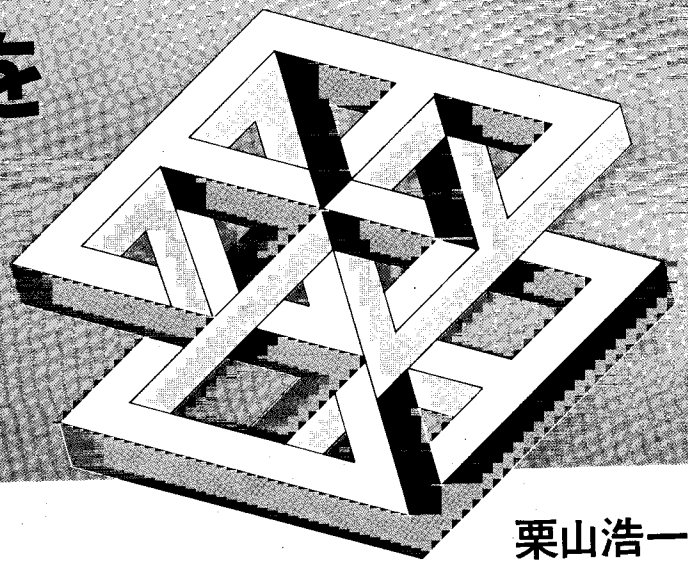
for FM Logo

誰にでもわかる人工知能

FMに推論機能を持たせよう



栗山浩一

1 第五世代コンピュータ

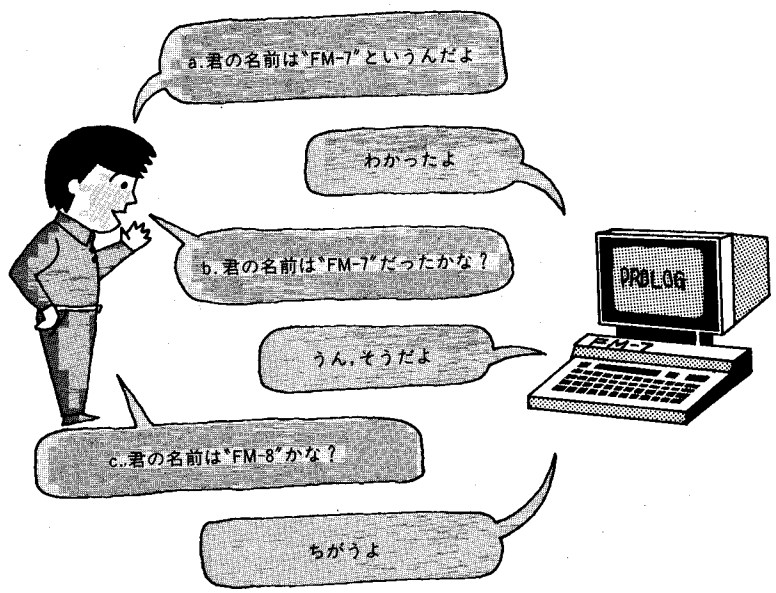
ちょっと前の話ですが，第五世代コンピュータが話題になりましたね。ファイゲンバウムの『第五世代コンピュータ』がベストセラーになりどんなものか知っている人も多いと思いますが，一言でいえば「第五世代コンピュータは推論マシン」なのです。推論とは「A＝B, B＝CのときA＝C」という三段論法のことです。三段論法自体はそう難しいものではありませんが，それを組み合わせることにより人間の思考のような高度なことができます。第五世代では高速・大量の推論を行うため専用のハードを必要としますが，程度を限定すれば現在のパソコンレベルでも充分な推論機能を持たすことができます。現にパソコン上で働く推論機能を持った言語もあります。

この稿ではFMにも推論機能を持たせることをねらい，人工知能言語PROLOGの簡単な紹介と，FM-Logoによる推論機能プログラム例を示します。

2 人工知能言語-PROLOG

PROLOGとは第五世代コンピュータに採用された言語で，推論機能を持っています。PROLOGは，本来フランス語の処理を目的として考案されたもので，そのため，とても会話的で，簡単な，そして人間的な言語です。だから，まるでコンピュータと会話をしているかのようにプログラミングできるのです。もっともPROLOGは"YES"と"NO"しかしゃべってくれないのですが，図1でaのことを「登録」，b, cを「問い合わせ」といいます。あまりプログラムという感じではないですが，これがPROLOGの基本です。つまり，PROLOGは「問い合わせ」されたのが「登録」されているか確認して，その結果を教えてくれるのです。

