列で表される。

複合項はpをアトム，$t_1,t_2,\cdots,t_n$をアトム，変項あるいは複合項として，p（$t_1,t_2,\cdots,t_n$）の形式をもつ。

複合項の特別なものとしてリストがある。項の個数が不定のとき用いられるデータ構造である。項を“〔“と”〕”でくくって表す。リストの要素としてリストを含めることができる。すなわち，リストとは

(1) 空リスト [] はリストである。

(2) X,Yを項とするとき，[X | Y] はリストである。

の2条件で決まるものである。

リストの表記には，$A,A_1,A_2,\cdots,A_n,B,C$

を項とするとき

(1) [A | []] を [A] と書く。

(2) M= [B | L], L= [$A_1,A_2,\cdots,A_n$ | C] のとき，Mを [$B,A_1,A_2,\cdots,A_n$ | C] と書く。

2. 2. 述語

PROLOGのプログラムの基本要素は述語である。述語はいくつかの項の並びであり，次の形式をもつ。

述語名（$項_1,項_2,\cdots,項_n$）

すなわち，述語は上述の複合項の形をもち，ある対象の性質あるいはいくつかの対象間の関係を表現するものである。

2. 3. 節

プログラムの個々の文を節という。節は次の形式をもつ。

(1) P：$-Q_1,Q_2,\cdots,Q_m$

(2) P：−

(3) ？$-Q_1,Q_2,\cdots,Q_m$

ここに，$P,Q_1,\cdots,Q_m$は述語であり，“,”は論理積を表わす。形式(1)，(2)はホーン節と呼ばれ，形式(3)はゴール節と呼ばれる。ホーン節はネック記号：−を挟んで頭部と本体に分けられるが，形式(2)のように本体がなく頭部だけのものもある。頭部だけのホーン節は事実を表わし，頭部と本体をもつホーン節は規則を表わす。ゴール節は頭部がなく本体のみの節である。

このように，PROLOG の文は複合項の特別なものとみなされ，プログラムとデータとが同じように扱えて都合がよい。

PROLOG のプログラムはホーン節の集合であり，ホーン節ベースと呼ばれる。ホーン節ベースで同じ述語を頭部にもつホーン節全体は，その述語にプログラム記述者の意志が反映されたものであり，その述語の定義という。そして個々のホーン節はその定義を構成する主張と呼ばれる。

ホーン節ベースをロードしておくと，一つのゴール節を入力した時点で完全なプログラムとなり，PROLOG システムの処理機能が働き始める。

2. 4.　PROLOG の処理機能

2. 4. 1.　探索的処理機能

PROLOG の処理機能の一つに事実の表現とその検索機能がある。PROLOG のプログラムはホーン節の集まりであるが，これ自身では処理は実行されない。プログラムを実行させ，所用の目的を達成するには，ゴール節の入力が必要である。事実のみから成るプログラムに対して，ゴール節の入力による実行の起動はデータベースへの検索と同じである。ゴール節の入力，すなわちデータベースへの質問に対し，システムはそのゴール節を満足すべき定項を探索する。ところで，ゴール節入力者は，システム応答として Yes か No を期待しているのではなく，Yes となるべきすべての可能な解を期待している場合が多い。システム応答 Yes！？＝ に対し，セミコロン（；）を入力することにより複数の答を求めることができる。この場合の内部処理は後述のバックトラックを利用して行なわれる。

2. 4. 2.　逐次的処理機能

PROLOG システムは論理的プログラム言語であるが，実際のプログラミングにあっては，PROLOG の節を通常のプログラミング言語における手続きとして解釈し，逐次型であることを意識しなければならない場合もある。すなわち，プログラムの記述の仕方によってはホーン節ベースにおけるホーン節の順序がシステム応答に影響することがある。

ホーン節の頭部Pは手続きの名前を表わし，本体 $Q_1,Q_2,\cdots,Q_m$ は他の手続き $Q_1,Q_2,\cdots,Q_m$ をこの順に呼び出すことを意味する。ゴール節は頭部をもたないホーン節，すなわち無名の手続きと解釈される。一つのプログラムの中に無名の手続きが二つ以上あると区別がつかないから，ゴール節はプログラムの中に唯一つだけ現われる。プログラムが走り始めるときは，この無名の手続きが最初に実行されるのである。

