高知大学教育学部の情報数学のテキスト

文責　：　高知大学名誉教授　中村　治

Prolog プログラミング

まず Prolog (GNU Prolog) から始めます。論理型のプログラミング言語 PROLOG (PROgramming in LOGic) は 1972 年に A. Colmerauer によってフランスで開発されます。他の言語のように手続きを記述するのではなく、事実（物事の論理的関係）を記号論理に基づいて記述し、パターン・マチィングとバックトラッキングでプログラムを実行します。次に紹介する LISP と同じくリスト処理に優れ人工知能のための言語で、日本の第五世代コンピュータの開発用言語としても使われました。しばらく忘れ去られていましたが、Bruce A. Tate 著「7 つの言語　7 つの世界」Ruby, Io, Prolog, Scala, Erlang, Clojure, and Haskell 平成 23 年　3200 円の影響で再び注目されています。

無料の処理系が色々ありますが、日本語が使えないので不便ですが数独のプログラムを実行するために、ここでは GNU Prolog を使います。

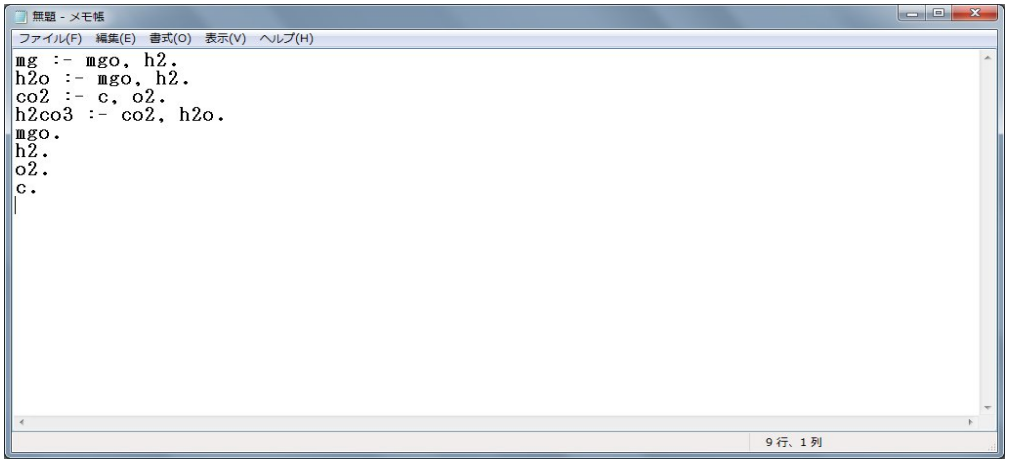
化学反応

まず単純な Prolog のプログラムを作ります。最初のプログラムはいくつかの化学反応式と存在する物質を Prolog のプログラムとして表現したものです。

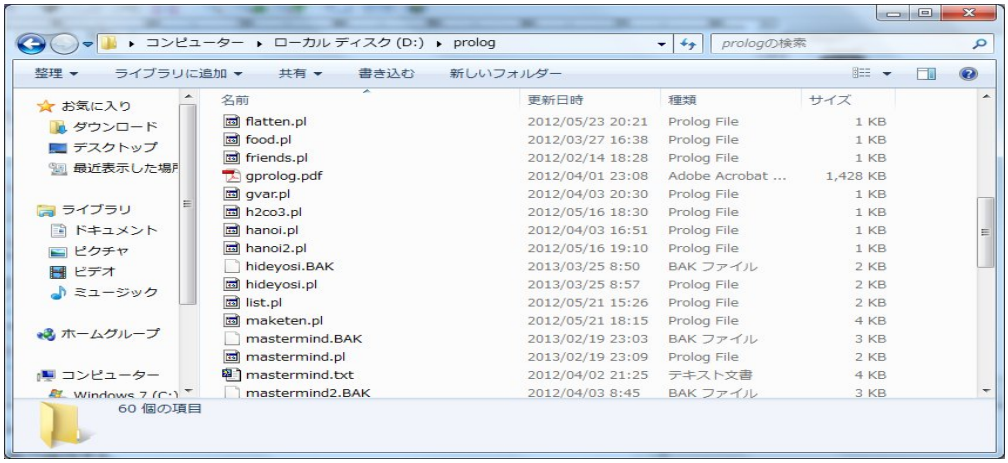
アクセサリの「メモ帳」を起ち上げて、

```
mg :- mgo, h2.
h2o :- mgo, h2.
co2 :- c, o2.
h2co3 :- co2, h2o.
mgo.
h2.
o2.
c.
```

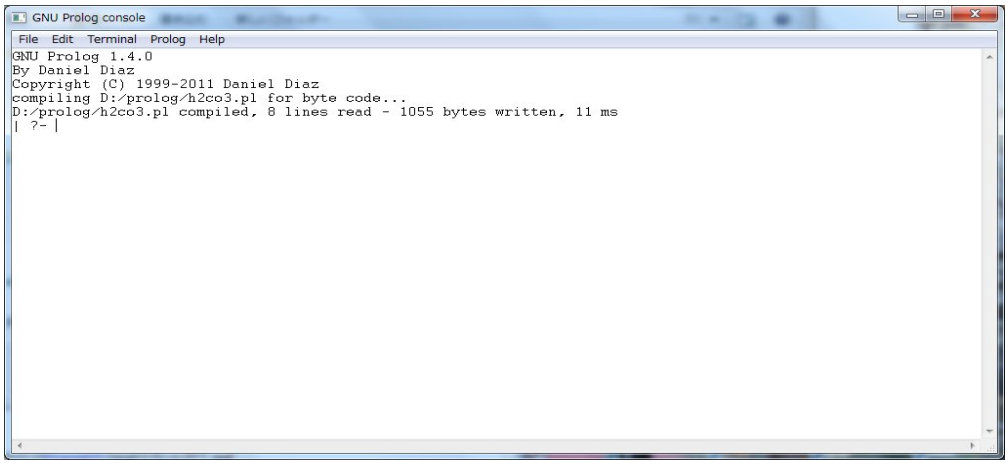
と打ち込み、



h2co3.pl と言う名前で保存する。拡張子の .pl が Prolog のプログラムであるという印です。



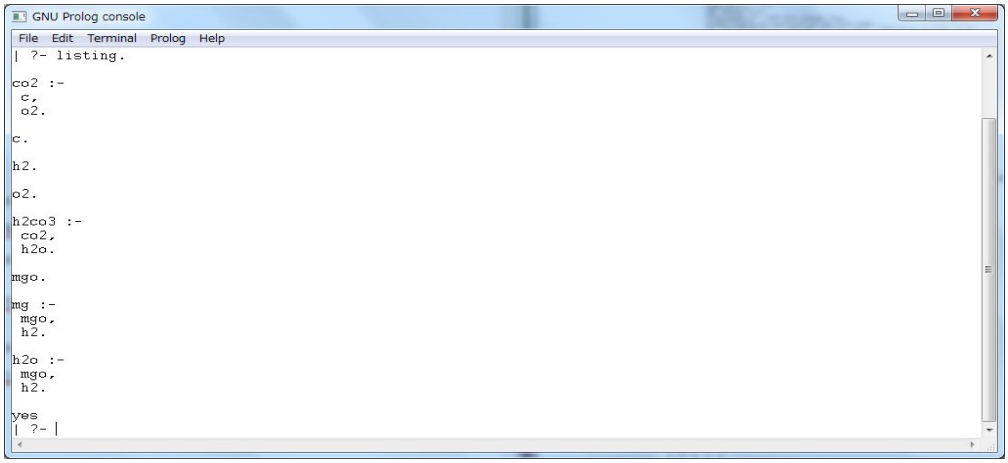
h2co3.pl をダブルクリックする。



Prolog　が立ち上がり、h2co3.pl のプログラムが Prolog に読み込まれます。

```
| ?- listing.
```

としてみる。これは読み込んだプログラムを表示しなさいという命令（質問）です。



```
| ?- listing.

co2 :-
 c,
 o2.

c.

h2.

o2.

h2co3 :-
 co2,
 h2o.

mgo.

mg :-
 mgo,
 h2.

h2o :-
 mgo,
 h2.

yes
| ?-
```

と覚えているデータベース（プログラム）を表示する。listing. と最後のピリオドを忘れないようにして下さい。Prolog のプログラムはピリオドで終わります。このプログラムが何を意味するかを解説する前に Prolog を使ってみましょう。

```
| ?- mgo.
```

と打ち込むと、yes と答える。

```
| ?- h2.
```

と打ち込むと、yes と答える。

```
| ?- au.
```

と打ち込むと、


```
uncaught exception: error(existence_error(procedure,au/0),top_level/0)
```

と表示する。

Prolog のプログラムの

```
mg :- mgo, h2.
h2o :- mgo, h2.
co2 :- c, o2.
h2co3 :- co2, h2o.
mgo.
h2.
o2.
c.
```

の

```
mgo.
```

で酸化マグネシュウム (MgO) が存在するという事実を述べています。MgO のように大文字で始めると、 Prolog では大文字で始まる文字列は変数を表すことになっているので、mgo と小文字を使って酸化マグネシュウム (MgO) を表現します。mgo のようなものをアトム (atom) と呼びます。mgo の次にピリオドを必ず打ちます。これで Prolog の文が終わります。同様に

```
h2.
```

は水素 (H2) が存在するという事実を述べています。

```
o2.
```

は酸素 (O2) が存在するという事実を述べています。

```
c.
```

は炭素 (C)　が存在するという事実を述べている。
　Prolog ではプログラムを読み込んだら、そのプログラムに基づいて質問をしていきます。

```
| ?-
```

