Prolog 入門

田村直之 (tamura@kobe-u.ac.jp)

1 Prologの概要

- AI 用言語 (AI プログラムの記述・開発に適している言語) の一種
- 通産省の第五世代コンピュータプロジェクトの核言語
- 1974 年に仏国の Colmerauer らにより開発された
- PROgramming in LOGic (論理プログラミング) の省略
- 英国エジンバラ大学の DEC-10 Prolog が実質的標準

2 Prologの特徴

- **記号処理言語，リスト処理言語**
  データとして，記号 (シンボル) を取り扱うことができる．また，リストと呼ばれる可変長のデータの列を取り扱うことができる．
- **論理型言語**
  Prolog では新たな述語を論理式で定義することによってプログラムを作り上げていく．すなわち，Prolog のプログラムは論理式の集まりである．Prolog や LISP は，FORTRAN や BASIC などの手続き型言語とは異なり，非手続き型言語と呼ばれる．
- **対話的使用，会話形使用**
  Prolog システムの使用形態は対話的である．すなわち，ユーザは Prolog システムを立ち上げたあと，システムと会話するような形で命令を与え，述語を定義したり実行したりできる．

3 記号

Prolog では，記号は英小文字 (a, b, . . . , z) の後に，英字 (下線も英字に含める) または数字を 0 文字以上続けた名前で表される．

```
a  coffee  n_Tamura  may2
```

Prolog では，Lisp と異なり英字の大文字と小文字は区別する．いくつかの Prolog システムでは漢字も使用できる．

4　変数

Prolog では，変数は英大文字 (A, B, . . . , Z, 下線も英大文字に含める) の後に，英字または数字を 0 文字以上続けた名前で表される．

```
A  Coffee  MAY2  _1
```

5　リスト

Prolog では，リストは要素をコンマで区切って並べたものを角カッコでくくって表す．たとえば

```
[jan, 31, 1957, thu]
[pi, 3.14]
```

であり，一般的には

$$[\text{要素}_1, \text{要素}_2, \ldots, \text{要素}_n]$$

と書く．空リスト [] は記号 nil で表すこともできる．

6　述語

Prolog では**述語** (predicate) と呼ばれるものを定義することによってプログラムを作り上げていく．

述語は，データの間の**関係**やデータの持つ性質を表す．たとえば太郎 (記号 taro で表す) と次郎 (記号 jiro で表す) の間で，太郎が次郎の父親であるという関係が存在するとする．この関係は「父親である」という関係を表す述語 father を使って以下のように表現できる．

```
father(taro, jiro)
```

すなわち，father(F,C) により「F が C の父親である」という関係を表現している．taro と jiro は**引数**と呼ばれる．同様に,「男である」という性質を表す述語 male を用いると，太郎が男であるということは次のように表現できる．

```
male(taro)
```

一般的には,

$$\text{述語}\,(\text{引数}_1, \text{引数}_2, \ldots, \text{引数}_n)$$

と書くことで引数の間の関係 (あるいは性質) を表す.

練習問題

問 1　太郎を表すのに taro ではなく，大文字を使って Taro としたほうがわかりやすいと思うが，なぜそうしないだろうか.

7　事実

Prolog では述語 (あるいは関係といってもいい) は，**事実** (fact) または**規則** (rule) によって定義される．ここでは，事実による定義について説明する (規則については後述).

上で述べた，太郎が次郎の父親であるという関係を正しい事として定義するには，以下の行を Prolog プログラム中に記述する (最後にピリオドがあることに注意!).

```
father(taro, jiro).
```

これは**事実** (fact) と呼ばれる．

同様に，太郎が男であるという事実は，プログラム中で

```
male(taro).
```

と記述する．

8 Prologを使ってみる (その1)

Prologシステムを立ち上げると，プロンプト (入力促進記号) として “| ?-” が表示される (カーソルは箱で示す)．

```
| ?-
```

以下では，次の8つの事実からなるプログラムを例にとる．

```
father(naoyuki, hyogo).
father(saburo, naoyuki).
father(saburo, shinji).
father(yoshihisa, hisako).
mother(hisako, hyogo).
mother(yoko, naoyuki).
mother(yoko, shinji).
mother(nobuko, hisako).
```

ここで，述語 father は「第1引数が第2引数の父親である」という関係を表し，述語 mother は「第1引数が第2引数の母親である」という関係を表す．また，このプログラムは family という名前のファイルに書かれているものとする．