2. 4. 3.　論理的処理機能

PROLOG のプログラムの実行は我々が日常行なっている推論を機械的に行なっていると考えることができる。いわゆる三段論法である。大前提は一般規則を，小前提は個々の事実をそれぞれ述べている。ただし，PROLOG の場合は結論が先にあってそれを満たすために推論が起動される。結論からものごとが始まるので，これを逆向き推論という。これに対し，三段論法は前向き推論といわれる。

前向き推論は，事実と規則を組合わせることによって次々に新しい事実を推論していくのに対し，逆向き推論は結論が与えられたときにそれを逆に事実まで引き戻す。つまり，A を証明したいとき，A：－Bという規則があれば，これは「A を証明するには，B を証明すればよい」という意味であるから，B を証明すればよいことになる。それには，B：－C という規則があれば，C を証明すればよいことになる。この過程を Z：－という事実に出会うまで続ければよい。PROLOG ではこの逆向き推論を用いている。

2. 5.　PROLOG の推論メカニズム

プログラムの実行はシステムにゴール節を入力したときに始まり，入力された節を肯定あるいは否定することによって一つの評価を終わる。評価の終了したゴール節はシステム内から消滅し，次の評価すべきゴール節の入力待状態に戻る。

PROLOG の評価は，一つのホーン節では左から右への向きに論理積の評価がなされる。一方，ホーン節とホーン節は論理和であり，上から下への順序で評価される。しかし，論理和であるから，節と節とは論理的に同等の優先順位であり，どれかのホーン節が肯定されれば一つの解が得られたことになる。

PROLOG の評価のメカニズムの基本は統一化（ユニフィケーション）と後戻り制御（バックトラック）である。

統一化は変項に定項，変項あるいは複合項を代入する操作であり，後戻り制御は入力ゴール節に対しすべてのホーン節を探索する過程で，ゴール節とホーン節の統一化を試みる順番を制御することをいう。

統一化の過程においては，ゴール節の最左端の述語とホーン節の頭部の述語の相対する項の形（パター

ン）を同一化する。この同一形化をパターンマッチングという。すなわち，パターンマッチングでは二つのパターンをもってきたときに，それらに含まれる変項に適当な値を代入することによりそれらが同じ形になるとき，その二つのパターンはマッチするという。

統一化で注意すべきことは，述語名，変項，定項のそれぞれの語には，PROLOG自体が規約した特別な意味がないことである。プログラム記述者がある観点から対象あるいは対象間の関係を見いだし，それに名前をつけたにすぎない。意味的解釈はプログラム記述者に帰属する事柄であって，統一化は単なる構文上の同一形化を機械的に行なうにすぎない。パターンマッチングの規則を以下に示す。

(1) 変項は何とでもマッチする。ただし，同じ変項は同じものとしかマッチしない。
(2) 定項はそれと等しい定項あるいは，まだ値をもっていない変項としかマッチしない。
(3) 複合項はそのすべての要素がマッチするときだけ他の複合項とマッチする。もちろん，まだ値をもっていない変項とはマッチする。
(4) 以上の外には二つのパターンはマッチしない。

2. 6. バックトラックとカットオペレータ

ゴールに辿り着くのにいくつかの選択枝があるとき，とりあえずそのうちの一つ（上から順に探して，最初に見つかったもの）を選んで進み，もしどこかで行き詰まったら後戻り（バックトラック）する。バックトラックで戻ってきたときには，当然のことであるが，別の枝を選ばなければならない。再び古い失敗した枝を選んだのでは，いつまでもそこから抜け出せない。このような誤まりを防ぐためには，各分岐点において枝の選択の履歴を保持しなければならない。これがバックトラックを実現する上で重要なポイントになる。このように，PROLOGには，バックトラックの機構が基本メカニズムとして組み込まれているので，普通にプログラムを書くと，評価に失敗したとき自動的にバックトラックの機構が働く。しかし，時にはバックトラックの機構が不要になることもある。そのためにカットオペレータというものが用意されている。これはある範囲のバックトラックを制限するもので，！で表わす。！は左から右への一方通行である。すなわち，カットを含む節を選択した分岐点以降のすべての分岐点の履歴を破棄して，あたかも全部の選択が失敗したかのように履歴を書き直すのである。