は Prolog が質問を受け付ける準備が出来たというプロンプト（入力促進記号）です。

```
| ?- mgo.
```

と入力すると酸化マグネシュウム (MgO) は存在するか？と質問したことになります。酸化マグネシュウム (MgO) は存在する

```
mgo.
```

とプログラムにかいてあるので

```
| ?- mgo.
```

と質問すると、yes と答えたわけです。

```
| ?- h2.
```

という質問にも、プログラムに

```
h2.
```

と書いてあるので yes と答えるます。

```
| ?- au.
```

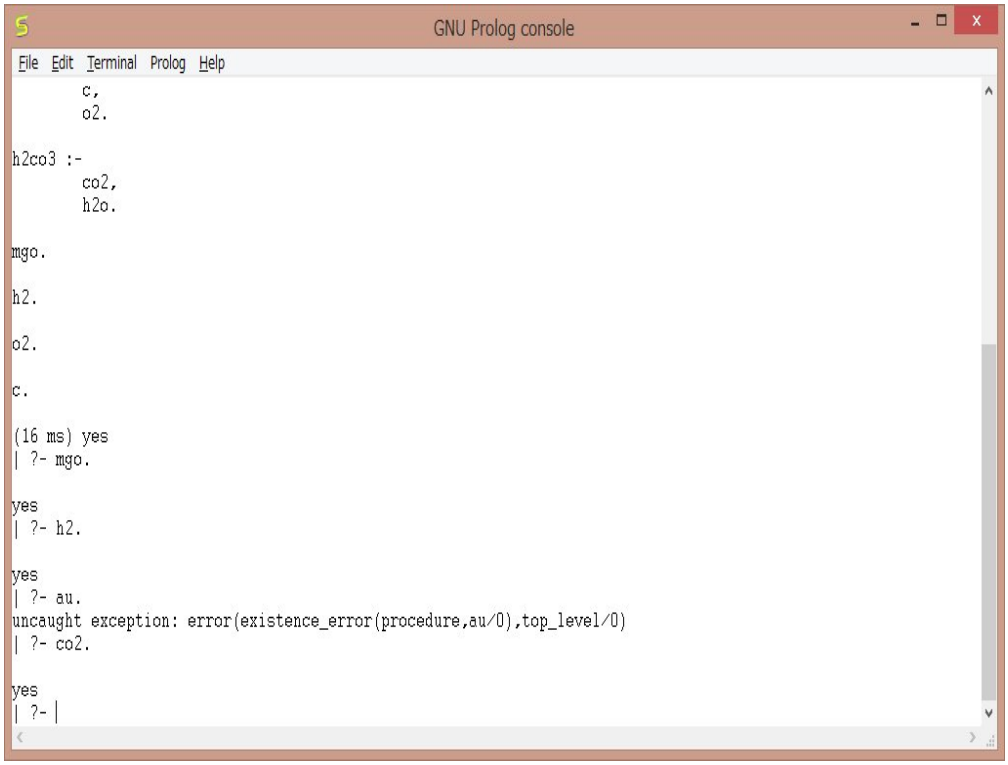
という質問には、プログラムに

```
au.
```

という記述がないので、au は知らないと Prolog がエラーメッセージを出しました。次に

```
| ?- co2.
```

と質問すると、yes と答えます。



プログラムに

```
co2.
```

という記述がないのにどうして yes と答えたのでしょうか？ au の場合と異なり、

```
co2.
```

という記述（二酸化炭素 CO2 が存在する）はないですが、プログラムをよく読むと

```
co2 :- c, o2.
```

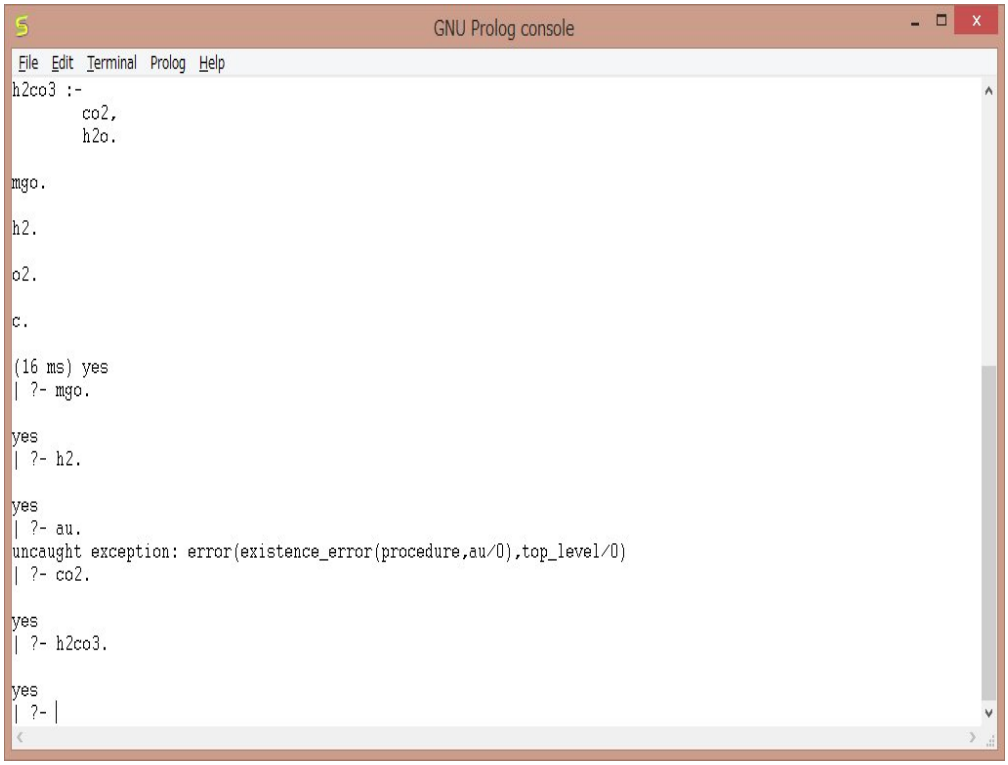
という記述があります。これは co2 が存在するためには c と o2 があればよいという記述です。これは炭素 (C) と酸素 (O2) が反応して二酸化炭素 (CO2) が出来るという化学反応式を Prolog で表現したものです。この記述により、二酸化炭素 (CO2) が存在するかという質問は炭素 (C) と酸素 (O2) が存在するかという質問に置き換えられます。プログラムには

```
o2.
c.
```

という記述があるので、Prolog は yes と答えたのです。

```
| ?- h2co3.
```

と質問すると今度も yes と答えます。



今回も

```
h2co3.
```

という記述はないですが

```
h2co3 :- co2, h2o.
```

という記述があるので、h2co3 は存在するかという質問は、co2 と h2o は存在するかという質問に置き換わります。co2 は上でみたように存在する（与えられている化学反応で c と o2 で作れる）ので、h2o は存在するかを Prolog は調べます。

```
h2o.
```

という記述はないですが、

```
h2o :- mgo, h2.
```

という記述があります。これは酸化マグネシウム (MgO) と水素 (H2) が反応して、マグネシウム (Mg) と水 (H2O) が出来る言う化学反応を

```
mg :- mgo, h2.
h2o :- mgo, h2.
```

という二行（酸化マグネシウム (MgO) と水素 (H2) があればマグネシウム (Mg) が存在すると酸化マグネシウム (MgO) と水素 (H2) があれば水 (H2O) が存在する）で表現しています。従って、水 (H2O) が存在するためには酸化マグネシウム (MgO) と水素 (H2) があれば良いことになります。

```
mgo.
h2.
```

の記述があるので、Prolog は yes と答えたわけです。
　整理してみると

```
mg :- mgo, h2.
h2o :- mgo, h2.
co2 :- c, o2.
h2co3 :- co2, h2o.
mgo.
h2.
o2.
c.
```

のプログラムで

```
| ?- h2co3.
```

と質問する。

```
yes
```

と答える。これは、プログラム（データベース）で ヘッド（head） が h2co3 であるものを探す。

```
h2co3 :- co2, h2o.
```

が見つかるので、ボディ（body）

```
co2, h2o.
```

の co2 と h2o が成り立つかを調べる。

```
h2co3 :- co2, h2o.
```

の h2co3 をヘッド（head） といい、co2, h2o. をボディ（body）といいます。ゴール（goal）が

```
co2, h2o.
```

を調べることになりました。まず、 ヘッド（head） が co2 であるものをプログラム（データベース）で調べる。

```
co2 :- c, o2.
```

が見つかるので、co2 を ボディ（body）

```
c, o2.
```

で置き換えて

```
c, o2, h2o.
```

の c と o2 と h2o が成り立つかどうか調べる。まず、 ヘッド（head） が c であるものをプログラム（データベース）で調べる。

```
c.
```

が見つかるので、残りの

```
o2, h2o.
```

の o2 と h2o が成り立つかどうか調べる。ヘッド（head） が o2 であるものをプログラム（データベース）で調べる。

```
o2.
```

があるので、最後に ヘッド（head） が h2o であるものをプログラム（データベース）で調べる。

```
h2o :- mgo, h2.
```

があるので、goal の

```
mgo, h2.
```

が成り立つか調べる。

```
mgo.
```

があるので、ヘッド（head） が h2 であるものをプログラム（データベース）で調べる。

```
h2.
```

があるので、すべて成り立った。それで Prolog は yes と答えた。

問題：

料理のレシピと現在ある食材で、作れる食事を判定するプログラムを作れ。

家族関係を調べる

アクセサリの「メモ帳」を起ち上げて、 少し長いですが

```
child(toyotomi_hideyori, yodonokata).
child(toyotomi_hideyori, toyotomi_hideyosi).
child(turumatsu, yodonokata).
child(turumatsu, toyotomi_hideyosi).
child(yodonokata, oichi).
child(yodonokata, azai_nagamasa).
child(ohatsu, oichi).
child(ohatsu, azai_nagamasa).
child(ogou, oichi).
child(ogou, azai_nagamasa).
child(toyotomi_hideyosi, kinosita_yaemon).
child(toyotomi_hideyosi, naka).
child(toyotomi_hidenaga, takeami).
child(toyotomi_hidenaga, naka).
child(asahi, takeami).
```

```
child(asahi, naka).
child(tomo, kinosita_yaemon).
child(tomo, naka).
child(toyotomi_hidetsugu, miyosi_yosihusa).
child(toyotomi_hidetsugu, tomo).
child(senhime, tokugawa_hidetada).
child(senhime, ogou).
child(tokugawa_iemitsu, tokugawa_hidetada).
child(tokugawa_iemitsu, ogou).
male(toyotomi_hideyori).
male(turumatsu).
male(toyotomi_hideyosi).
male(toyotomi_hidenaga).
male(toyotomi_hidetsugu).
male(tokugawa_iemitsu).
male(azai_nagamasa).
male(kinosita_yaemon).
male(takeami).
male(tokugawa_hidetada).
female(yodonokata).
female(oichi).
female(ohatsu).
female(ogou).
female(naka).
female(tomo).
female(senhime).
father(X, Y) :- child(Y, X), male(X).
mother(X, Y) :- child(Y, X), female(X).
father(X) :- child(Y, X), male(X).
mother(X) :- child(Y, X), female(X).
ancestor(X, Y) :- child(Y, X).
ancestor(X, Y) :- child(Z, X), ancestor(Z, Y).
descendant(X, Y) :- child(X, Y).
descendant(X, Y) :- child(X, Z), descendant(Z, Y).
```

と打ち込み、family.pl という名前で保存する。
family.pl をダブルクリックして、GNU Prolog を起ち上げる。

```
child(toyotomi_hideyori, yodonokata).
child(toyotomi_hideyori, toyotomi_hideyosi).
child(turumatsu, yodonokata).
child(turumatsu, toyotomi_hideyosi).
child(yodonokata, oichi).
child(yodonokata, azai_nagamasa).
```

```
child(ohatsu, oichi).
child(ohatsu, azai_nagamasa).
child(ogou, oichi).
child(ogou, azai_nagamasa).
child(toyotomi_hideyosi, kinosita_yaemon).
child(toyotomi_hideyosi, naka).
child(toyotomi_hidenaga, takeami).
child(toyotomi_hidenaga, naka).
child(asahi, takeami).
child(asahi, naka).
child(tomo, kinosita_yaemon).
child(tomo, naka).
child(toyotomi_hidetsugu, miyosi_yosihusa).
child(toyotomi_hidetsugu, tomo).
child(senhime, tokugawa_hidetada).
child(senhime, ogou).
child(tokugawa_iemitsu, tokugawa_hidetada).
child(tokugawa_iemitsu, ogou).
```

の部分は、親子関係を表現していて、例えば、

```
child(toyotomi_hideyori, yodonokata).
```

は、`toyotomi_hideyori` は yodonokata の子供であるという事実を表している。

```
child(toyotomi_hideyori, yodonokata).
```

