Ｐｒｏｌｏｇ

Prolog（Programming in Logic）は論理型言語の代表的なもので、1972 年頃、仏・マルセイユ大学の A.Colmerauer によって提唱された。述語処理による数学定理の自動証明を記述することが目的で開発された言語であるが、その後、プログラミング言語として利用されるようになった。

最初の Prolog 処理系は英・エジンバラ大学で開発された。1980 年代に日本の第五世代コンピュータ開発プロジェクトにおいて主要プログラミング言語として採用されたことで一時期もてはやされたが、現在ではあまり使われていない。

特徴：

- 数理論理学に基づく数学的基礎理論がバックにある。
- 事実や処理（＝プログラム）を、対象の間の関係（述語）として記述する。
- プログラミング言語としては、LISP 流のリスト処理も含む。
- 応用分野：　データベース（推論システム、知識ベース）、人工知能研究など。
- Prolog のプログラムにおける計算は、パターンマッチングによって実行される（定理の自動証明における「証明の過程」に対応する）。
- 処理系は基本的にインタープリタ。実行効率は低い。

述語

述語（predicate）とは、真理値（＝論理値：「真」true, **T**、または「偽」false, **F**)を値に取る関数のことである。

（例）

(1) parent(x,y) = { **T**　x が y の親であるとき ; **F**　そうでないとき

述語名：parent　　変数：x, y（2 変数述語）

(2) =(x, y)：　"x=y?" を表す
=(3, 2+1) は **T**,　　=(3, 5) は **F**
2, 3 は定数

(3) factorial(f, n)：　f=n!
factorial(6, 3)=**T**,　　factorial(7,2)=**F**

(4) triangle(x, y, z)： x, y, z は三角形の三辺をなす

すなわち、triangle(x,y,z) = (x+y>z)∧(y+z>x)∧(z+x>y) が成り立つなら **T**,

そうでないなら **F**

(5) prime_minister(x)：x は現在、日本国の首相である

prime_minister(小泉純一郎)=**T**, prime_minister(守屋悦朗)=**F**

Prolog の構文

・定数、変数、述語名、…とは何かを明確にする必要あり

・プログラミング言語としての構文をきちんと定める必要がある

以下のように順次、プログラムの構成要素を定義する。

項 (term)とは何かを次のように再帰的に定義する：

項	定数（アトム）	文字列定数	ABC, ｱｲｳｴｵ, xx12y_ABC, 'ABC', ==, '>='
		整数	1234, -456
		実数	12.3, -3.45, -5.6E-7
	変数（_で始まる文字列）		_A, _xyz, _x123
	複合項		g(12), f(_x, g(12))

複合項とは … 関数名(引数 1 , 引数 2,…)

変数以外の文字列定数　　項

（例） ABC(_x)

好き(ソクラテス, _x)

親子(_x, _y)

>(25, 10)　　25>10 と書いてもよい（前置記法と中置記法）

+(5, 7)　　5+7 と書いてもよい

関東地方(千葉)

>(+(5, 7), *(+(2, 3), 2))　　(5+7)>((2+3)*2)

文 (statement) の定義

- 文
  - 事実文　　p.
  - 規則文　　p:-q, r, ….
  - 実行文
    - 質問文　　?-p, q, ….
    - コマンド文　　:-p, q, ….

p, q, r, … は項

文末にピリオドを付ける

Prolog のプログラムとは、文の列のことである。

① 事実文

p.　　p は述語で、「p は真である」を表す

ある事実が真であることを表す

(例)

(1) 親(太郎, 花子).　　「太郎は花子の親である」

(2) 男(太郎).　　「太郎は男である」

② 規則文

p :- q_1, q_2,…, q_n.　　各 q_i は述語で、

「q_1 かつ q_2 かつ…かつ q_n が真ならば p も真である」を表す

ある規則を与える(定義する)のに使う

(例)

(1) 父親(_X, _Y) :- 親(_X, _Y), 男(_X).

「x が y の父親であるとは、x が y の親であり、かつ x が男であることである」

(2) 兄弟(_X, _Y) :- 親(_Z, _X), 親(_Z, _Y).

兄弟(_X, _Y) :- 兄弟(_Y, _X).

「x と y が兄弟であるとは、z が x の親であり、かつ z が y の親である

(そのような z が存在する)ことである」

「y が x と兄弟なら、x と y は兄弟である」

(問) 次のことを規則文として表せ。

(1) x と y はいとこ同士である

(2) x は y の子孫である

(3) x は素数である

③　実行文

③－(a) 質問文

?- $p_1, p_2, \cdots, p_n$.　　各 p_i は述語で、

「p_1 かつ p_2 かつ…かつ p_n が成り立つか？」を表す

③－(b) コマンド文

:- $p_1, p_2, \cdots, p_n$.　　各 p_i は述語で、

「p_1 かつ p_2 かつ…かつ p_n を実行せよ」を表す

（例）

?- 父親(_X, 花子).　　花子の父親は誰か？

_X = 太郎;　　システムからの応答

別解がある場合、；を入力すると別解が表示される

NO　　NO は別解なしの意

?- 父親(太郎, 花子).