3. PROLOG. BASのシステム構造

3. 1. グローバル変数

(1) ホーン節ベースとして文字型配列H$ () を用いる。個々の配列要素に入力順にホーン節を格納する。

(2) 探索空間用に文字型配列S$ () を当てる。スタック的に用いられ，評価中に内容が動的に変化する。入力されたゴール節はS$ (0) に格納され，統一化に成功して内部生成されたゴール節は順次上方に向かってS$ () に一時的に保存される。また，統一化に成功したとき変項とその代入項の対が，最後の要素から下方に向かって順次一時的に保存される。

(3) 制御スタックとして配列CG (), CL (), CH () を用意する。これらはCG (i), CL (i), CH (i) が1組となって，後戻り制御に用いられる。CG () には内部生成されたゴール節の探索空間上の位置が記録される。

CL () にはそれぞれのゴール節が探索空間上でいくつの要素に分割されているかが記録される。

CH () にはそれぞれのゴール節の最左端の述語について，どのホーン節まで統一化が試みられたかが，すなわち，次に統一化を試みる相手のホーン節の番号が記録される。

制御スタックは，評価ごとに内部生成されるゴール節とともに上方に向かって成長する。

(4) レジスタ類

ULレジスタは制御スタックの最新位置を指すポインタである。ゴール節が探索空間に記録されるごとにULの値は＋1され，後戻り制御のときは－1される。

Uレジスタの値は，ゴール節の変項を内部表現形式に変換するときに接尾辞として使われる。初期値は0である。原則としてULレジスタの値と同じであるが，カット！が現われたときに限りULレジスタとUレジスタの内容は一致しなくなる。すなわち，カット制御により探索空間に保存されていたいくつかのゴール節が強制的に削除されるのでULレジスタの値はUレジスタの値よりも小さくなる。

NHCレジスタはホーン節ベースに記録されている最後のホーン節の位置を指すポインタである。

ATレジスタは内部生成された最新ゴール節の探索空間上の位置を指すポインタである。ゴール節の述語とホーン節の述語との統一化が成功すると変項と代入項の対が探索空間内に下方に向かって一時的に保存されるが，CSSレジスタはこの対の探索空間

上の位置を指すポインタである。なお，ATの値は上方に向かい，CSSレジスタの値は下方に向かって成長するが，これらが衝突すると探索空間が不足した状態となり，それ以降評価ができなくなる。

3. 2. プロセス構造

PROLOG. BASのシステムプログラムは図1ならびに図2に示すようなブロックとプロシージャから成る。各ブロック，サブルーチンならびにプロシージャの処理の内容を以下に概説する。GOSUBサブルーチンを多用したのは，以下の説明からも分るように，サブルーチンからの戻り先が複数あるからである。

```
DECLARE SUB INPUT.CHECK (FLAG%)
DECLARE SUB STORE.CLAUSE ()
DECLARE SUB UPDATE.CLAUSE ()
DECLARE SUB EXEC.COMMAND ()
DECLARE SUB VARIABLE.RENAME (A$)
DECLARE SUB DAINYU (ST%, A$)
DECLARE SUB GET.PREDICATE (A$, AM1%, AM2%)
DECLARE SUB GET.TERM (A$, TM1%)
DECLARE SUB PRINT.MSG (MSG$)
DECLARE FUNCTION DIGIT% (A$)
```

図1

```
INPUT.CLAUSE:
  GOSUB CLAUSE.INPUT
'
' START OF SEARCH
UL = 0: U = 0: CG(0) = 0: CL(0) = 1: CH(0) = 0
CALL VARIABLE.RENAME(S$(0))
'
SEARCH.GOAL:
  GOSUB GOAL.SEARCH
  GOSUB UP.PREDICATE
  GOSUB EXEC.PREDICATE
'
SEARCH.HORN:
  GOSUB HORN.SEARCH
'
BACKTRACKING:
  GOSUB BACKTRACK
'
UNIFICATION:
  GOSUB UNIFICATE
  GOSUB MATCH
'
SUBSTITUTION:
  GOSUB SUBSTITUTE
'
END
```

図2

(1) INPUT. CLAUSE ブロック

ホーン節ならびにゴール節の入力を行なう。S$(0)に読み込まれる。プロシージャINPUT. CHECKを呼び出し，入力文中の不要な空白をとり除くなどのチェックをする。