は

```
child (toyotomi_hideyori, yodonokata).
```

のように child と (の間にはスペースを入れてはいけません。また、

```
child(toyotomi_hideyori, yodonokata).
```

の `child (toyotomi_hideyori, yodonokata)` をヘッド（head）と言う。

```
male(toyotomi_hideyori).
male(turumatsu).
male(toyotomi_hideyosi).
male(toyotomi_hidenaga).
male(toyotomi_hidetsugu).
male(tokugawa_iemitsu).
male(azai_nagamasa).
male(kinosita_yaemon).
male(takeami).
male(tokugawa_hidetada).
```

の部分は、例えば、

```
male(toyotomi_hideyori).
```

は、toyotomi_hideyori は男性であるという事実を表している。

```
female(yodonokata).
female(oichi).
female(ohatsu).
female(ogou).
female(naka).
female(tomo).
female(senhime).
```

の部分は、例えば、

```
female(yodonokata).
```

は、yodonokata は女性であるという事実を表している。

```
father(X, Y) :- child(Y, X), male(X).
```

は、ルール (規則) を Prolog に教えている箇所で、「child(Y, X) かつ male(X)」であれば、「father(X, Y)」であるという規則を表現している。あるいは、「father(X, Y)」であるためには、「child(Y, X) かつ male(X)」でなければならないということを表現している。つまり、「X が Y の父親である」ためには、「Y が X の子供でありかつ X が男である」事が成り立てばよいと Prolog に教えているわけである。X や Y は大文字で表現されているが、これは変数であることを Prolog に教えている。これ以外の father や yodonokata はアトムと呼ばれ、小文字で始める必要がある。

```
father(X, Y) :- child(Y, X), male(X).
```

の father(X, Y) はヘッド (head) で、child(Y, X) と male(X) はボデー (body) です。

同様に、

```
mother(X, Y) :- child(Y, X), female(X).
```

は、「X が Y の母親である」ためには、「Y が X の子供でありかつ X が女である」事が成り立てばよいと Prolog に教えている。

```
father(X) :- child(Y, X), male(X).
```

は、「X が父親である」ためには、「誰か Y が X の子供でありかつ X が男である」事が成り立てばよいと Prolog に教えている。ただこの場合、知りたいのは、X が父親かどうかで、X の子供が誰かではないので、Y の値には興味がない。ともかく誰か X の子供がいればよい。このような場合、無名変数 _ を使うのが普通である。従って、

```
father(X) :- child(_, X), male(X).
```

と定義するのが普通である。

```
mother(X) :- child(Y, X), female(X).
```

は、「X が母親である」ためには、「誰か Y が X の子供でありかつ X が女である」事が成り立てばよいと Prolog に教えている。これも無名変数 _ を使って

```
mother(X) :- child(_, X), female(X).
```

と定義するのが普通です。

```
ancestor(X, Y) :- child(Y, X).
ancestor(X, Y) :- child(Z, X), ancestor(Z, Y).
```

は、「X が Y の先祖である」というのは、「Y が X の子供である」かまたは「誰か Z が X の子供で、かつ、 Z が Y の先祖である」という事だと Prolog に教えている。Prolog はプログラムを最初から順番に調べるので、同じヘッド (head) のルールが並ぶ場合、このような再帰的な定義（ancestor(X, Y) を説明するために、ancestor(Z, Y) と同じ ancestor を使うことを再帰的な定義といいます）のときは、Prolog ではその順番が非常に重要で、終了条件（「Y が X の子供である」のような再帰的なルールでないもの）が先に来るように、この順序で定義しなければならないです。

```
descendant(X, Y) :- child(X, Y).
descendant(X, Y) :- child(X, Z), descendant(Z, Y).
```

は、「X が Y の子孫である」とは、「X が Y の子供である」かまたは「X が誰か Z の子供で、かつ、Z が Y の子孫である」という事だと Prolog に教えている。これも終了条件が先に来るように、この順序で定義しなければならない。

このような事実と規則 (ルール）の元で、Prolog に質問をする。

次のような質問をする。

```
| ?- child(tokugawa_iemitsu, ogou).
```

これは tokugawa_iemitsu は ogou の子供かを質問したことで、child(tokugawa_iemitsu, ogou) と言う事実がプログラム（データベース）にあるかまたはルールと事実から導けるかを質問したことになる。child() が ヘッド (head) に含まれている 事実やルールを定義されている順番に調べていく。最初に見つかるのは

```
child(toyotomi_hideyori, yodonokata).
```

であるが、child(tokugawa-iemitsu, ogou) と child(toyotomi_hideyori, yodonokata) では括弧の中身が一致しない。両方一致するのは最後の

```
child(tokugawa_iemitsu, ogou).
```

である。事実が見つかったので

```
| ?- child(tokugawa_iemitsu, ogou).

yes
| ?-
```

と yes と答え、次の質問を促すプロンプトを表示する。

次のような質問をする。

```
| ?- child(X, ogou).
```

これは child(X, ogou) が成り立つ X を見付けるよう Prolog に指示したことになる。つまり、ogou（お江）の子供は誰かと質問したことになる。するとプログラム（データベース）を順に調べ

```
child(senhime, ogou).
```

をまず見つける。child(X, ogou) と child(senhime, ogou) は X に senhime を代入すれば一致する。そこで

```
| ?- child(X, ogou).

X = senhime ?
```

と答える。このような場合、? で終わっているのは別解が必要か聞いているので

```
X = senhime ? ;
```

と ; を打ち込むと別の子供は誰かと質問したことになる。すると

```
child(senhime, ogou).
```

の続きからプログラム（データベース）を調べ、

```
child(tokugawa_iemitsu, ogou).
```

を見つけ、child(X, ogou) と `child(tokugawa_iemitsu, ogou)` は X に `tokugawa_iemitsu` を代入すれば一致するので

```
| ?- child(X, ogou).

X = senhime ? ;

X = tokugawa_iemitsu

(31 ms) yes
| ?-
```

と答え、次の質問のためのプロンプトを表示する。; は別解を求めさせることを意味する。
次のような質問をする。

```
| ?- child(X, azai_nagamasa).
```

すると

```
| ?- child(X, azai_nagamasa).

X = yodonokata ?
```

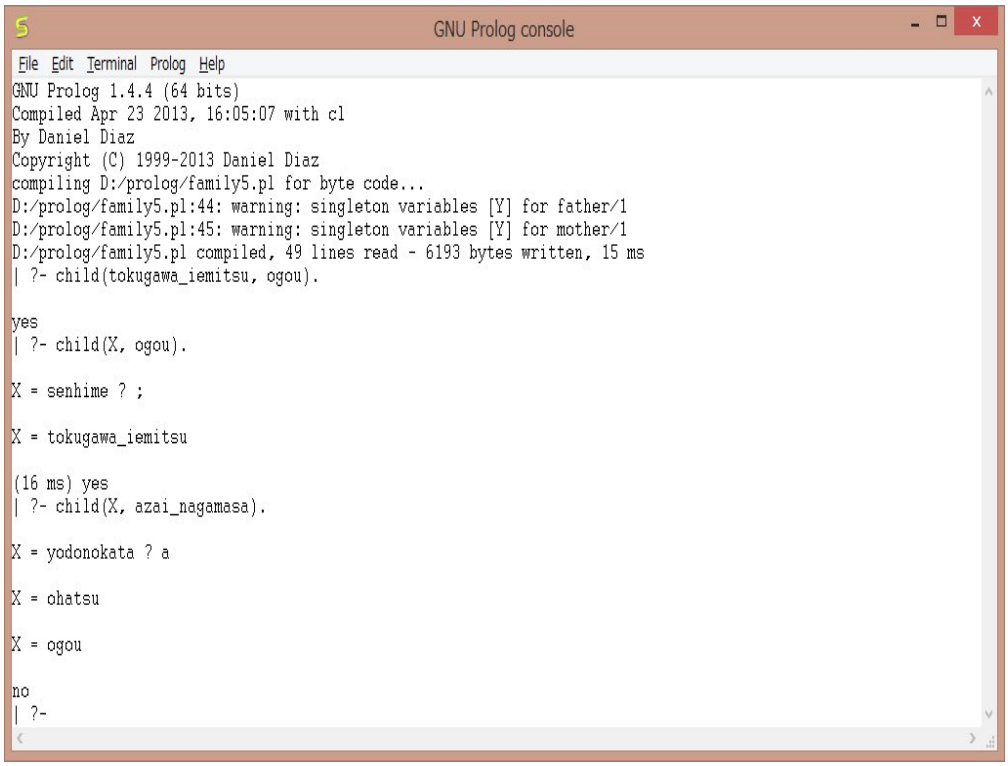
と答える。今度は

```
X = yodonokata ? a
```

と a を打ち込む。すると

```
| ?- child(X, azai_nagamasa).

X = yodonokata ? a

X = ohatsu

X = ogou

no
| ?-
```

と答える。a はすべての別解を求めさせることを意味する。



次のような質問をする。

```
| ?- child(senhime, X), female(X).
```

すると

```
X = ogou
```

と答える。

これは、「child(senhime, X)」と「female(X)」と両方を満たす X は何かと質問したことになる。「senhime は X の子供」で「X は女」の両方を満たす X は何か、すなわち、「senhime（千姫） の母親は誰か？」と質問したことになる。答えは、X = ogou （お江）である。これは、

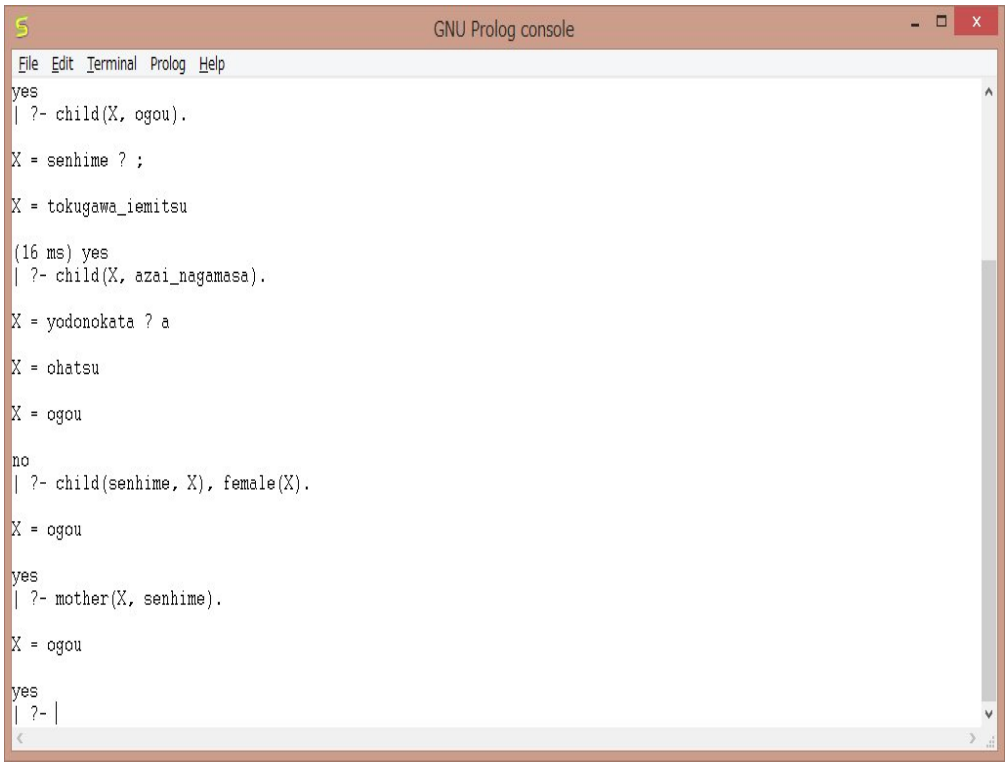
```
mother(X, Y) :- child(Y, X), female(X).
```