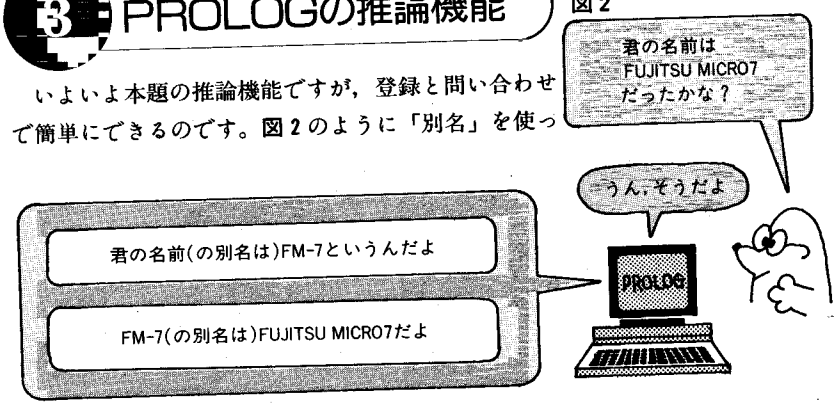
図1 登録と問い合わせ



3 PROLOGの推論機能

図2

いよいよ本題の推論機能ですが，登録と問い合わせで簡単にできるのです。図2のように「別名」を使っ



て登録することで，三段論法をPROLOGは実行します。ただ，これでは単語の別名しか表現できませんので，PROLOGでは文も表現できるようにしています。もちろん日本語のままでは理解してくれません。そこで述語論理という方法で文を表現しています。これは，述語を中心に文をまとめる方法で，

君の名前は　FM−7である。
(主語)　　(述語)

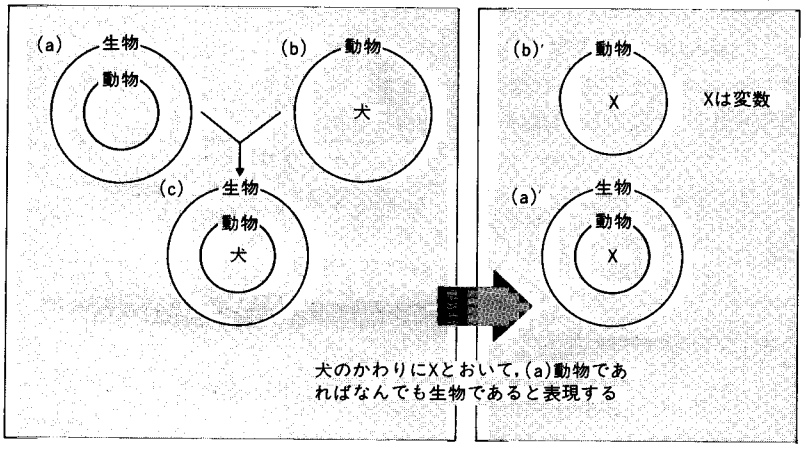
FM−7　　(君の名前)

と表現します。この方法を使って次の三段論法を表現してみましょう。

(a)　動物は生物である。
(b)　犬は動物である。
(c)　犬は生物である。

いろいろな方法が考えられますが，この例は図3のような状態になっていると考えられます。

図3



この考え方を使って(a)を書き替えると，

(a)′　Xが動物ならばXは生物である。

と変数を使って表現されます。なぜ，このようなことをするかというと，変数を使用せずに表現すると登録すべき数が増えてしまうことになってしまうからです。ところで(a)′は，

(a)″　動物(X)　ならば　生物(X)

と表現されますが，

A　ならば　B

これを，

B　＝　A

と表現するとすれば，

(a)‴　生物(X)　＝　動物(X)

となります。さきほどの例は，

(a)‴　生物(X)　＝　動物(X)
(b)′　動物(犬)
(c)′　生物(犬)

と表現されるわけです。

少し複雑になってきましたので，わからなくなってきた人もいるでしょうから，説明を加えますが，述語論理は，後ろから訳すとおぼえておけば理解しやすいでしょう。たとえば，