文中にネック記号：－を含むものはホーン節であるから，SUBプロシージャSTORE. CLAUSEを呼び出しホーン節ベースに登録する。

文頭が数字のものはホーン節の修正であるから，SUBプロシージャUPDATE. CLAUSEを呼び出し，文の削除あるいは挿入を行なう。

文頭が？－のものはゴール節であるから，ブロックを抜け出し探索を開始する。

それ以外はシステムコマンドであるから，SUBプロシージャEXEC. COMMANDを呼び出しそれぞれのコマンド処理を行なう。コマンドには，プログラムリスト表示用コマンドにlistならびにllist，ファイル用コマンドにsaveならびにload，ホーン節ベース管理用コマンドにnew，デバッギング用コマンドにtrace，システム終了コマンドとしてquitなどが用意されている。

(2) SEARCH. GOAL ブロック

サブルーチンGOAL. SEARCHを呼び統一化試行ゴール節の有無を調べる。ゴール節が有ればサブルーチンUP. PREDICATEを呼び，ゴール節が無いときは入力ゴール節が肯定されたのであるから、システム応答Yesに続きシステムより別解を求めるか否かを問われるので，応答する。“y”ならBACK-TRACKINGブロックへ行き，“n”ならINPUT. CLAUSEブロックへ戻る。

UP. PREDICATEサブルーチンではSUBプロシージャGET. PREDICATEを呼び出し，統一化試行ゴール節の最左端述語を取り出す。

続いてサブルーチンEXEC. PREDICATEを呼び，この最左端述語が組込み述語かを調べ，組込み述語のときは対応するそれぞれの処理を実行する。

組込み述語としては，入出力述語のinpならびにprt，制御用述語failならびに！，算術演算述語add，sub，mul，div，mod，数値比較述語eqn，nen，ltn，len，gtn，gen，文字比較述語eqa，nea，lta，lea，gta，geaが用意されている。

(3) SEARCH. HORN ブロック

統一化試行ゴール節の最左端述語が組込み述語でないときはサブルーチンHORN. SEARCHを呼び，GET. PREDITE　SUBプロシージャを用いてこれと同じ述語名を頭部にもつホーン節を探す。統一化を試行すべき相手のホーン節が見つかったときはUNIFICATIONブロックへ，見つからないときはBACKTRACKINGブロックへそれぞれ進む。

(4) BACKTRACKINGブロック

統一化に失敗したとき，あるいは入力ゴール節を肯定できて別解を求めようとするときこの機能が用いられる。サブルーチンBACKTRACKで制御スタックを調べ，一つ手前のゴール節があればSEARCH. GOALブロックに戻り、なければシステム応答Noを出力してINPUT. CLAUSEブロックへ戻る。

(5) UNIFICATIONブロック

サブルーチンUNIFICATEを呼び，VARIABLE. RENAME SUBプロシージャを使ってホーン節頭部述語の変項に接尾辞を付加する。続いて，ゴール節の最左端述語とホーン節頭部の述語とのマッチングを試みる。その際，探索空間に変項と代入項の対を保存する。

マッチング成功ならばサブルーチンMATCHを呼び，VARIABLE. RENAMEプロシージャを使って，ホーン節本体の述語の変項に接尾辞を付加するとともに，ホーン節の本体の後にゴール節よりこの述語を除いた残りの部分を接続した新しいゴール節を内部生成し，探索空間に保存する。

(6) SUBSTITUTIONブロック

統一化に成功したとき，サブルーチンSUBSTITUTEを呼び，UNIFICATIONブロックで探索空間に保存しておいた変項と代入項の対を利用し，最新内部生成ゴール節中の変項を代入項で置き換える。これには，SUB DAINYUプロシージャが用いられる。

(7) GET. PREDICATEプロシージャ

文より一つの述語を取り出すために用いられる。すなわち，節中の述語の位置がこのプロシージャにより得られる。サブルーチンUP. PREDICATE, HORN. SEARCHなどから呼び出される。

(8) GET. TERMプロシージャ

述語から述語名の部分，個々の項，リストの部分などを取り出すために用いられる。すなわち，述語中のそれぞれの位置がこのプロシージャにより得られる。サブルーチンUNIFICATEなどから呼び出される。