と母親であるための規則を教えているので、次のような質問をしたことと同じである。

```
| ?- mother(X, senhime).
```

```
X = ogou
```

と答える。




```
| ?- child(senhime, X), female(X).
```

と質問したときの Prolog の動きは次のようになる。まず child(senhime, X) とパターンマッチングするヘッド (head) を探す。

```
child(senhime, tokugawa_hidetada).
```

をまず見つける。child(senhime, X) と `child(senhime, tokugawa_hidetada)` は X に `tokugawa_hidetada` を代入すると一致するので、次のゴールである female(X) の X に `tokugawa_hidetada` を代入した `female(tokugawa_hidetada)` が成り立つかどうか調べる。しかし、`female(tokugawa_hidetada)` にパターンマッチングする（`female(tokugawa_hidetada)` そのものか変数に適当に代入すれば一致する）ヘッド (head) は存在しない。そこで、

```
child(senhime, tokugawa_hidetada).
```

の後ろで child(senhime, X) とパターンマッチングするヘッド (head) を探す。これをバックトラッキングと言います。

```
child(senhime, ogou).
```

を見つけ、child(senhime, X) と child(senhime, ogou) は X に ogou を代入すると一致するので、次のゴールである female(X) の X に ogou を代入した female(ogou) が成り立つかどうか調べる。female(ogou) とパターンマッチングするヘッド (head) を探す。

```
female(ogou).
```

を見つける。これで全部成り立つことを確かめたので、Prolog は

```
| ?- child(senhime, X), female(X).

X = ogou

yes
| ?-
```

と答えた。一方

```
| ?- mother(X, senhime).
```

と質問した場合は、mother(X, senhime) とパターンマッチングするヘッド (head) を探し、

```
mother(X, Y) :- child(Y, X), female(X).
```

を見つけ、

```
mother(X, senhime) :- child(senhime, X), female(X).
```

と Y に senhime を代入し、mother(X, senhime) を child(senhime, X), female(X) と置き換え、

```
child(senhime, X), female(X).
```

が成り立つかどうかを調べる。後の動きは上と同じです。ここで、

```
mother(X, Y) :- child(Y, X), female(X).
```

は Prolog の内部では X と Y は名前が競合しないように適当に書き換えられています。

X は Y の姉妹（sister）であると言うルールを

```
sister(X, Y) :- female(X), child(X, Z), child(Y, Z).
```

で定義してみます。family.pl の最後に追加し、再度 Prolog を起ち上げるか

```
| ?- assertz((sister(X, Y) :- female(X), child(X, Z), child(Y, Z))).
```

で一時的に追加します。

```
| ?- sister(senhime, X).
```

と質問すると

```
| ?- sister(senhime, X).

X = senhime ?
```

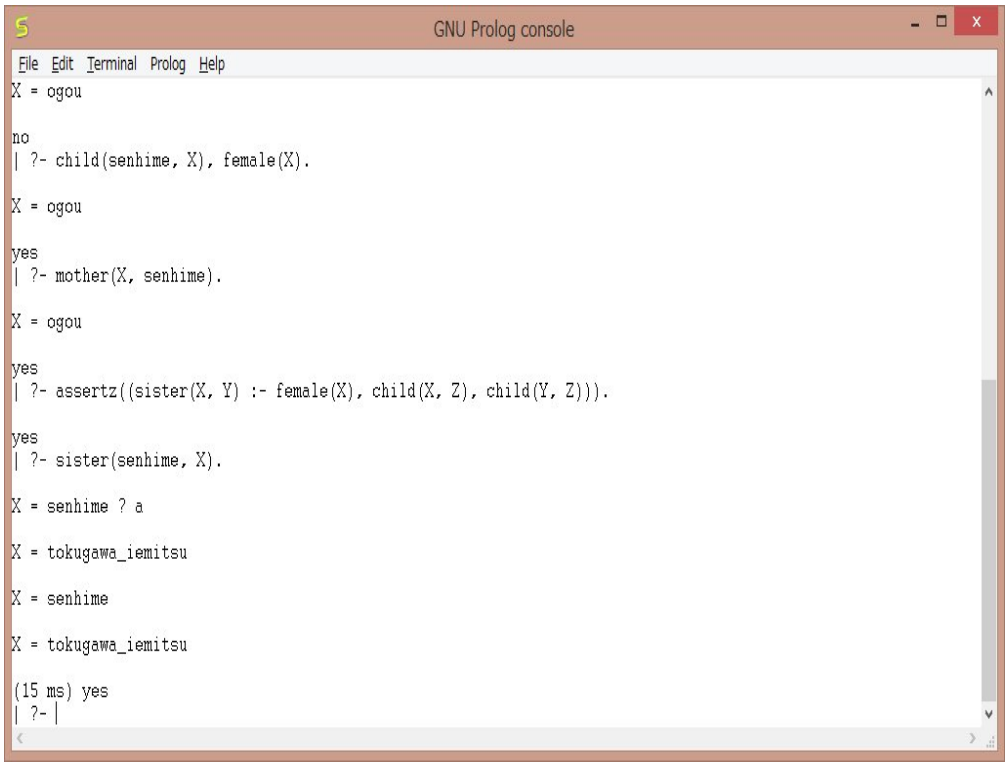
と答えます。

```
| ?- sister(senhime, X).

X = senhime ? a
```

と別解を求めると

```
| ?- sister(senhime, X).

X = senhime ? a

X = tokugawa_iemitsu

X = senhime

X = tokugawa_iemitsu

yes
| ?-
```

と答えます。



これには 2 つ問題があります。1 つは senhime の姉妹が senhime になっていることです。Prolog の動きは

```
| ?- sister(senhime, X).
```

の質問に対して、

```
sister(X, Y) :- female(X), child(X, Z), child(Y, Z).
```

のルールで示すべき目標を

```
female(sennhime), child(sennhime, Z), child(X, Z).
```

に代え、

```
female(sennhime).
```

と言う事実があるので

```
child(sennhime, Z), child(X, Z).
```

を新たな目標にし、

```
child(senhime, tokugawa_hidetada).
```

を見つけ、Z に tokugawa_hidetada を代入し

```
child(X, tokugawa_hidetada).
```

を新たな目標にし、

```
child(senhime, tokugawa_hidetada).
```

を見つけ、X に senhime を代入すれば一致するので

```
| ?- sister(senhime, X).

X = senhime ?
```

と答えました。これを除くには X と Y は異なるということを追加しておけばいいです。即ち、

```
sister(X, Y) :- female(X), child(X, Z), child(Y, Z), X \= Y.
```

とすればいいです。もう 1 つの問題は同じ答えが二度出てくることです。Z として tokugawa_hidetada を代入する場合と Z として ogou を代入する場合と二度合致するからです。これを避けるには例えば

```
parents(X, Y, Z) :- female(Y), child(X, Y), male(Z), child(X, Z).
```

と X の母が Y で、父が Z であるというルールを新たに定義し、

```
sister(X, Y) :- female(X), parents(X, Z, W), parents(Y, Z, W), X \= Y.
```

と定義すれば良いです。

```
| ?- retract((sister(X, Y) :- female(X), child(X, Z), child(Y, Z))).

yes
| ?- assertz((parents(X, Y, Z) :- female(Y), child(X, Y), male(Z), child(X, Z))).

yes
| ?- assertz((sister(X, Y) :- female(X), parents(X, Z, W), parents(Y, Z, W), X \= Y)).

yes
| ?- sister(senhime,X).

X = tokugawa_iemitsu ? ;

(16 ms) no
| ?-

| ?- sister(asahi, X).

no
| ?-
```

となります。これは

```
female(asahi).
```

が定義されていないからです。この場合

```
| ?- assertz(female(asahi)).
uncaught exception: error(permission_error(modify,static_procedure,female/1),assertz/1)
| ?-
```

となり、assertz() では追加できません。family.pl を修正します。

```
female(asahi).
parents(X, Y, Z) :- female(Y), child(X, Y), male(Z), child(X, Z).
sister(X, Y) :- female(X), parents(X, Z, W), parents(Y, Z, W), X \= Y.
```

を追加します。追加する時、

```
female(asahi).
parents(X, Y, Z) :- female(Y), child(X, Y), male(Z), child(X, Z).
sister(X, Y) :- female(X), parents(X, Z, W), parents(Y, Z, W), X \= Y.
```

をそのまま、family.pl の最後に追加するとどういうわけか

```
female(asahi).
```

を認識しません。

```
female(asahi).
```

は、female(). のグループの最後に追加して下さい。

```
| ?- sister(asahi,X).

X = toyotomi_hidenaga ? ;

(16 ms) no
| ?-
```

となります。toyotomi_hideyosi がでません。
　母親または父親が異なる場合はどうすれば良いでしょうか？

```
sister(X, Y) :- female(X), parents(X, Z, W1), parents(Y, Z, W2), X \= Y, W1 \= W2.
sister(X, Y) :- female(X), parents(X, Z1, W), parents(Y, Z2, W), X \= Y, Z1 \= Z2.
```