動物（犬）　　生物（X）　＝　動物（X）
2　1　　4　3　　2　1

の順に訳せば，

犬は動物である。Xが動物ならばXは生物である。となるわけです。以上でPROLOGの推論機能の説明は終わりです。世界に話題をふりまいた第五世代コンピュータも，その中心となる推論機能は，そんなに複雑で難しいものではない，ということがわかってもらえたと思います。

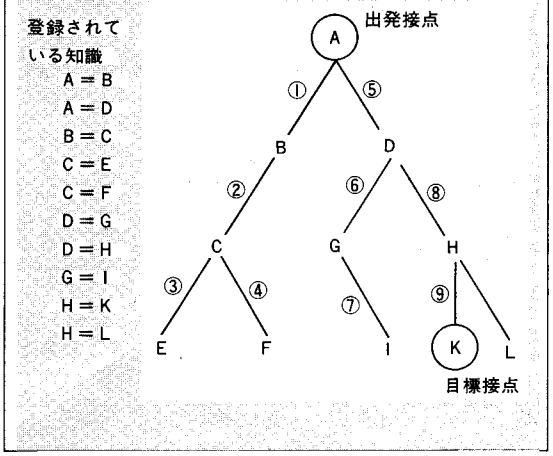
PROLOGの実際

次にPROLOGが，どのようにして推論しているのか説明します。少し複雑になりますが，ゴマかさないでお読みください。

まず最初に，単語どうしの別名による登録の場合を考えてみましょう。登録されている知識のすべては，「木」によって表現できます。木とは，問題を解いていく順序を表現したものです。つまり，この木を解くということは，その問題を解くということなのです。

図4は，「＝」で結ばれているものを線で結んだ結果

図4　知識の木による表現



できた木です。ここで，

「AはKですか？」

と問い合わせたらどうなるでしょう。もちろん答えは，

「はい」

です。それは，図4の木を見ればすぐにわかりますね。A−D−H−Kとつながっているのが，人間ならば，すぐにわかります。

しかし，コンピュータは，目で見て判断することはできません。そこで，木の枝を，一つひとつ，これは答えかな，と試しながら解を発見していくのです。PROLOG

では，左側の枝から試していきます。すると，**図4**の木の場合は，枝にふっている数字の順に試していくことになります。つまり，9回目に答えを発見します。これは枝の数が増えると，指数的にどんどん回数も多くなっていき，時間もメモリも，ものすごく食うことになります。だから，第五世代コンピュータは，この推論のスピードをあげることを第一目標としているのです。

話をもとに戻しますが，木の探索は，Oh!FM Vol. 6で説明しています。しかし，知らない人も多いでしょうから，その方法（アルゴリズム）を説明しておきます。

木の探索

① 出発接点を変数OPENに入れる。
② もし，OPENの中になにも入っていなければNOを出力して終了する。
③ NにOPENの最初の要素を代入する。
④ もし，Nが目標接点ならばYESを出力し，終了する。
⑤ OPENからNを取りのぞく。
⑥ 接点Nから枝分かれしている接点をすべてOPENの最初に，つけ加える。
⑦ GOTO 2

変数OPENは，BASICでは配列変数，Logoではリストを使用します。

出発接点は，**図4**でいえばA，目標接点はKとなります。

③ の「最初の要素」とは，BASICでは，OPEN (0)，Logoでは，FIRST (OPEN) です。

⑤ の処理はBASICでは，

OPEN (0)＝OPEN (1)
OPEN (1)＝OPEN (2)
⋮

Logoでは，BF (OPEN) となります。

⑥は，「接点を展開する」と呼ばれる動作です。たとえば，変数NがAならば，Aの枝であるBとDをOPENに入れることになるわけです。

図4を探索させたときのOPENの経過を**表1**に示します。少し複雑になってきたので，**表1**と上のアルゴリズムと**図4**を見比べて，どのように木を探索しているかを理解してください。

表1 探索中のOPEN

リストOPEN	配列 1 2 3	
[A]	A	出発接点
[B D]	B D	
[C D]	C D	
[E F D]	E F D	
[F D]	F D	
[D]	D	
[G H]	G H	
[I H]	I H	
[H]	H	
[K L]	K L	目標接点