YES

ユニフィケーション

ユニフィケーション(unification)とは、2 つの項を比較して、同じ構造・値をもつかを調べ、もし一方または両方に変数が含まれていればそれらに適当な値を代入して 2 つの項の値を一致させるようにすること。そのためには、2 つの項の**パターンマッチング**（pattern matching）を行なう。

Prolog では、質問文の各項とユニファイ可能な事実や規則があるかどうかを調べ、ユニフィケーションを繰り返して答えを探す：

・質問文の各項を左から右の順に調べる。
・１つの項に対しては、事実文、規則文を登録順に調べる。
・この順でマッチングを進め、失敗するとバックトラックする（backtracking：失敗の原因点に戻って、別の可能性に当たること）。

・**マッチング規則**：
　＊定数は同じ値の定数とだけマッチする。
　＊変数は定数、変数、複合項のいずれともマッチする。
　＊複合項は、そのすべての引数がマッチするような項とマッチする。

（例）　マッチする例　　マッチしない例

F(_X) と F(ABC)　　F(A, B, C) と F(D, _X, _Y)

P(_X, ABC) と P(A, _Y)　　P(1, 2, 3) と P(1, 2)

F(G(H(A),K(B,C)),D) と_X　　F(G(A,B),1) と F(H(A),_X)

（例）マッチングの実行例

① 母(千鶴子, 良子).

② 母(良子, 洋子).

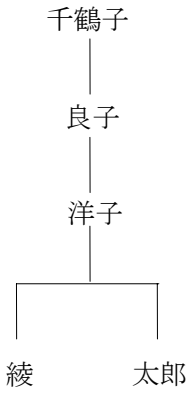
③ 母(洋子, 綾).

④ 兄弟(綾, 太郎).

⑤ 兄弟(_X, _Y) :- 兄弟(_Y, _X).

⑥ 叔母(_X, _Y) :- 女(_X), 親(_Z, _Y), 兄弟(_Z, _X).

⑦ 祖母(_X, _Y) :- 母(_X, _Z), 母(_Z, _Y).



?- 祖母(_A, 綾).　　綾の祖母は誰か？

⑦とマッチさせる

↓

⑦' 祖母(_A, 綾) :- 母(_A, _Z), 母(_Z, 綾).

③とマッチする

↓

⑦" 祖母(_A, 綾) :- 母(_A, 洋子), 母(洋子, 綾).

②とマッチする

↓

⑦''' 祖母(良子, 綾) :- 母(良子, 洋子), 母(洋子, 綾).

_A = 良子　　システムの応答

?- 母親(洋子, _B).　　洋子を母とするのは誰か？

_B = 綾;　　綾（システム応答）以外には？

_B = 太郎;　　さらに別解を要求

NO　　これ以上の解なし

（問）叔父、姪を定義せよ。

リスト

Prolog では LISP で言うところのリスト $(a_1\ a_2\ \cdots\ a_n)$ を

$[a_1, a_2, \cdots, a_n]$

のように記す。

（例）

[春，夏，秋，冬]

[父，母, [祖父，祖母]]

[ ]　…　空リスト

リスト同士のパターンマッチングの仕方

$[\_x_1, \_x_2, \cdots, \_x_n \mid \_y]$ と $[A_1, A_2, \cdots, A_n, B_1, \cdots, B_m]$ とは

$\_x_1 = A_1,\quad \_x_2 = A_2, \cdots, \_x_n = A_n, \_y = [B_1, \cdots, B_m]$

のようにマッチする。すなわち、｜はリストをその位置で前後２つに分割することを表す演算である。

（例）

```
動物([イヌ，ネコ，ウサギ]).
?- 動物(_X).
_X = [イヌ，ネコ，ウサギ]
?- 動物([_X, _Y, _Z]).
_X = イヌ, _Y = ネコ, _Z = ウサギ
?- 動物([_X | _Y]).
_X = イヌ, _Y = [ネコ，ウサギ]
```

（例）

```
①　順列([], []).
②　順列([_X | _Y], _Z) :- 順列(_Y, _W), 挿入(_X, _W, _Z).
```

順列の定義

順列(a, b)　…　"b は a の順列である"

```
③　挿入(_X, _Y, [_X | _Y]).
④　挿入(_X, [_Y | _Z], [_Y | _W]) :- 挿入(_X, _Z, _W).
```

挿入の定義

挿入(a, b, c)　…　"b に a を挿入したものが c である"

```
⑤　順列([1, 2, 3, 4, 5], _X).
```

[1,2,3,4,5]の順列

このとき、

② [0, 0, 0, 0, 0, 0, 0]