Prologプログラムをファイルから読み込むには，“[ファイル名].” と入れる．family というファイルからプログラムを読み込むには次のようにする．

```
| ?-[family].
yes
| ?-
```

プログラムが正しく読み込まれた時は yes と表示される．

読み込まれたプログラムを表示させるには，“listing.” と入力する．

```
| ?-listing.
father(naoyuki,hyogo).
father(saburo,naoyuki).
father(saburo,shinji).
father(yoshihisa,hisako).
mother(hisako,hyogo).
mother(yoko,naoyuki).
mother(yoko,shinji).
mother(nobuko,hisako).
yes
```

9 単純な質問

Prolog では，**質問** (問合せ，query) により，関係が成立するかどうかを調べることができる．たとえば，naoyuki が hyogo の父親であるかどうかを調べるには，“?-father(naoyuki, hyogo).” という質問を入力する (“?-” の部分は，すでに表示されているので実際に入力する必要はない)．関係が成立すれば yes と表示され，成立しなければ no と表示される (より正確には，成立することがいえなければ)．

```
| ?-father(naoyuki, hyogo).
yes
| ?-father(saburo, hyogo).
no
```

yes が表示されることを**成功**，no が表示されることを**失敗**と呼ぶことがある．

知りたい箇所を変数にして質問することもできる．たとえば，誰が hyogo の父親であるかを調べるには，“?-father(F, hyogo).” という質問を行う．Prolog システムは，質問の関係が成立するような変数の値を答える．変数にどのような値を代入しても成立しない場合は no と答える．

```
| ?-father(F, hyogo).
F       = naoyuki
yes
| ?-father(F, saburo).
no
```

各質問ごとで変数は独立である．したがって，上の例での 1 つめの質問 “?-father(F, hyogo).” の変数 F と，2 つめの質問 “?-father(F, saburo).” の変数 F とは異なる変数と考えて良い．

逆の質問，すなわち誰が naoyuki の子供であるかを知るには第 2 引数を変数にすれば良い．

```
| ?-father(naoyuki, C).
C       = hyogo
yes
```

変数を含んだ質問では，複数の答えが存在する場合がある．たとえば，saburo の子供は naoyuki と shinji の 2 人がいるから “?-father(saburo, C).” の答えは 2 種類存在するはずである．Prolog ではこのような場合，プログラム中での記述の順序にしたがって答えが表示される．したがって，この例では C=naoyuki がまず示され，ユーザからの入力待ちになる．

```
| ?-father(saburo, C).
C       = naoyuki
```

ここで，セミコロン (“;”) を入れてリターンキーを押すと，Prolog システムは次の答えを求め，表示する．

```
| ?-father(saburo, C).
C       = naoyuki;
C       = shinji
```

さらに，セミコロン，リターンと入力すると，Prolog システムは次の答えを求めようとするが，存在しないので no と表示する．

```
| ?-father(saburo, C).
C       = naoyuki;
C       = shinji;
no
```

セミコロン，リターンの代わりに，単にリターンを入力した場合は，システムは次の答えを求めないで，yes と表示する．実はこれまでの例，“?-father(F, hyogo).” などでも答えの表示に対してリターンを入力していた．

すべてを変数にした質問も可能である．たとえば，“?-father(F, C).” という質問をすると，変数が 1 つの場合と同様に，質問の関係が成立するような変数の値が表示される．

```
| ?-father(F, C).
F       = naoyuki,
C       = hyogo
```

この場合も，答えの表示される順序はプログラムでの記述の順序に従う．また，セミコロン，リターンにより次の答えが表示されるのも同じである．

```
| ?-father(F, C).
F       = naoyuki,
C       = hyogo;
F       = saburo,
C       = naoyuki
```

練習問題

問 2　hyogo に子供がいて yoko という名前だとしよう (naoyuki の母親の yoko とは別人)．どの様な事実を付け加えればよいか．

問 3　このプログラムに常識とは矛盾する事実，たとえば “father(hyogo，hyogo)．” を付け加えることはできるだろうか (常識では，自分が自分の父親にはなれない)．

10　複雑な質問

Prolog では，単純な質問を組み合わせた複雑な質問をすることもできる．

たとえば，誰が hyogo の父方の祖父かという質問を考えてみる．hyogo の父を変数 F で，hyogo の父方の祖父を変数 G で表したとすると，hyogo と F との関係は “father(F，hyogo)” で，F と G との関係は “father(G，F)” で表される．そして，求める答えはその 2 つの関係が両方とも成立する場合である．これを Prolog では，2 つの関係をコンマ (“,”) でつないだ次のような質問で表現する．

```
| ?-father(F, hyogo), father(G, F).
G       = saburo,
F       = naoyuki
```