次に，文による推論の場合を考えてみましょう。文の場合も，単語による推論と同じ，木の探索となります。**図5**を見れば，推論の構造が理解してもらえると思います。

図5 推論の構造

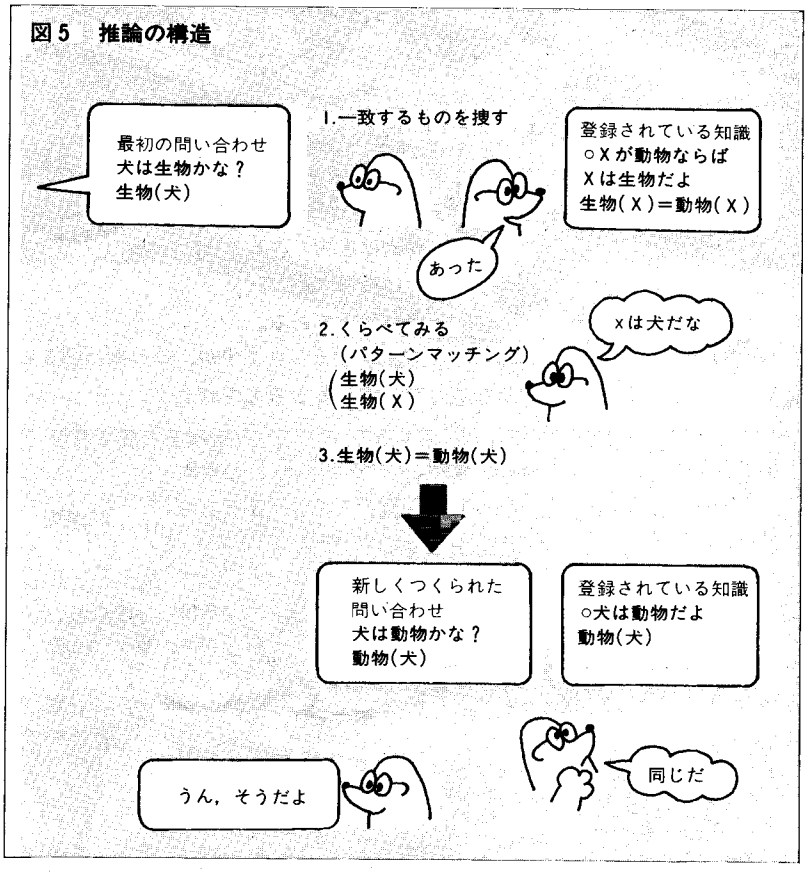


図5のように，PROLOGでは，問い合わせから内部の処理が始まり，推論し，答えを得ています。つまり，推論が逆向きなわけです。しかし，結果的には同じ答えを得ているところに注意してください。

5 FM Logoによる推論機能プログラム

Logoで実験的に推論を実行するプログラムを組んでみました。実は，PROLOGにはパターンマッチングというもう1つの大きな特徴があるのです。しかし，今回のプログラムでは，これをはずしました。そのため，PROLOGとはほどとおいものになってしまいましたが，現在Logoによる完全なPROLOGを製作中なので，いつか発表しようと思っています。

今回のプログラムで，実行できる推論は，先程述べた，犬の三段論法の形のものです。つまり，変数を1つだけふくんだ文の推論です。ただ，(a)の形が少し異なるので注意してください。

PROLOGによる表現	FM PROLOG
LIFE (X)＝ANIMAL (X)	[LIFE[ANIMAL]]
ANIMAL (DOG)	[ANIMAL[DOG]]
LIFE (DOG)	[LIFE[DOG]]

また，(a)は推論の途中に使われる事実，(b)は推論の最後に使われる事実と分けられます。つまり，問い合わせされたもの(c)を(a)を使用して(b)に導く関係にあるわけです(これは図5で説明しましたね)。そこで，(a)と(b)を分けて考え，

(a) —推論に使用される事実—は，リストKNOW

(b) —導かれる事実—は，リストLOGIC

に入れておくことにします。つまり，先程の三段論法は**図6**のようにして登録されるわけです。

図6　登録

```
MAKE␣KNOW　[LIFE[ANIMAL]]
MAKE␣LOGIC　[ANIMAL[DOG]]
```

次に問い合わせですが，

PROLOG　問い合わせリスト　FLAG

というオペレーションで実行します。FLAGは，**表1**のような経過を表示するときは[ON]，しないときは[OFF]にします。

実行例を**図7**に示しますので，やってみてください。

図7　実行例

図6のように登録したあと

```
PR␣PROLOG　[LIFE[DOG]] [OFF]
TRUE　(はい)
PR␣PROLOG　[LIFE[FISH]][OFF]
FALSE　(いいえ)
```