(9) VARIABLE. RENAMEプロシージャ

ゴール節またはホーン節の中で“＊”で始まる項があれば，この語の後に接尾辞“.n”を付加する。nはレジスタUの値をとる。サブルーチンUNIFICATE, MATCHなどから呼び出される。

(10) DIGIT FUNCTIONプロシージャ

引数の文字が数値か否かを判定する。サブルーチンCLAUSE. INPUTより呼び出され，数値だったら論理値Tを，数値でなかったら論理値Nをそれぞれ返す。

4. 応用

上述の機能と構造を持つ自作のPROLOG. BASシステムは15KBに満たない小さなものである。このように小規模のシステムでもどれ程の仕事が出来るかをみるために，若干の数学の問題に適用してみよう。

(1) 図3に示すプログラムcomb（N, R）は相異なるn個のものからr個取り出す組合せをすべて作り出す述語である。述語generate_e（N, E）は与えられた自然数Nに対し，単位順列（1, 2,…,N）を生成する。また，n個の要素からなるリストXからr個の元を取り出して作ったリストをAとするとき，memberr（A, X, R）は一つの組合せを具体的に与える述語である。そして，member（A, List）は項Aが与えられたリストListの要素かどうかを判定する述語である。

```
comb(*N,*R):-generate_e(*N,*X),memberr(*A,*X,*R),
             prt(*A),fail
comb(*A,*B)
generate_e(*N,*E):-gene_e_aux(*E,*N,[])
gene_e_aux(*L,'0,*L):-!
gene_e_aux(*Result,*N,*L):-sub(*N,'1,*N1),!,
    gene_e_aux(*Result,*N1,[*N|*L])
memberr([*A],*X,'1):-member(*A,*X)
memberr([*A|*LA],[*A|*Y],*R):-gtn(*R,'1),sub(*R,'1,*R1),
                              memberr(*LA,*Y,*R1)
memberr(*A,[*X|*Y],*R):-gtn(*R,'1),memberr(*A,*Y,*R)
member(*X,[*X|*A])
member(*X,[*A|*List]):-member(*X,*List)
```

図3

(2) 有限集合Sから有限集合Tへの写像f：S→Tに対し，fのグラフは｛(y,x)｜y＝f(x), xはSの要素｝と表わされるので，(y,x) 全体の作るリストFをもって写像fを表現したものと考える。ただし，いつもこのように書くのは面倒なので，n個の元からなる集合Sはリストで［1, 2, …, n］と表わされているとして，f(1), f(2), … f(n) を順に並べたリストで写像fのグラフを表わすものとする。すなわち，定義どおりなら写像fはリストを用いて［(x, 1), (y, 2), (z, 3) ,…］と書かれるが，これを簡単に［x, y, z,…］とTの元よりなる長さnのリストで表わすことにする。

図4に示すプログラムlist_all（T, F, N）はn個の元からなる集合Sから同じくn個の元からなる集合Tへの写像をすべて与える述語である。

```
list_all([],*A,*B):-!,fail
list_all(*T,[*A],'1):-member(*A,*T)
list_all(*T,[*A|*LA],*N):-gtn(*N,'1),sub(*N,'1,*N1),
                member(*A,*T),list_all(*T,*LA,*N1)
```

図4

(3) 写像f：S→Tが全射とはfの像すなわちf(S)がTに一致することである。また，fが単射とはSの相異なる2元x,yの像f(x)，f(y)が常に異なるということである。特にS，Tが同じ個数Nの元よりなる集合の場合，全射はまた単射でもあり，写像fは長さNのリストに対する順列と考えられる。図5に示すプログラムbijection（PF，S，T）はS，Tを与えられた集合とするとき，SからTへの全単射PFを生成する述語である。act（PF，P，F）は置換Pが写像Fに作用した結果のPFを求める述語である。また，permutation（New，Given）は集合Givenの要素からなる順列Newを生成する述語であり，delete（X，A，Y）はリストXから要素Aを除いたリストYを作る述語である。

```
bijection(*PF,*S,*T):-biject(*F,*S,*T),
                      permutation(*P,*S),act(*PF,*P,*F)
act([(*X,*U)],[*U],[(*X,*A)]):-!
act([*X,*U)|*PF],[*U|*F],[(*X,*A)|*F]):-!,act(*PF,*P,*F)
biject([(*B,*A)],[*A],[*B]):-!
biject([(*B,*A)|*F],[*A|*S],[*B|*T]):-!,biject(*F,*S,*T)
permutation([],[])
permutation([*A|*X1],*Y):-delete(*Y,*A,*Y1),
                          permutation(*X1,*Y1)
delete([*A|*X],*A,*X)
delete([*A|*X],*A,[*B|*Y]):-delete(*X,*A,*Y)
```