コンマは論理における「かつ」(連言，and) を意味している．もちろん，2 つの関係の順序を入れ換えても同じ意味である．

```
| ?-father(G, F), father(F, hyogo).
G       = saburo,
F       = naoyuki
```

Prolog システムは，コンマを含んだ質問を左の述語から順に調べていく．たとえば，“?-father(F，hyogo)，father(G，F)．” の質問では，まず father(F，hyogo) の答えを求める．答えは F=naoyuki である．次に，その答えのもとで father(G，F)，すなわち father(G，naoyuki) の答えを求める．答えは G=saburo となり，質問全体の答えは G=saburo，F=naoyuki となる．

質問 “?-father(G，F)，father(F，hyogo)．” の場合も左から順に処理される．まず father(G，F) の最初の答え G=naoyuki，F=hyogo が求められる．次に father(F，hyogo)，すなわち father(hyogo，hyogo) の関係を調べるが，成立しないで失敗する．このような場合，Prolog は**後戻り** (**バックトラック**，**バックトラッキング**，backtracking) して，別の答えを探す．すなわち，後戻りして father(G，F) の次の答えを求めにいき，G=saburo，F=naoyuki を得る．続いて father(F，hyogo)，すなわち father(naoyuki，hyogo) を調べる．こんどは成功し，質問全体の答えは G=saburo，F=naoyuki となる．

練習問題

問 4　hyogo の母方の祖父を求めるにはどの様な質問をすれば良いか．

問 5　“?-father(F，naoyuki)，father(F，C)．” は何を質問していると考えられるだろうか．

問 6　質問 “?-father(F，naoyuki)，father(F，C)．” はどのように処理されるか．

11 規則

前にも触れたように，Prolog では事実だけでなく**規則** (rule) と呼ばれるものによって新たな述語 (すなわち関係) を定義することができる．

たとえば,「P が C の親である」という関係を述語 parent を用いて parent(P, C) で表し，この述語を既存の father と mother という述語で定義することを考える．これらの述語間の関係を考えると，まず father(P, C) ならば parent(P, C) である．これは Prolog では以下の規則として表現される．

```
parent(P, C) :- father(P, C).
```

ここで，":-" は「ならば」を意味する．ただし ":-" の右側が条件で左側が結論である．すなわち P :- Q. により「Q ならば P」を意味する．さらに，mother(P, C) ならば parent(P, C) も正しい．

```
parent(P, C) :- mother(P, C).
```

父親と母親だけが親であるから，述語 parent は以上の 2 つの規則で定義できる．

「ならば」の条件がもっと複雑であってもよい．たとえば「祖父である」という関係を述語 grandfather で表し，この述語を既存の father と mother という述語で定義することを考える．父親の父親は祖父であり，母親の父親も祖父であるから，述語 grandfather は次のような 2 つの規則で定義できる．

```
grandfather(GF, C) :- father(F, C), father(GF, F).
grandfather(GF, C) :- mother(M, C), father(GF, M).
```

あるいは，parent と father を使って次の規則 1 つで定義することもできる．

```
grandfather(GF, C) :- parent(P, C), father(GF, P).
```

練習問題

問 7　「GM が C の祖母である」を表す述語 grandmother(GM, C) を定義せよ．

問 8　「C が P の子供である」を表す述語 child(C, P) を定義せよ．

問 9　「C が G の孫である」を表す述語 grandchild(C, G) を定義せよ．

問 10　「X が Y の兄弟姉妹である」を表す述語 sibling(X, Y) を定義せよ．

問 11　「X が Y の兄弟である」を表す述語 brother(X, Y) を定義せよ．

12 Prolog を使ってみる (その 2)

ここではプログラムをキーボードから入力する方法を説明する．キーボードから入力するには，まず '[user].' と入れる．そうするとプロンプトとして '| ' が表示される．

```
| ?-[user].
| 
```

ここで，事実や規則をプログラムとして入力していく．プログラムの最後では CTRL-Z キーを入力する．

```
| ?-[user].
| parent(P, C) :- father(P, C).
| parent(P, C) :- mother(P, C).
| grandfather(GF, C) :- parent(P, C), father(GF, P).
| ⟨CTRL − Z キーを押す⟩
yes
| ?-
```