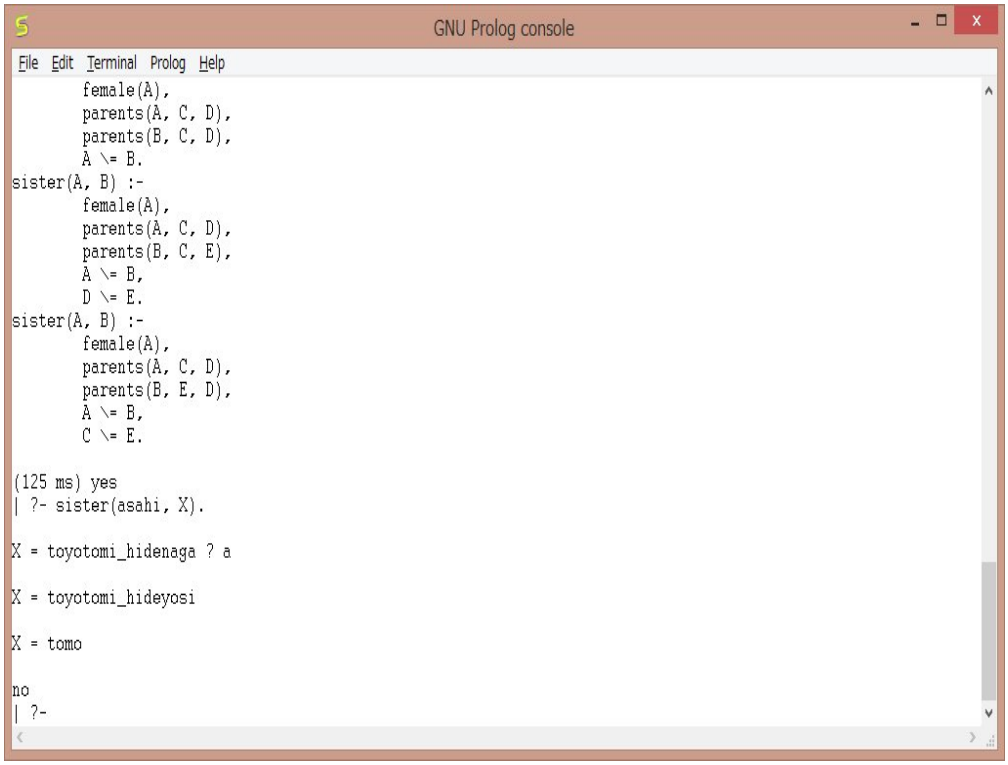
を追加し、sister(X, Y) の定義を

```
sister(X, Y) :- female(X), parents(X, Z, W), parents(Y, Z, W), X \= Y.
sister(X, Y) :- female(X), parents(X, Z, W1), parents(Y, Z, W2), X \= Y, W1 \= W2.
sister(X, Y) :- female(X), parents(X, Z1, W), parents(Y, Z2, W), X \= Y, Z1 \= Z2.
```

とします。

```
| ?- sister(asahi, X).

X = toyotomi_hidenaga ? a

X = toyotomi_hideyosi

X = tomo

no
| ?-
```

と表示するようになりました。



問題：

親（parent）、娘（daughter）、息子 (son)、祖父（grandfather）、祖母（groundmother）、孫（grandchild）、兄または弟（brother）、兄または弟または姉または妹（sibling）、伯母または叔母（aunt）、伯父または叔父（uncle）、姪（niece）、甥（nephew）、従兄弟（cousin）はどのように定義すればよいか？

X と Y が異なることを `\+(X=Y)` で表現することもできます。`\+` で not を表しています。これは GNU Prolog の特別な記述法です。

自然数の定義及び基本演算

自然数を Prolog に教えてみましょう。「0 は自然数です」という事実と「X が自然数なら X の次の数も自然数です」というルール（規則）を Prolog に教えます。 X の次の数を関数 s(X) で表すことにするとこのルールは再帰的な定義を使って

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

で表現できます。これを Prolog に教えて、質問してみましょう。高等学校までは自然数は 1 から始まるのが常識ですが、大学以上では 0 から始まるのが常識です。0 から始めたほうが以下に見るように色々便利だからです。


```
| ?- natural_number(0).

yes
| ?- natural_number(s(s(0))).

yes
| ?- natural_number(s(s(s(s(s(0)))))).

yes
```

となります。

```
| ?- natural_number(X).
```

と、自然数は何かと質問してください。

これはまず natural_number で始まるヘッドを探します。まず、

```
natural_number(0).
```

を見つけ、

```
X = 0 ?
```

と答えます。他にないですかと質問すると

```
natural_number(s(X)) :- natural_number(X).
```

を見つけ、 natural_number(X) と natural_number(s(X)) をパターンマッチさせます。この時、それぞれの X は重ならないように Prolog の内部では別の名前になっています。今仮に

```
natural_number(s(Y)) :- natural_number(Y).
```

となっているとします。natural_number(X) と natural_number(s(Y)) を一致させるためには X を s(Y) とすればいいです。そこで $X = s(Y)$ として natural_number(Y) を新たなゴールとして探索をします。新たな探索ですからまた最初から探します。

```
natural_number(0).
```

を見つけ、$Y = 0$ とすればよいことに気づきます。$X = s(Y)$ だったので、 $X = s(0)$ と答えます。更に、別解を聞かれると natural_number(Y) と natural_number(0) をパターンマッチさせたところだったので次の

```
natural_number(s(X)) :- natural_number(X).
```

を見つけます。natural_number(Y) と natural_number(s(X)) を一致させますがnatural_number(s(X)) は内部的には

```
natural_number(s(Z)) :- natural_number(Z).
```

であると考えると、Y = s(Z) とすれば一致します。そこでゴールを natural_number(Z) にして探索します。

```
natural_number(0).
```

を見つけ、Z = 0 とすれば、natural_number(Z) と natural_number(0) は一致します。Y = s(Z) だったので、Y = s(0) となり、更に、 X = s(Y) だったので X=s(s(0)) となり、 X=s(s(0)) と表示します。以下、これの繰り返しです。これが再帰的な定義のすごいところで、簡潔に沢山のものを表現できます。これから出てくる全部のプログラミング言語で再帰的な定義が出てきますから、そのうち慣れます。ここで、すべての自然数は何ですかとは聞かないように。自然数は無限個あります。メモリーを使い切って停止します。

Prolog に自然数の足し算を教えてみましょう。

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
plus(0, X, X) :- natural_number(X).
plus(s(X), Y, s(Z)) :- plus(X, Y, Z).
```

というプログラムを作るか、

```
| ?- assertz((plus(0, X, X) :- natural_number(X))).
| ?- assertz((plus(s(X), Y, s(Z)) :- plus(X, Y, Z))).
```

とします。

これは、「X が自然数の時、0 たす X は X です」と「X たす Y が Z の時、X の次の自然数（つまり s(X)) たす Y は Z の次の自然数（つまり s(Z)) です」という二つのルールを教えています。二番目の規則が再び再帰的な定義です。これで Prolog は自然数の足し算を理解します。確かめてみましょう。

いっぱい打ち間違えていますが、別にコンピュータが故障するわけではありません。あらゆる失敗を経験して人はプログラミングが出来るようになります。出来るようになっても、人は間違いをするものです。どうしてコンピュータが使えるようになったかと聞かれて、あらゆる失敗を経験したから使えるようになった。だから人には教えたくないと言っている人がいましたが、最近の学生さんを見ていると分かるような気がします。コンピュータはこれは本当はこうだろうという判断は基本的にしません。ソフトが勝手に直すと迷惑することもあります。例えば、ワープロソフトで学生さんの間違いを表現する文章を書いているときに、ソフトが勝手にその間違いを修正したら、意図した文章にならなく、そのソフトは使い物になりません。エラーが出たら、どこがエラーと言っているか調べて直せば良いだけです。指摘が的確でない場合もたまにありますが、エラーの指摘の近所に間違いがあります。

さて、

```
| ?- plus(0, s(0), X).

X = s(0)
```

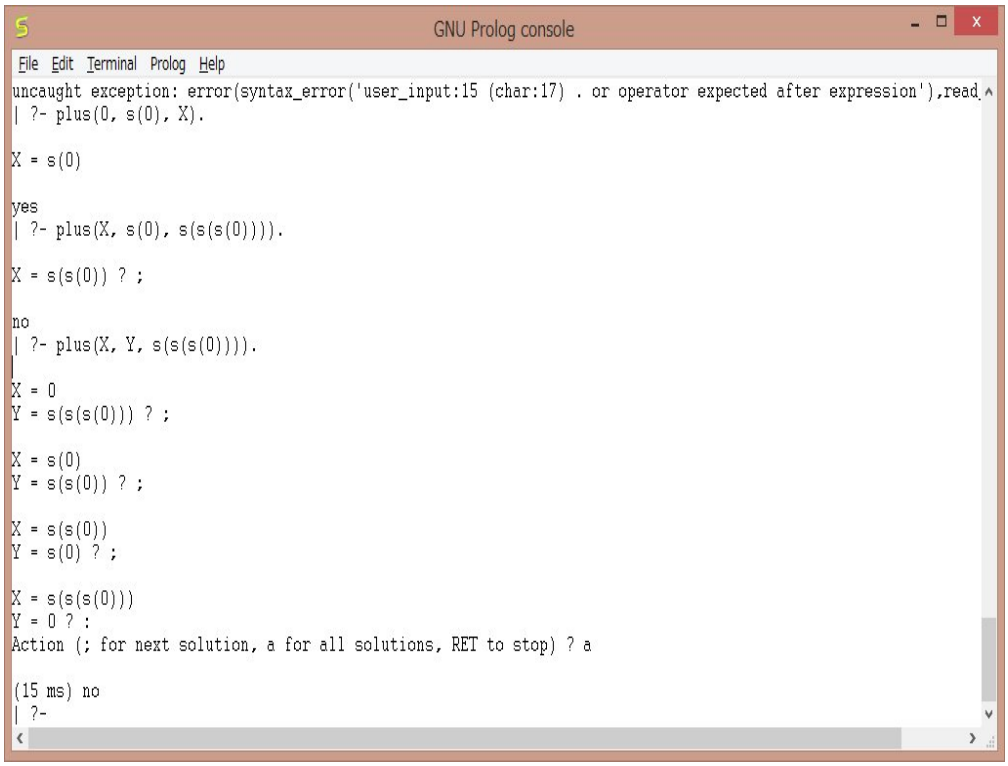
は、 0 たす 1 は何か質問したので、X = s(0)、すなわち 1 ですと答えたのですが、次の

```
| ?- plus(X, s(0),  s(s(s(0)))).

X = s(s(0))
```

は X に 1 をたして 3 になる X は何ですかと質問し、 X は 2 ですと答えています。Prolog はこのように柔軟な使い方が出来るところが面白いところです。

```
| ?- plus(X, Y, s(s(s(0)))).
```

と質問すると、たして 3 になる自然数をすべて教えてくれます。



色々自分で調べて見て下さい。

次は自然数の掛け算を Prolog に教えてみましょう。

```
times(0, X, 0) :- natural_number(X).
times(s(X), Y, Z) :- times(X, Y, XY), plus(XY, Y, Z).
```

で掛け算を教えます。「X が自然数の時、 0 掛ける X は X です」と「X の次の自然数と Y の積は、X と Y の積を XY とするとき、 XY と Y の和 Z です」の二つのルールで教えることが出来ます。変数も何文字使って表現しても良いです。では、掛け算を理解しているか調べて見ましょう。