6 応用例——迷路を解くプログラム

推論機能を用いて，迷路を解くプログラムを作ってみましょう。ここに，**図8**のような迷路があったとします。このままでは，PROLOGは解いてくれませんから，これを**図9**のように木で表現してみましょう。**図8**と**図9**を見比べてください。両方が同じものであるのがわかりますか。

次に，この木をPROLOGに登録するために，WAYという命令を使用します。これは

WAY　接点1　接点2

で，「接点1から接点2まで道があるぞ」と教えるわけです。これを利用して迷路を表現すると**図10**のようになるのです。

WAY　A　Bは「Aを通ればBも通れる」「Bを通ればAも通れる」と2つの意味を持っています。つまり，

A (X)＝B (X)　　B (X)＝ A (X)

図8　迷路の例

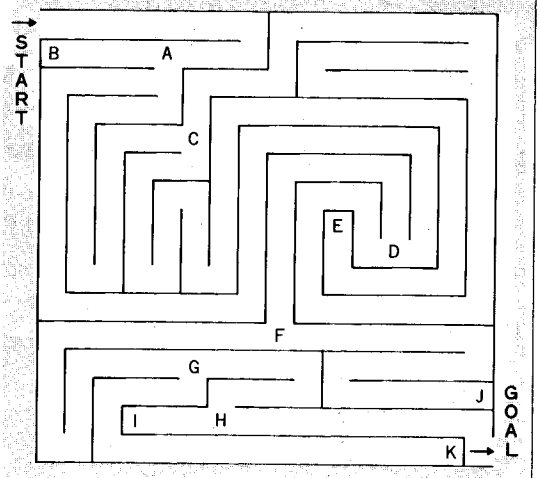


図9　木による迷路の表現

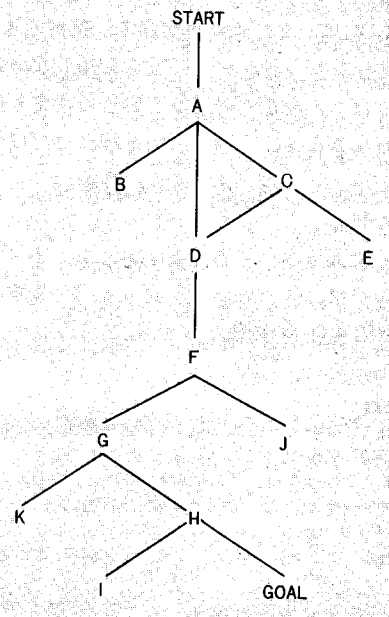


図10　WAYによる迷路の表現

```
WAY　START　A
WAY　A　B
WAY　A　C
WAY　A　D
WAY　D　C
WAY　C　E
WAY　D　F
WAY　F　G
WAY　F　J
WAY　G　H
WAY　G　K
WAY　H　I
WAY　H　GOAL
```

と表現できます。これをFM PROLOGに登録するときは，

KNOWリスト [[A[B]] [B[A]]]

となるわけです。実際には，MAKE·KNOWという命令を使用していますので，**リスト1**を参照してください。

そして，最後は「GOALはGOALである」に導かれます。これを登録してあれば，そこでPROLOGは「はい」と答えてくれるわけです。この経過を表示すれば，今までに通った道を表示してくれるわけです。そして**図11**のようになるわけです。

図11 実行例

```
MEILO [OFF]
[START A D F G H GOAL]
```

おわりに

PROLOGは，まだまだ奥が深く，難解なものです。特にパターンマッチングがIF文を使用しない条件判断をしていたり，今までの言語と異色の性質を持っています。みなさんも，今回の説明を読んだだけで納得しないで，いろいろ調べてみてください。

かなり難しいところもあったので，わからなくなってしまった人も多いと思いますが，1．2．だけでも読んでおけば，きっと，役に立つと思います。これからは，PROLOGの時代ですから。