X Y

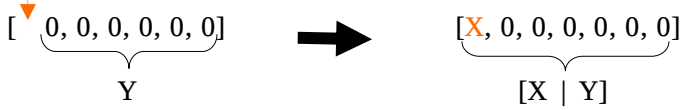
[y, y, y, y, y, y] Y の順列

W

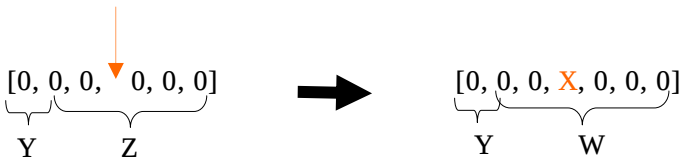
[y y, X, y, y, y, y] W の中に X を挿入したもの

Z [X | Y] の順列

③ X を Y の前に挿入する



④ X を Z の中に挿入する



(問)

(1) 何を定義しているか？

```
append([ ], _X, _X).
append([_W | _X], _Y, [_W | _Z]) :- append(_X, _Y, _Z).
```

(2) リスト_X を反転させる reverse(_X) を定義せよ。

(例１) n！を求める

```
f(0, 1).                    0! = 1
f(_N, _FN) :- _N1 is _N-1, f(_N1, _FN1), _FN is _FN1*_N.
```

N1 = N-1

N1! = FN1

FN = FN1*N = N1!*N = (N-1)!*N = N!