入力したプログラムはすでにあるプログラムに加えられる.

```
| ?-listing.
father(naoyuki,hyogo).
father(saburo,naoyuki).
father(saburo,shinji).
father(yoshihisa,hisako).
mother(hisako,hyogo).
mother(yoko,naoyuki).
mother(yoko,shinji).
mother(nobuko,hisako).
parent(P,C) :-
   father(P,C).
parent(P,C) :-
   mother(P,C).
grandfather(GF,C) :-
   parent(P,C),
   father(GF,P).
yes
```

規則によって定義された述語に対しても，事実の場合と同様に質問をすることができる.

```
| ?-parent(P, hisako).
P       = yoshihisa;
P       = nobuko;
no
| ?-grandfather(GF, hyogo).
GF      = saburo
```

規則の場合も事実の場合と同様に，プログラム中で記述した規則の順序にしたがって答えが表示される．上の例では,「父親は親である」にあたる規則がプログラム中で先に現れているから，hisako の親として父親の yoshihisa が先に表示されている.

ここでさらに「男である」を意味する述語 male と,「女である」を意味する述語 female の事実を付け加える.

```
male(hyogo).
male(naoyuki).
male(shinji).
male(saburo).
male(yoshihisa).
female(hisako).
female(yoko).
female(nobuko).
```

練習問題

問 12　「S が P の息子である」を表す述語 `son(`S`,` P`)` を定義せよ.

問 13　「U が C のおじである」を表す述語 `uncle(`U`,` C`)` を定義せよ.

13　再帰的定義

Prolog でも Lisp と同様に再帰的定義が可能である．たとえば,「先祖である」という関係は次のように再帰的に定義される.

```
ancestor(A, C) :- parent(A, C).
ancestor(A, C) :- parent(P, C), ancestor(A, P).
```

1 行目の規則は「親は先祖である」を意味し，2 行目の規則は「親の先祖は先祖である」を意味する.

hyogo の先祖は次のようにして求められる.

```
| ?-ancestor(A, hyogo).
A       = naoyuki;
A       = hisako;
A       = saburo;
A       = yoko;
A       = yoshihisa;
A       = nobuko;
no
```

練習問題

問 14　「D が P の子孫である」を表す述語 `descendant(`D`,` P`)` を定義せよ.

14　リスト処理

これまでは，記号を取り扱うプログラムだけを説明してきたが，もちろん Prolog でリストを取り扱うこともできる．たとえば,

```
menu([tea,sugar,lemon]).
menu([coffee,sugar]).
```

という事実があったとき，次のような質問により，与えたパターンに一致する事実を捜し出せる.

```
| ?-menu(A).
A       = [tea,sugar,lemon];
A       = [coffee,sugar]
yes
| ?-menu([A]).
no
| ?-menu([A,B,C]).
A       = tea,
B       = sugar,
C       = lemon;
no
```

A は，[tea,sugar,lemon] と [coffee,sugar] のどちらとも一致する．[A] は，異なる長さのリストであるため，どちらにも一致しない．[A,B,C] は，長さ 3 のリスト [tea,sugar,lemon] とだけ一致する．

1 以上の任意の長さのリストと一致する事実を捜すためには，次のように質問する．

```
| ?-menu([A|B]).
A       = tea,
B       = [sugar,lemon];
A       = coffee,
B       = [sugar]
yes
| ?-menu([A,B|C]).
A       = tea,
B       = sugar,
C       = [lemon];
A       = coffee,
B       = sugar,
C       = []
yes
```

[A|B] の A がリストの先頭要素と一致し，B が残りのリストと一致する．[A,B|C] の場合は，A がリストの先頭要素，B が 2 番目の要素，C が残りのリストと一致する．

このような「一致」の処理は，**ユニフィケーション** (単一化) と呼ばれる (パターンマッチと呼ばれることもある)．Prolog では，リストなどの複雑な構造に対する処理 (要素の取り出し，構造の作成など) は，このユニフィケーションにより行うことができる．したがって，Lisp の CAR，CDR，CONS などの関数は必要ない．たとえば，次のように定義すればよい．

```
car([X|Y], X).
cdr([X|Y], Y).
cons(X, Y, [X|Y]).
```

```
| ?-car([tea,sugar,lemon], A).
A       = tea
yes
| ?-cdr([tea,sugar,lemon], A).
A       = [sugar,lemon]
yes
| ?-cons(ice, [tea,sugar,lemon], A).
A       = [ice,tea,sugar,lemon]
yes
```