```
| ?- times(s(s(0)), s(s(s(0))), X).
```

と質問すると

```
X = s(s(s(s(s(s(0))))))
```

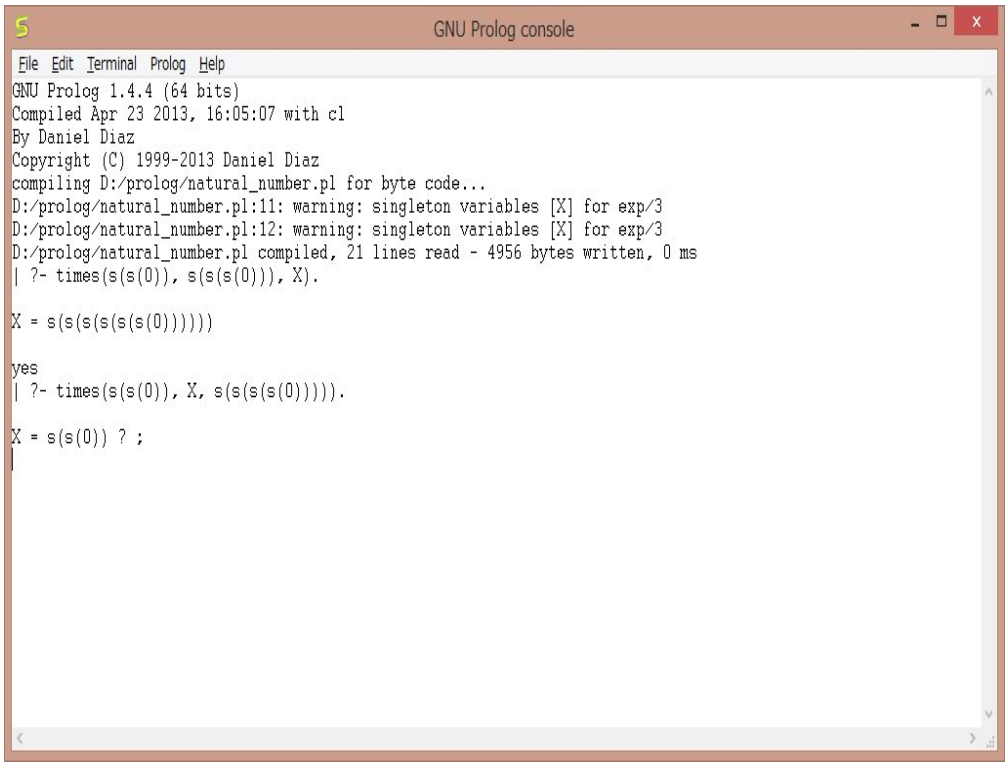
と答えますが

```
times(s(s(0)), X, s(s(s(s(0))))).
```

と質問すると

```
X = s(s(0))
```

と答えた後、フリーズします。何時でも意図した以外の使い方が出来る訳ではありません。自分で Prolog が別解を探そうとしてどうして行き詰ったか考えて見て下さい。



その他

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
plus(0, X, X) :- natural_number(X).
plus(s(X), Y, s(Z)) :- plus(X, Y, Z).
times(0, X, 0) :- natural_number(X).
times(s(X), Y, Z) :- times(X, Y, XY), plus(XY, Y, Z).
exp(s(X), 0, 0).
exp(0, s(X), s(0)).
exp(s(N), X, Y) :- exp(N, X, Z), times(Z, X, Y).
factorial(0, s(0)).
factorial(s(N), F) :- factorial(N, F1), times(s(N), F1, F).
greater_or_equal(0, X) :- natural_number(X).
greater_or_equal(s(X), s(Y)) :- greater_or_equal(X, Y).
greater(0, X) :- natural_number(X), \+(0 = X).
greater(s(X), s(Y)) :- greater(X, Y).
minimum(N1, N2, N1) :- greater_or_equal(N1, N2).
minimum(N1, N2, N2) :- greater_or_equal(N2, N1).
mod(X, Y, X) :- greater(X, Y).
mod(X, Y, Z) :- plus(X1, Y, X), mod(X1, Y, Z).
```

```
gcd(X, 0, X) :- greater(0, X).
gcd(X, Y, Gcd) :- mod(X, Y, Z), gcd(Y, Z, Gcd).
```

と定義すれば、何が出来るか自分で考えて下さい。

問題：exp, factorial, `greater_or_equal`, greater, minimum, mod, gcd は何をどのように Prolog に教えているか文章で説明しなさい。

問題：偶数（even）、奇数（odd）かを判定するルールを作れ。

数値計算

Prolog でも普通の数値計算も出来ます。階乗を計算するプログラムは次のようになります。

```
factorial(N,F) :- N>0, N1 is N-1, factorial(N1,F1), F is N*F1.
factorial(0,1).
```

次のような質問をする。

```
| ?- factorial(5,F).
```

次のような答えを得る。

```
F = 120
```

これは、階乗の一つの定義である

```
n! = n * (n-1)! if n > 0 and
n! = 1          if n = 0
```

を Prolog に教えている。このように Prolog と次の Scheme では再帰的に（数学でいう帰納的に）物事を定義する。このようなやり方に慣れることが Prolog と Scheme でプログラミング出来るかどうかの分かれ目である。C++ であれば繰り返しで定義できるが、C++ でも面白いプログラミングをするためには、再帰的定義が必要である。ここで、Prolog で数値計算をするときは

```
N1 is N-1,
F is N*F1.
```

のように、is を使う。それぞれ、N1 は N - 1 である、F は N * F1 であることを表現している。

```
| ?- N is 4 + 7.


N = 11


yes
```

ですが

```
| ?- 11 is N + 7.
uncaught exception: error(instantiation_error,(is)/2)
```

です。is の右辺は評価するとき値が確定していなければなりません。

最大公約数を求めるには、ユークリッドの互除法を使って

```
gcd(X, 0, X).
gcd(X,Y,Z):- U is X mod Y, gcd(Y, U, Z).
```

と定義すればよい。

```
| ?- gcd(24, 9, X).

X = 3 ?

(47 ms) yes
```

と答える。この場合は定義の順序が大切です。

```
gcd(X,Y,Z):- U is X mod Y, gcd(Y, U, Z).
gcd(X, 0, X).
```

と定義すると

```
| ?- gcd(24, 9, X).
uncaught exception: error(evaluation_error(zero_divisor),(is)/2)
```

とエラーになります。Prolog は gcd() の定義を定義している順に調べるので終了条件をまず定義しておく必要があります。但し、階乗を計算するプログラムのように

```
gcd(X,Y,Z):- Y > 0, U is X mod Y, gcd(Y, U, Z).
gcd(X, 0, X).
```

としておけばＯＫです。

問題：Ｎ番目の三角数を計算するプログラムを作れ。三角数とは１，３，６，１０，１５，２１，２８，３６，４５，… と続く数である。

リスト

リストは Prolog と次に説明する Scheme のプログラミングできわめて重要なもので、可変長の入れ物です。人工知能のための言語の専売特許でしたが、最近の言語、例えば、Python でも普通に使えますし、C や C++ では連結リストとしてプログラミングします。 Prolog では、

`[], [1], [1, 2, 3], [A, B, C] , [a, [b, c], d]`

のように指定する。いくつか質問してみよう。

```
| ?- [1, 2, 3] = [1, 2, 3].

yes

| ?- [1, 2, 3] = [X, Y, Z].

X = 1
Y = 2
Z = 3

yes
```

```
| ?- [1, 2, 3] = [X, X, Z].

no

| ?- [] = [].

yes

| ?- [a, b, c, d] = [Head|Tail].

Head = a
Tail = [b, c, d]

yes

| ?- [] = [Head|Tail].

no

| ?- [a] = [Head|Tail].

Head = a
Tail = []

yes

| ?- [a, b, c] = [a|Tail].

Tail = [b, c]

yes

| ?- [a, b, c] = [a|[Head|Tail]].

Head = b
Tail = [c]

yes

| ?- [a, b, c, d, e] = [_,_|[Head|Tail]].

Head = c
Tail = [d, e]

yes
```

```
| ?- [a, b, c, d, e] = [_,_|[Head|_]].

Head = c

yes
```

_ はワイルドカードで、何とでもユニファイする。
　座標がリストで与えられた多角形の符号付き面積を求めるプログラムは次のようになります。

```
area([Tuple],0).
area([(X1,Y1),(X2,Y2)|XYs],Area) :-
   area([(X2,Y2)|XYs],Area1),Area is (X1*Y2-Y1*X2)/2 + Area1.
```

次のような質問をする。

```
| ?- area([(4,6),(4,2),(0,8),(4,6)],Area).
```

次のような答えを得る。

```
Area = -8.0 ?
```

座標 (4,6),(4,2),(0,8),(4,6) で与えられる多角形の符号付き面積を計算している。
　問題 : 多角形の符号付き面積が上のプログラムで与えられる理由を説明せよ。ヒント : ベクトルのベクトル積（ベクトルの外積）を使えばよい。
　リストの最大値を求めるを求めるプログラム
　最初から複雑なプログラムを書ける人はいません。例題のプログラムを沢山読んで、何に気付いたからこのプログラムを書くことが出来たか、その目玉になった考えが何かを考えます。

```
maxlist([X|Xs],M) :- maxlist(Xs,X,M).
maxlist([X|Xs],Y,M) :- maximum(X,Y,Y1),maxlist(Xs,Y1,M).
maxlist([],M,M).
maximum(X,Y,Y) :- X =< Y.
maximum(X,Y,X) :- X > Y.
```

次のような質問をする。

```
| ?- maxlist([5,3,9,4,7,1],M).
```

次のような答えを得る。

```
M = 9 ?
```

```
maxlist([X|Xs],M) :- maxlist(Xs,X,M).
maxlist([X|Xs],Y,M) :- maximum(X,Y,Y1),maxlist(Xs,Y1,M).
maxlist([],M,M).
maximum(X,Y,Y) :- X =< Y.
maximum(X,Y,X) :- X > Y.
```

では、リストの最大値を求める 2 変数の述語 (ルール、規則)

```
maxlist(X,M)
```

を求めるのに 3 変数の述語 (ルール、規則)

```
maxlist(X, Y, M)
```

を導入し、

```
maxlist([X|Xs],M) :- maxlist(Xs,X,M).
```

で、X が最初の要素であるリスト [X—Xs] の最大値 M は、仮に現時点の最大値を X とし、X とリストの残り Xs の最大値 M を求めればよいと問題を書き換える。そして、

```
maximum(X,Y,Y) :- X =< Y.
maximum(X,Y,X) :- X > Y.
```

で、二つの数の大きい方を与える述語 (ルール、規則) を導入し (X と Y で Y が大きいのは Y が X 以上の時であり、X と Y で X が大きいのは、X が Y より大きいときである)

```
maxlist([X|Xs],Y,M) :- maximum(X,Y,Y1),maxlist(Xs,Y1,M).
```

で、Y と X を最初の要素とする [X—Xs] の最大値が M であるとは、X と Y の大きい方を Y1 とするとき、リストの残り Xs と Y1 の最大値が M であると定義できると Prolog に教えている。そして、リストが空になったとき、

```
maxlist([],M,M).
```