●参考文献

白井良明，辻井潤一，「人工知能」（岩波講座情報科学22），1983年，岩波書店

後藤滋樹，「Prologとは」，「Computer Today」，サイエンス社

ファイゲンバウス，マコーダック，「第五世代コンピュータ」，1980年，TBSブリタニカ

「FM Logo リファレンスマニュアル」，1984年，富士通

リスト1 迷路の自動探索

初期設定およびメインプログラム

```
TO MEILO :A
MAZE.WAY
MAKE "IN [START [GOAL]]
MAKE "LOGIC [GOAL [GOAL]]
MAKE "MEILO [ ]
MAKE "MCLOSE [ ]
MAKE "MOPEN [ ]
MAKE "LOGIC LPUT LIST FIRST LAST :IN LA
ST :IN [ ]
PR PROLOG :IN :A
FINISH :MEILO
END
```

PROLOGのメインプロシジャ

```
TO PROLOG :IN :FLAG
MAKE "CLOSE [ ]
MAKE "OPEN LPUT :IN [ ]
IF LOOP :OPEN :FLAG [OUTPUT "TRUE] [OUT
PUT "FALSE]
END
```

木の探索に相当するプロシジャ
①OPENが空ならFILESを出力 ②NにOPENの第1要素を代入する ③Nが目標接点ならばTRUEを出力 ④違うならばOPEN＝BF(OPEN)してNを展開したものをOPENに加える

```
TO LOOP :OPEN :FLAG
IF EMPTYP :IN [OUTPUT "FALSE]
IF EQUALP :FLAG [ON] [SHOW :OPEN]
MAKE "N FIRST :OPEN
MAKE "MEILO LPUT FIRST :N :MEILO
MAKE "CLOGIC :LOGIC
IF GOAL :N [OUTPUT "TRUE]
MAKE "CKNOW :KNOW
MAKE "OPEN BF :OPEN
TENKAI :N
MAKE "CLOSE LPUT :N :CLOSE
OUTPUT LOOP :OPEN :FLAG
END
```

Nを展開してOPENの前に入れる

```
TO TENKAI :N
IF EQUALP FIRST :N FIRST FIRST :CKNOW [I
IF MEMBERP :N :CLOSE [ ] [MAKE "OPEN FP
UT LIST FIRST LAST FIRST :CKNOW LAST :N
 :OPEN]]
MAKE "CKNOW BF :CKNOW
IF EMPTYP :CKNOW [STOP]
TENKAI :N
END
```

もしNが目標接点ならばTRUEを出力。違うならFALSEを出力

```
TO GOAL :N
IF EQUALP :N FIRST :CLOGIC [OUTPUT "TRU
E]
MAKE "CLOGIC BF :CLOGIC
IF EMPTYP :CLOGIC [OUTPUT "FALSE]
OUTPUT GOAL :N
END
```

CLOSEを整理する。PROLOGの通った枝がCLOSEに入っているが〔START ADFJFGKGKGHIH GOAL〕のようにバックトラック(もとにもどること)したものが残っているので〔START ADFGH GOAL〕のように整理している

```
TO FINISH :CMEILO
IF EMPTYP :CMEILO [PR :MOPEN STOP]
MAKE "W FIRST :CMEILO
IF MEMBERP :W :MCLOSE [ ] [MAKE "CMEILO
 BF :CMEILO MAKE "MCLOSE LPUT :W :MCLOS
E MAKE "MOPEN LPUT :W :MOPEN FINISH :CM
EILO STOP]
MAKE "CMEILO BF :CMEILO
MLOOP :W
FINISH :CMEILO
END
```

FINISHのサブ

```
TO MLOOP :IN
IF EQUALP :IN LAST :MOPEN [STOP]
MAKE "MOPEN BL :MOPEN
MLOOP :IN
END
```

リストKNOWに入力されたリストを加える

```
TO WAY :A :B
MAKE "X LPUT :A :B
MAKE "Y LPUT :B :A
MAKE.KNOW :X
MAKE.KNOW :Y
END
```

迷路を登録する

```
TO MAZE.WAY
MAKE "KNOW [ ]
WAY [START] [A]
WAY [A] [C]
WAY [A] [B]
WAY [A] [D]
WAY [D] [C]
WAY [C] [E]
WAY [D] [F]
WAY [F] [G]
WAY [F] [J]
WAY [G] [H]
WAY [G] [K]
WAY [H] [GOAL]
WAY [H] [I]
END
```

リストLOGICに入力されたリストを加える

```
TO MAKE.LOGIC :T
MAKE "LOGIC LPUT :T :LOGIC
END
```

リストKNOWに入力されたリストを加える

```
TO MAKE.KNOW :T
MAKE "KNOW LPUT :T :KNOW
END
```