2 つのリスト X と Y を連結したリスト Z を求める述語 append(X,Y,Z) は次のように定義できる.

```
append([], Z, Z).
append([W|X], Y, [W|Z]) :- append(X, Y, Z).
```

このプログラムは，リストの分解にも使用できる.

```
| ?-append([coffee,sugar], [tea,sugar,lemon], A).
A       = [coffee,sugar,tea,sugar,lemon]
yes
| ?-append(A, B, [tea,sugar,lemon]).
A       = [],
B       = [tea,sugar,lemon];
A       = [tea],
B       = [sugar,lemon];
A       = [tea,sugar],
B       = [lemon];
A       = [tea,sugar,lemon],
B       = [];
no
```

練習問題

問 15　リストの 2 番目の要素と 3 番目の要素を交換したリストを求める述語 ex23 を定義せよ.

15　応用プログラム例

Prolog のプログラム例として，簡単な英文の文法解析を行うプログラムを紹介する.

Time flies like an arrow.

という文は，文法的には次の 3 つの解釈が可能である.

(1) “Time” を名詞，“flies” を動詞，“like an arrow” を前置詞句とした解釈「時は矢のように飛ぶ」(光陰矢のごとし).

(2) “Time” を形容詞, “flies” を名詞, “like” を動詞とした解釈「時蝿は矢を好む」.
(3) “Time” を動詞 (命令文), “flies” を名詞 (目的語), “like an arrow” を前置詞句とした解釈「矢のように蝿を調整せよ」.

これらの 3 つの解析結果を出力する文法規則は Prolog で以下のように書ける.

```
文(文(NP,VP), S0, S) :-
    名詞句(NP, S0, S1), 動詞句(VP, S1, S).
文(文(VP), S0, S) :-
    動詞句(VP, S0, S).
名詞句(名詞句(N), S0, S) :-
    名詞(N, S0, S).
名詞句(名詞句(A,N), S0, S) :-
    形容詞(A, S0, S1), 名詞(N, S1, S).
名詞句(名詞句(D,N), S0, S) :-
    冠詞(D, S0, S1), 名詞(N, S1, S).
動詞句(動詞句(V), S0, S) :-
    動詞(V, S0, S).
動詞句(動詞句(V,NP), S0, S) :-
    動詞(V, S0, S1), 名詞句(NP, S1, S).
動詞句(動詞句(V,PP), S0, S) :-
    動詞(V, S0, S1), 前置詞句(PP, S1, S).
動詞句(動詞句(V,NP,PP), S0, S) :-
    動詞(V, S0, S1), 名詞句(NP, S1, S2), 前置詞句(PP, S2, S).
前置詞句(前置詞句(P,NP), S0, S) :-
    前置詞(P, S0, S1), 名詞句(NP, S1, S).
```

述語定義

```
文(文(NP,VP), S0, S) :-
    名詞句(NP, S0, S1), 動詞句(VP, S1, S).
```

は, 文法規則

文 → 名詞句 動詞句

に相当する.

辞書は次のように書ける.

```
名詞(名詞(time), [time|S], S).
名詞(名詞(flies), [flies|S], S).
名詞(名詞(arrow), [arrow|S], S).
動詞(動詞(time), [time|S], S).
動詞(動詞(flies), [flies|S], S).
動詞(動詞(like), [like|S], S).
形容詞(形容詞(time), [time|S], S).
前置詞(前置詞(like), [like|S], S).
冠詞(冠詞(an), [an|S], S).
```

実行結果は次のようになる (見やすいように整形してある).

```
| ?-文(T, [time,flies,like,an,arrow], []).
T       = 文(名詞句(名詞(time)),
             動詞句(動詞(flies),
                    前置詞句(前置詞(like),
                             名詞句(冠詞(an), 名詞(arrow))
          )));
T       = 文(名詞句(形容詞(time), 名詞(flies)),
             動詞句(動詞(like), 名詞句(冠詞(an), 名詞(arrow)))
          );
T       = 文(動詞句(動詞(time),
                    名詞句(名詞(flies)),
                    前置詞句(前置詞(like),
                             名詞句(冠詞(an), 名詞(arrow))
          )));
no
```

16 参考文献

1. 後藤滋樹 著,「PROLOG 入門 — 知識情報処理の序曲」, ソフトウェア ライブラリ 1, サイエンス社
   (Prolog についてのやさしい入門書)