と仮の最大値 M が元々の最大値になると再帰的に定義している。

問題：リストの数値の和を求めるプログラムを作れ。

数独を解くプログラム

Bruce A. Tate 著　7 つの言語　7 つの世界、　Ohmsha　平成２３年　に数独が解けると書いてあります。本当かどうか確かめてみましょう。

数独のルールを記述します。

```
valid([]).
valid([Head|Tail]):-
    fd_all_different(Head), valid(Tail).
sudoku(Puzzle, Solution) :-
    Solution = Puzzle,
    Puzzle = [S11, S12, S13, S14, S15, S16, S17, S18, S19,
              S21, S22, S23, S24, S25, S26, S27, S28, S29,
              S31, S32, S33, S34, S35, S36, S37, S38, S39,
              S41, S42, S43, S44, S45, S46, S47, S48, S49,
              S51, S52, S53, S54, S55, S56, S57, S58, S59,
              S61, S62, S63, S64, S65, S66, S67, S68, S69,
              S71, S72, S73, S74, S75, S76, S77, S78, S79,
              S81, S82, S83, S84, S85, S86, S87, S88, S89,
```

```
              S91, S92, S93, S94, S95, S96, S97, S98, S99],
    fd_domain(Solution, 1, 9),
    Row1 = [S11, S12, S13, S14, S15, S16, S17, S18, S19],
    Row2 = [S21, S22, S23, S24, S25, S26, S27, S28, S29],
    Row3 = [S31, S32, S33, S34, S35, S36, S37, S38, S39],
    Row4 = [S41, S42, S43, S44, S45, S46, S47, S48, S49],
    Row5 = [S51, S52, S53, S54, S55, S56, S57, S58, S59],
    Row6 = [S61, S62, S63, S64, S65, S66, S67, S68, S69],
    Row7 = [S71, S72, S73, S74, S75, S76, S77, S78, S79],
    Row8 = [S81, S82, S83, S84, S85, S86, S87, S88, S89],
    Row9 = [S91, S92, S93, S94, S95, S96, S97, S98, S99],
    Col1 = [S11, S21, S31, S41, S51, S61, S71, S81, S91],
    Col2 = [S12, S22, S32, S42, S52, S62, S72, S82, S92],
    Col3 = [S13, S23, S33, S43, S53, S63, S73, S83, S93],
    Col4 = [S14, S24, S34, S44, S54, S64, S74, S84, S94],
    Col5 = [S15, S25, S35, S45, S55, S65, S75, S85, S95],
    Col6 = [S16, S26, S36, S46, S56, S66, S76, S86, S96],
    Col7 = [S17, S27, S37, S47, S57, S67, S77, S87, S97],
    Col8 = [S18, S28, S38, S48, S58, S68, S78, S88, S98],
    Col9 = [S19, S29, S39, S49, S59, S69, S79, S89, S99],
    Square1 = [S11, S12, S13, S21, S22, S23, S31, S32, S33],
    Square2 = [S14, S15, S16, S24, S25, S26, S34, S35, S36],
    Square3 = [S17, S18, S19, S27, S28, S29, S37, S38, S39],
    Square4 = [S41, S42, S43, S51, S52, S53, S61, S62, S63],
    Square5 = [S44, S45, S46, S54, S55, S56, S64, S65, S66],
    Square6 = [S47, S48, S49, S57, S58, S59, S67, S68, S69],
    Square7 = [S71, S72, S73, S81, S82, S83, S91, S92, S93],
    Square8 = [S74, S75, S76, S84, S85, S86, S94, S95, S96],
    Square9 = [S77, S78, S79, S87, S88, S89, S97, S98, S99],
    valid([Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9,
           Col1, Col2, Col3, Col4, Col5, Col6, Col7, Col8, Col9,
           Square1, Square2, Square3, Square4, Square5,
           Square6, Square7, Square8, Square9]).
s([_,_,_,7,1,2,4,6,_,
   _,_,_,_,3,_,1,_,8,
   _,_,_,_,_,_,_,_,7,3,
   4,_,_,3,_,_,_,_,5,
   6,5,_,_,_,_,_,_,1,7,
   8,_,_,_,_,6,_,_,2,
   2,9,_,_,_,_,_,_,_,_,
   5,_,1,_,9,_,_,_,_,
   _,3,8,1,2,4,_,_,_]).
p([_,1,8,5,_,_,_,_,_,
```

```
   _,_,_,9,_,_,8,7,_,
   _,_,4,_,_,2,_,_,_,
   6,_,_,8,_,_,7,_,_,
   _,_,_,_,7,_,_,_,_,
   _,_,2,_,_,5,_,_,4,
   _,_,_,4,_,_,1,_,_,
   _,4,9,_,_,1,_,_,_,
   _,_,_,_,_,_,7,6,4,_]).
q([_,_,3,7,_,8,_,_,_,
   _,_,_,_,_,_,_,_,_,
   6,_,_,3,1,_,_,_,_,
   9,_,8,_,_,2,_,_,5,
   _,_,4,_,_,_,3,_,_,
   2,_,_,1,_,_,4,_,9,
   _,_,_,_,2,9,_,_,_,
   _,_,_,_,_,_,_,_,_,
   _,_,_,8,_,4,7,_,_]).
```

次のような質問をする。

```
| ?- s(P),sudoku(P,S).
```

次のような答えを得る。

```
P = [3,8,5,7,1,2,4,6,9,
     9,7,6,4,3,5,1,2,8,
     1,4,2,9,6,8,5,7,3,
     4,2,7,3,8,1,6,9,5,
     6,5,3,2,4,9,8,1,7,
     8,1,9,5,7,6,3,4,2,
     2,9,4,6,5,3,7,8,1,
     5,6,1,8,9,7,2,3,4,
     7,3,8,1,2,4,9,5,6]
S = [3,8,5,7,1,2,4,6,9,
     9,7,6,4,3,5,1,2,8,
     1,4,2,9,6,8,5,7,3,
     4,2,7,3,8,1,6,9,5,
     6,5,3,2,4,9,8,1,7,
     8,1,9,5,7,6,3,4,2,
     2,9,4,6,5,3,7,8,1,
     5,6,1,8,9,7,2,3,4,
     7,3,8,1,2,4,9,5,6]

(16 ms) yes
```

これは正しい答えです。次のような質問をする。

```
| ?- p(P),sudoku(P,S).
```

次のような答えを得る。

```
P = [_#3(2..3:7:9),1,8,5,_#63(3..4:6),_#84(3..4:6),_#105(2..4:9),
       _#126(2..3:6:9),_#147(2..3:6:9),_#168(2..3:5),_#189(2..3:5..6),
       _#210(3:5..6),9,_#244(1:3..4:6),_#265(3..4:6),8,7,_#312(1..3:5..6),
       _#333(3:5:7:9),_#354(3:5..7:9),4,_#388(1:3:6..7),_#409(1:3:6:8),2,
       _#443(3:5:9),_#464(1:3:5..6:9),_#485(1:3:5..6:9),6,_#519(3:5:9),
       _#540(1:3:5),8,_#574(1..4:9),_#595(3..4:9),7,_#629(1..3:5:9),
       _#650(1..3:5:9),_#671(1:3..5:8..9),_#692(3:5:8..9),_#713(1:3:5),
       _#734(1..3:6),7,_#768(3..4:6:9),_#789(2..3:5:9),_#810(1..3:5..6:8..9),
       _#831(1..3:5..6:8..9),_#852(1:3:7..9),_#873(3:7..9),2,_#907(1:3:6),
       _#928(1:3:6:9),5,_#962(3:9),_#983(1:3:6:8..9),4,_#1017(2..3:5:7..8),
       _#1038(2..3:5..8),_#1059(3:5..7),4,_#1093(2..3:5..6:8..9),
       _#1114(3:6:8..9),1,_#1148(2..3:5:8..9),_#1169(2..3:5:7..9),
       _#1190(2..3:5:7..8),4,9,_#1237(2..3:6),_#1258(2..3:5..6:8),1,
       _#1292(2..3:5),_#1313(2..3:5:8),_#1334(2..3:5:7..8),_#1355(1..3:5:8),
       _#1376(2..3:5:8),_#1397(1:3:5),_#1418(2..3),_#1439(2..3:5:8..9),
       7,6,4,_#1499(2..3:5:8..9)]
S = [_#3(2..3:7:9),1,8,5,_#63(3..4:6),_#84(3..4:6),_#105(2..4:9),
       _#126(2..3:6:9),_#147(2..3:6:9),_#168(2..3:5),_#189(2..3:5..6),
       _#210(3:5..6),9,_#244(1:3..4:6),_#265(3..4:6),8,7,_#312(1..3:5..6),
       _#333(3:5:7:9),_#354(3:5..7:9),4,_#388(1:3:6..7),_#409(1:3:6:8),2,
       _#443(3:5:9),_#464(1:3:5..6:9),_#485(1:3:5..6:9),6,_#519(3:5:9),
       _#540(1:3:5),8,_#574(1..4:9),_#595(3..4:9),7,_#629(1..3:5:9),
       _#650(1..3:5:9),_#671(1:3..5:8..9),_#692(3:5:8..9),_#713(1:3:5),
       _#734(1..3:6),7,_#768(3..4:6:9),_#789(2..3:5:9),_#810(1..3:5..6:8..9),
       _#831(1..3:5..6:8..9),_#852(1:3:7..9),_#873(3:7..9),2,_#907(1:3:6),
       _#928(1:3:6:9),5,_#962(3:9),_#983(1:3:6:8..9),4,_#1017(2..3:5:7..8),
       _#1038(2..3:5..8),_#1059(3:5..7),4,_#1093(2..3:5..6:8..9),
       _#1114(3:6:8..9),1,_#1148(2..3:5:8..9),_#1169(2..3:5:7..9),
       _#1190(2..3:5:7..8),4,9,_#1237(2..3:6),_#1258(2..3:5..6:8),1,
       _#1292(2..3:5),_#1313(2..3:5:8),_#1334(2..3:5:7..8),_#1355(1..3:5:8),
       _#1376(2..3:5:8),_#1397(1:3:5),_#1418(2..3),_#1439(2..3:5:8..9),
       7,6,4,_#1499(2..3:5:8..9)]

(16 ms) yes
| ?-
```

これは部分解です。例えば、

```
S = [_#3(2..3:7:9),1,8,5,_#63(3..4:6),_#84(3..4:6),_#105(2..4:9),_#126(2..3:6:9),
```