図5

(4) 写像f：S→T，g：T→Uに対し，fとgの合成写像h：S→Uはh(x)＝g(f(x))で定義される。図6に示すプログラムmseki（H，F，G）は合成写像のグラフ表現を求める述語である。また，そこに使われているval（V，F，X）は写像fのxにおける値y＝f(x)を求める述語である。

```
mseki([],[],*G)
mseki([],*F,[])
mseki([(*Z,*A)|*H],[(*X,*A)|*F],*G):-val(*Z,*G,*X),!,
                                      mseki(*H,*F,*G)
val(*Y,[],*X):-!,fail
val(*Y,[(*Y,*X)|*F],*X):-!
val(*Y,[*A|*F],*X):-!,val(*Y,*F,*X)
```

図6

(5) 図7に示すpseki（Z，X，Y）は同じ個数の要素からなる集合Xから集合Yへの全単射の積を求める述語である。そして，list_map（List，Map）は写像のリスト表現Listから写像のグラフ表現Mapを作る述語であり，map_list（Map，List）は逆に写像のグラフ表現Mapからリスト表現Listを作る述語である。

```
pseki(*Z,*X,*Y):-!,list_map(*X,*FX),list_map(*Y,*FY),
                   mseki(*FZ,*FX,*FY),map_list(*FZ,*Z)
list_map(*List,*Map):-length(*List,*N),
                      list_map_aux(*N,*List,*Map,'1),!
list_map_aux(*A,[],*B,*C):-!,fail
list_map_aux('1,[*A|*L],[(*A,*I)|*X],*I):-
          gtn(*N,'1),sub(*N,'1,*N1),add(*I,'1,*I1),
          list_map_aux(*N1,*L,*X,*I1)
length([],'0)
length([*A|*List],*L):-length(*List,*L1),add(*L1,'1,*L)
map_list(*Map,*List):-length(*List,*N),!
                      map_list_aux(*List,[],*Map,*N)
map_list_aux(*WM,*WM,[],'0):-!
map_list_aux(*Const,*WM,*WL,*WN):-sub(*WN,'1,*WN1),
                      delete(*WL1,*WL,(*Q,*WN)),!,
                map_list_aux(*Const,[*Q|*WM],*WL1,*WN1)
delete(*X,[*A|*X],*A)
delete([*B|*Y],[*B|*X],*A):-delete(*Y,*X,*A)
```

図7

(6) 図8に示すプログラムsignat（L，S）は順列の符号すなわち，順列Lが隅順列から奇順列かを判定する述語である。Lが隅順列ならS＝1となり，奇順列ならS＝－1となる。順列の符号の計算は順列の転倒数を計算することで行なえる。すなわち，転倒数が偶数なら隅順列，奇数なら奇順列である。Lの転倒数Mを求める述語tentou（L，M）は補助述語tentou（A，L，N）を用いている。ここに，Aは数，Lはリストとするとき Aより大きくないLの要素の個数がNである。

```
signat(*L,*S):-tentou(*L,*M),
    mod(*M,'2,*W1),mul(*W1,'-2,*W2),add(*W2,'1,*S)
tentou(*A,[*B],'1):-gtn(*A,*B),!
tentou(*A,[*B],'0):-!
tentou(*A,[*B|*L],*N):-tentou(*A,*L,*N1),
    if_then_else(gtn(*A,*B),add(*N1,'1,*N),add(*N1,'0,*N))
tentou([*A],'0):-!
tentou([*A|*L],*T):-tentou(*L,*T1),tentou(*A,*L,*N1),
                                   add(*T1,*N1,*T)
if_then_else(*P,*Q,*R):-*P,*Q,!
if_then_else(*P,*Q,*R):-not(*P),*R,!
not(*P):-*P,!,fail
nopt(*P)
```

図8

参考文献

市川　新　BASICで学ぶPROLOGシステム　啓学出版　1987年
飯高　茂　Prologで作る数学の世界　朝倉書店　1990年