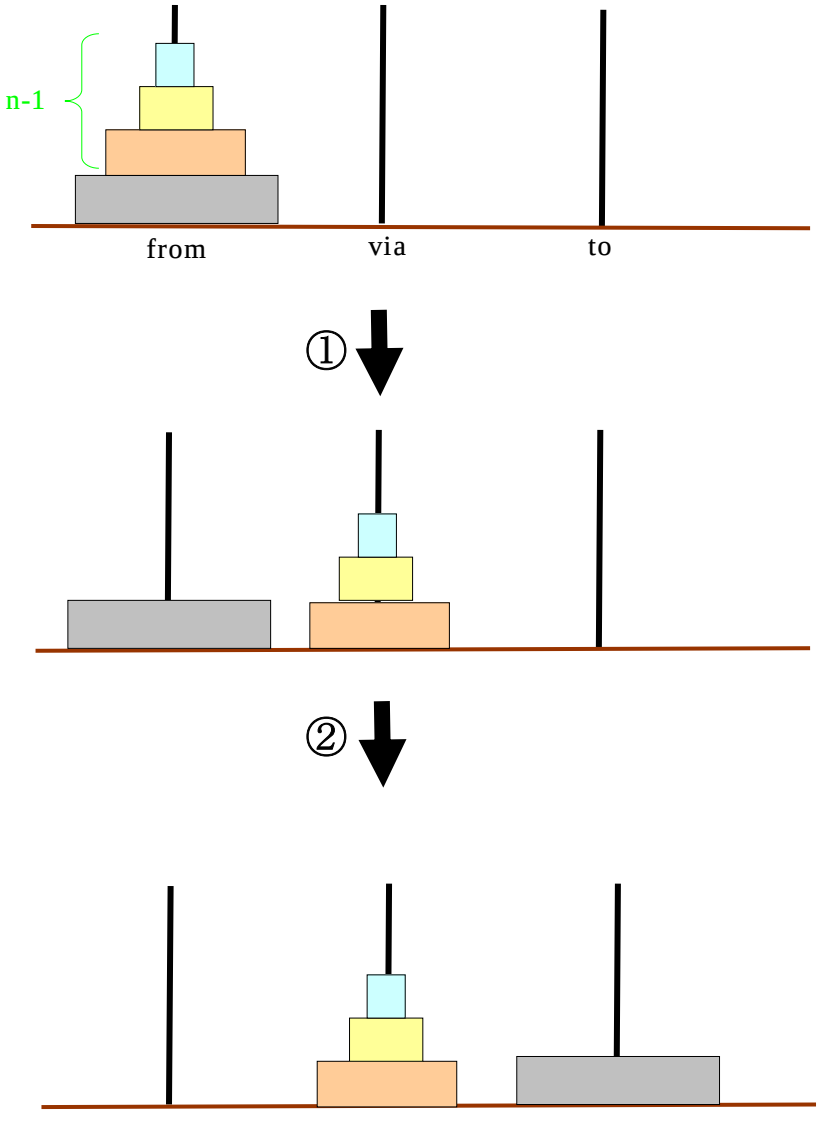
↓

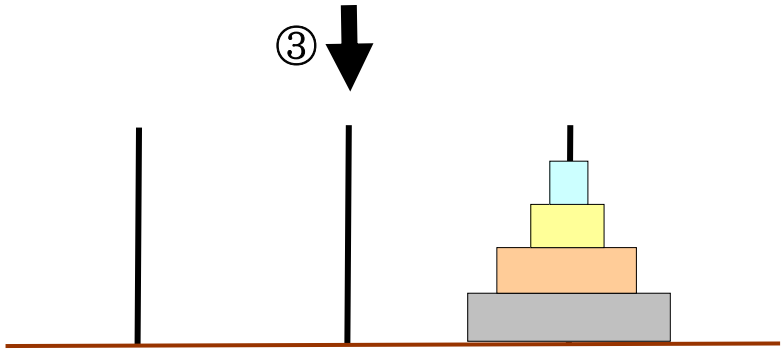
N! = FN

```
?- f(3, _X).                3!  は？
_X = 6
```

（例２）ハノイの塔

```
Hanoi(1, _from, _via, _to) :- move(_from, _to).                    ①
Hanoi(_n, _from, _via, _to) :-  _n1 is _n-1,                       ②
                                Hanoi(_n1, _from, _to, _via),      ③
                                move(_from, _to),                  ④
                                Hanoi(_n1, _via, _from, _to).      ⑤
move(_from, _to) :- write([MOVE, _from, TO, _to]).                 ⑥
```





（問）

(1) FATHER(_X, _Y) :- PARENT(_X, _Y), MAN(_X).
MOTHER(_X, _Y) :- PARENT(_X, _Y), WOMAN(_X).
PARENT(ごんべえ，たろう).
PARENT(ごんべえ，はなこ).
PARENT(おはる，たろう).
PARENT(おはる，はなこ).
PARENT(うたえもん，ごんべえ).
MAN(うたえもん).
MAN(ごんべえ).
MAN(たろう).
WOMAN(はなこ).
WOMAN(おはる).
と定義されているとき、太郎の母は誰か？

(2) 祖父母を定義し、花子の祖父を求めよ。

Prolog を使ってみる

Prolog の処理系としては研究用に作られたものがたくさんあったが、最近ではどの Prolog 処理系も開発が止まっている。現在でも使われている処理系の一つに SB-Prolog がある。SB-Prolog はニューヨーク州立大学の David Warren と Suzanne Dietrich によって作られたものであるが、1987 年以来まったくバージョンアップが行われていない。

SB-Prolog については、次のウェブサイトが詳しい：

http://www.axe-inc.co.jp/~take/ailabo/sbpro/sbprolog.html

（１）次のような内容のファイルを作成しておく：

ファイル **example.prolog** :

```
animal([dog, cat, rabbit]).
append([ ],X,X).
append([A|X],Y,[A|Z]):- append(X,Y,Z).
fact(0,1).
fact(N,X) :- N1 is N-1 , fact(N1,X1), X is X1*N.
Hanoi(1, FROM, VAI, TO) :- move(FROM, TO).
Hanoi(N, FROM, VIA, TO) :- N1 is N-1,
                           Hanoi(N1, FROM, TO, VIA),
                           move(FROM, TO),
                           Hanoi(N1, VIA, FROM, TO).
move(FROM, TO) :- write([move, FROM, to, TO]).
father(X, Y) :- parent(X, Y), man(X).
mather(X, Y) :- parent(X, Y), woman(X).
parent(gonbei, taro).
parent(gonbei, hanako).
parent(oharu, taro).
parent(oharu, hanako).
parent(utaemon, gonbei).
man(utaemon).
man(gonbei).
man(taro).
woman(hanako).
woman(oharu).
```

このように、SB-Prolog では、変数は大文字で記述し、定数は小文字で記述する。

（２）SB-Prolog インタープリタの実行

```
% sbprolog                          Prolog インタープリタの起動
SB-Prolog Version 3.1
| ?- consult('append.prolog').      | ?-は Prolog のプロンプト
yes
| ?- append([a,b,c], [x,y,z], X).
X = [a,b,c,x,y,z]                  一旦停止するので、Return キーを押下
yes
```

```
| ?- append(X,Y,[a,b,c,d]).
X = []
Y = [a,b,c,d];                    一旦停止するので、；を入力
X = [a]
Y = [b,c,d];
…                                 途中略（その他の X,Y の候補が出力される）
no
| ?- fact(6,X).
X = 720
yes
| ?- Hanoi(3,a,b,c).
| ?- parent(gonbei,X).
| ?- mother(X,taro).
```

（演習）定義し、実行してみよ。

(1) member(X,Y)　…　Ｙは集合の元をリストアップしたもので、X はＹの元である。

(2) intersection(X,Y)　…　集合ＸとＹの共通部分

(3) graph(G)　…　Ｇはグラフである（辺のリストを与えて定義する）

(4) reachable(X,Y,G)　…　グラフＧにおいて、頂点ＸからＹへの道がある