2. 塚本龍男 著,「わかる:-Prolog」, 情報処理入門シリーズ 12, 共立出版
   (楽しい例題を通じて Prolog を自然に学べる入門書)

3. 黒川利明 著,「Prolog のソフトウェア作法」, 岩波コンピュータサイエンス, 岩波書店
   (Prolog の入門から高度なプログラミングまで, いろいろなトピックを取り上げてある)

4. 黒川利明 田村直之 著,「Prolog プログラミング入門」, KE 養成講座 6, オーム社
   (Prolog 入門, プログラミング技法, 論理学との関係などについてまとめてある)

5. コワルスキ 著, 浦昭二 監, 山田眞市 ほか訳,「論理による問題の解法 — Prolog 入門」, 情報処理シリーズ 8, 培風館
   (Prolog の創始者コワルスキの教科書)

6. スターリング ほか著, 松田利夫 訳,「Prolog の技芸」, 共立出版
   (Prolog のプログラミングテクニックの標準的教科書)

7. 高野真 著,「Prolog で学ぶ AI 手法 — 推論システムと自然言語処理」, 啓学出版
   (プロダクションシステム, 自然言語インターフェイスなどについて, 実際にシステムをプログラムリストを含めて紹介している)

8. 後藤滋樹 著,「記号処理プログラミング」, 岩波講座 ソフトウェア科学 8, 岩波書店
   (LISP と Prolog についての教科書)

Prolog 練習問題解答

問 1　大文字で始まる名前は変数であるから，Taro で太郎を表すことはできない．具体的なものを指し示すには記号を使用する必要がある．

問 2　“father(hyogo，yoko)．” という事実を付け加えるのは間違いである．hyogo の子供の yoko と，naoyuki の母親の yoko とは別人なのだから，別の記号で表す必要がある．したがって，たとえば hyogo の子供の yoko を記号 yoko2 で表すことにして，“father(hyogo，yoko2)．” という事実を付け加えることになる．

問 3　付け加えることができる．Prolog システムは常識を持ち合わせていないから，どんな事実でも付け加えることができる．

問 4

```
?-mother(M, hyogo), father(G, M).
```

問 5　naoyuki の父親の子供を尋ねている．おおまかには，naoyuki の兄弟姉妹を尋ねていると考えて良いが，答えに naoyuki 自身も含まれてしまう点が異なる．兄弟姉妹を尋ねるには，X と Y とが異なることを意味する**組込み述語** X/==Y を利用して

```
?-father(F, naoyuki), father(F, C), C/==naoyuki.
```

とすればよい．

問 6　まず father(F，naoyuki) の答え F=saburo が求められ，次に father(saburo，C) の最初の答え C=naoyuki が得られ，全体の最初の答えとして F=sabruo，C=naoyuki が表示される．ユーザがセミコロンを入れると，father(saburo，C) の次の答え C=shinji が得られ，全体の答えとして F=sabruo，C=shinji が表示される．ユーザがさらにセミコロンを入れると，father(saburo，C) の次の答えを求めようとするが，存在しないので後戻りし，father(F，naoyuki) の次の答えを求めようとする．これも存在しないので質問全体が失敗し no が表示される．

問 7　たとえば，

```
grandmother(GM, C) :- parent(P, C), mother(GM, P).
```

もちろん，右辺のコンマの前後が逆でもよい．

問 8　たとえば，

```
child(C, P) :- parent(P, C).
```

問 9　たとえば，

```
grandchild(C, G) :- parent(C, P), parent(P, C).
```

問 10　たとえば，

```
sibling(X, Y) :- father(F, X), father(F, Y), X /== Y.
```

問 11　「X が男である」を意味する述語 male(X) がすでに定義されていれば，それを使って，たとえば

```
brother(X, Y) :- sibling(X, Y), male(X).
```

とできる．

問 12　たとえば，

```
son(S, P) :- child(S, P), male(S).
```

問 13 たとえば,

```
uncle(U, C) :- parent(P, C), brother(P, U).
```

問 14 たとえば, 次のプログラム

```
descendant(D, P) :- child(D, P).
descendant(D, P) :- child(C, P), descendant(D, C).
```

でもよいが, より簡単に

```
descendant(D, P) :- ancestor(P, D).
```

などとできる.

問 15 たとえば,

```
ex23([A,B,C|D], [A,C,B|D]).
```