の 1 番目は 2 か 3 か 7 か 9 かのいずれかであると答えています。

予想通り、難しい問題は解けません。と言うか、Bruce A. Tate 著　７つの言語　７つの世界、Ohmsha　平成２３年　に数独が解けると書いてあって、実際、単純なバックトラッキングで、簡単な問題が解けたことが信じられなかったです。GNU Prolog 固有の述語　`fd_domain()` のおかげです。

```
valid([]).
valid([Head|Tail]):-
    fd_all_different(Head), valid(Tail).
sudoku(Puzzle, Solution) :-
    Solution = Puzzle,
    Puzzle = [S11, S12, S13, S14, S15, S16, S17, S18, S19,
              S21, S22, S23, S24, S25, S26, S27, S28, S29,
              S31, S32, S33, S34, S35, S36, S37, S38, S39,
              S41, S42, S43, S44, S45, S46, S47, S48, S49,
              S51, S52, S53, S54, S55, S56, S57, S58, S59,
              S61, S62, S63, S64, S65, S66, S67, S68, S69,
              S71, S72, S73, S74, S75, S76, S77, S78, S79,
              S81, S82, S83, S84, S85, S86, S87, S88, S89,
              S91, S92, S93, S94, S95, S96, S97, S98, S99],
    fd_domain(Solution, 1, 9),
    Row1 = [S11, S12, S13, S14, S15, S16, S17, S18, S19],
    Row2 = [S21, S22, S23, S24, S25, S26, S27, S28, S29],
    Row3 = [S31, S32, S33, S34, S35, S36, S37, S38, S39],
    Row4 = [S41, S42, S43, S44, S45, S46, S47, S48, S49],
    Row5 = [S51, S52, S53, S54, S55, S56, S57, S58, S59],
    Row6 = [S61, S62, S63, S64, S65, S66, S67, S68, S69],
    Row7 = [S71, S72, S73, S74, S75, S76, S77, S78, S79],
    Row8 = [S81, S82, S83, S84, S85, S86, S87, S88, S89],
    Row9 = [S91, S92, S93, S94, S95, S96, S97, S98, S99],
    Col1 = [S11, S21, S31, S41, S51, S61, S71, S81, S91],
    Col2 = [S12, S22, S32, S42, S52, S62, S72, S82, S92],
    Col3 = [S13, S23, S33, S43, S53, S63, S73, S83, S93],
    Col4 = [S14, S24, S34, S44, S54, S64, S74, S84, S94],
    Col5 = [S15, S25, S35, S45, S55, S65, S75, S85, S95],
    Col6 = [S16, S26, S36, S46, S56, S66, S76, S86, S96],
    Col7 = [S17, S27, S37, S47, S57, S67, S77, S87, S97],
    Col8 = [S18, S28, S38, S48, S58, S68, S78, S88, S98],
    Col9 = [S19, S29, S39, S49, S59, S69, S79, S89, S99],
    Square1 = [S11, S12, S13, S21, S22, S23, S31, S32, S33],
    Square2 = [S14, S15, S16, S24, S25, S26, S34, S35, S36],
    Square3 = [S17, S18, S19, S27, S28, S29, S37, S38, S39],
    Square4 = [S41, S42, S43, S51, S52, S53, S61, S62, S63],
    Square5 = [S44, S45, S46, S54, S55, S56, S64, S65, S66],
```

```
    Square6 = [S47, S48, S49, S57, S58, S59, S67, S68, S69],
    Square7 = [S71, S72, S73, S81, S82, S83, S91, S92, S93],
    Square8 = [S74, S75, S76, S84, S85, S86, S94, S95, S96],
    Square9 = [S77, S78, S79, S87, S88, S89, S97, S98, S99],
    valid([Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9,
           Col1, Col2, Col3, Col4, Col5, Col6, Col7, Col8, Col9,
           Square1, Square2, Square3, Square4, Square5,
           Square6, Square7, Square8, Square9]).
s([_,_,_,7,1,2,4,6,_,
   _,_,_,_,3,_,1,_,8,
   _,_,_,_,_,_,_,_,7,3,
   4,_,_,3,_,_,_,_,5,
   6,5,_,_,_,_,_,_,1,7,
   8,_,_,_,_,_,6,_,_,2,
   2,9,_,_,_,_,_,_,_,_,
   5,_,1,_,9,_,_,_,_,
   _,3,8,1,2,4,_,_,_]).
p([_,1,8,5,_,_,_,_,_,
   _,_,_,9,_,_,8,7,_,
   _,_,4,_,_,2,_,_,_,
   6,_,_,8,_,_,7,_,_,
   _,_,_,_,7,_,_,_,_,
   _,_,2,_,_,5,_,_,4,
   _,_,_,4,_,_,1,_,_,
   _,4,9,_,_,1,_,_,_,
   _,_,_,_,_,_,7,6,4,_]).
q([_,_,3,7,_,8,_,_,_,
   _,_,_,_,_,_,_,_,_,_,
   6,_,_,3,1,_,_,_,_,
   9,_,8,_,_,2,_,_,5,
   _,_,4,_,_,_,3,_,_,
   2,_,_,1,_,_,4,_,9,
   _,_,_,_,2,9,_,_,_,
   _,_,_,_,_,_,_,_,_,_,
   _,_,_,8,_,4,7,_,_]).
```

の

```
valid([]).
valid([Head|Tail]):-
    fd_all_different(Head), valid(Tail).
```

は、

```
valid([]).
```

で、空のリストは正当なリストであり、

```
valid([Head|Tail]):-
    fd_all_different(Head), valid(Tail).
```

は、Head が最初のリストで、残りが Tail であるリストのリスト [Head|Tail] が正当なリストであるとは、

```
    fd_all_different(Head)
```

であること、つまり、リスト Head がすべて異なる数字からなっており（fd_all_different() は GNU Prolog 固有の述語）、更に、

```
                              , valid(Tail).
```

であること、つまり、残りのリストのリスト Tail が正当なリストであることであると定義している。

```
sudoku(Puzzle, Solution) :-
    Solution = Puzzle,
    Puzzle = [S11, S12, S13, S14, S15, S16, S17, S18, S19,
              S21, S22, S23, S24, S25, S26, S27, S28, S29,
              S31, S32, S33, S34, S35, S36, S37, S38, S39,
              S41, S42, S43, S44, S45, S46, S47, S48, S49,
              S51, S52, S53, S54, S55, S56, S57, S58, S59,
              S61, S62, S63, S64, S65, S66, S67, S68, S69,
              S71, S72, S73, S74, S75, S76, S77, S78, S79,
              S81, S82, S83, S84, S85, S86, S87, S88, S89,
              S91, S92, S93, S94, S95, S96, S97, S98, S99],
    fd_domain(Solution, 1, 9),
    Row1 = [S11, S12, S13, S14, S15, S16, S17, S18, S19],
    Row2 = [S21, S22, S23, S24, S25, S26, S27, S28, S29],
    Row3 = [S31, S32, S33, S34, S35, S36, S37, S38, S39],
    Row4 = [S41, S42, S43, S44, S45, S46, S47, S48, S49],
    Row5 = [S51, S52, S53, S54, S55, S56, S57, S58, S59],
    Row6 = [S61, S62, S63, S64, S65, S66, S67, S68, S69],
    Row7 = [S71, S72, S73, S74, S75, S76, S77, S78, S79],
    Row8 = [S81, S82, S83, S84, S85, S86, S87, S88, S89],
    Row9 = [S91, S92, S93, S94, S95, S96, S97, S98, S99],
    Col1 = [S11, S21, S31, S41, S51, S61, S71, S81, S91],
    Col2 = [S12, S22, S32, S42, S52, S62, S72, S82, S92],
    Col3 = [S13, S23, S33, S43, S53, S63, S73, S83, S93],
    Col4 = [S14, S24, S34, S44, S54, S64, S74, S84, S94],
    Col5 = [S15, S25, S35, S45, S55, S65, S75, S85, S95],
    Col6 = [S16, S26, S36, S46, S56, S66, S76, S86, S96],
    Col7 = [S17, S27, S37, S47, S57, S67, S77, S87, S97],
```

```
    Col8 = [S18, S28, S38, S48, S58, S68, S78, S88, S98],
    Col9 = [S19, S29, S39, S49, S59, S69, S79, S89, S99],
    Square1 = [S11, S12, S13, S21, S22, S23, S31, S32, S33],
    Square2 = [S14, S15, S16, S24, S25, S26, S34, S35, S36],
    Square3 = [S17, S18, S19, S27, S28, S29, S37, S38, S39],
    Square4 = [S41, S42, S43, S51, S52, S53, S61, S62, S63],
    Square5 = [S44, S45, S46, S54, S55, S56, S64, S65, S66],
    Square6 = [S47, S48, S49, S57, S58, S59, S67, S68, S69],
    Square7 = [S71, S72, S73, S81, S82, S83, S91, S92, S93],
    Square8 = [S74, S75, S76, S84, S85, S86, S94, S95, S96],
    Square9 = [S77, S78, S79, S87, S88, S89, S97, S98, S99],
    valid([Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9,
           Col1, Col2, Col3, Col4, Col5, Col6, Col7, Col8, Col9,
           Square1, Square2, Square3, Square4, Square5,
           Square6, Square7, Square8, Square9]).
```

は、数独の問題 Puzzle の解が Solution であるとは、

```
    Solution = Puzzle,
```

と、Solution と Puzzle は同じリストで、

```
    Puzzle = [S11, S12, S13, S14, S15, S16, S17, S18, S19,
              S21, S22, S23, S24, S25, S26, S27, S28, S29,
              S31, S32, S33, S34, S35, S36, S37, S38, S39,
              S41, S42, S43, S44, S45, S46, S47, S48, S49,
              S51, S52, S53, S54, S55, S56, S57, S58, S59,
              S61, S62, S63, S64, S65, S66, S67, S68, S69,
              S71, S72, S73, S74, S75, S76, S77, S78, S79,
              S81, S82, S83, S84, S85, S86, S87, S88, S89,
              S91, S92, S93, S94, S95, S96, S97, S98, S99],
```

で、Puzzle は S11, S12, S13, ... , S99 を要素とするリストで、

```
    fd_domain(Solution, 1, 9),
```

で、リスト Solution は 1 から 9 までの数字からなり（`fd_domain()` は GNU Prolog 固有の述語）、

```
    Row1 = [S11, S12, S13, S14, S15, S16, S17, S18, S19],
    Row2 = [S21, S22, S23, S24, S25, S26, S27, S28, S29],
    Row3 = [S31, S32, S33, S34, S35, S36, S37, S38, S39],
    Row4 = [S41, S42, S43, S44, S45, S46, S47, S48, S49],
    Row5 = [S51, S52, S53, S54, S55, S56, S57, S58, S59],
    Row6 = [S61, S62, S63, S64, S65, S66, S67, S68, S69],
    Row7 = [S71, S72, S73, S74, S75, S76, S77, S78, S79],
    Row8 = [S81, S82, S83, S84, S85, S86, S87, S88, S89],
    Row9 = [S91, S92, S93, S94, S95, S96, S97, S98, S99],
```

```
    Col1 = [S11, S21, S31, S41, S51, S61, S71, S81, S91],
    Col2 = [S12, S22, S32, S42, S52, S62, S72, S82, S92],
    Col3 = [S13, S23, S33, S43, S53, S63, S73, S83, S93],
    Col4 = [S14, S24, S34, S44, S54, S64, S74, S84, S94],
    Col5 = [S15, S25, S35, S45, S55, S65, S75, S85, S95],
    Col6 = [S16, S26, S36, S46, S56, S66, S76, S86, S96],
    Col7 = [S17, S27, S37, S47, S57, S67, S77, S87, S97],
    Col8 = [S18, S28, S38, S48, S58, S68, S78, S88, S98],
    Col9 = [S19, S29, S39, S49, S59, S69, S79, S89, S99],
    Square1 = [S11, S12, S13, S21, S22, S23, S31, S32, S33],
    Square2 = [S14, S15, S16, S24, S25, S26, S34, S35, S36],
    Square3 = [S17, S18, S19, S27, S28, S29, S37, S38, S39],
    Square4 = [S41, S42, S43, S51, S52, S53, S61, S62, S63],
    Square5 = [S44, S45, S46, S54, S55, S56, S64, S65, S66],
    Square6 = [S47, S48, S49, S57, S58, S59, S67, S68, S69],
    Square7 = [S71, S72, S73, S81, S82, S83, S91, S92, S93],
    Square8 = [S74, S75, S76, S84, S85, S86, S94, S95, S96],
    Square9 = [S77, S78, S79, S87, S88, S89, S97, S98, S99],
```

は、それぞれ、各列、各行、および、各ブロックがどれらの数字達で出来ているかを示し、

```
    valid([Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9,
           Col1, Col2, Col3, Col4, Col5, Col6, Col7, Col8, Col9,
           Square1, Square2, Square3, Square4, Square5,
           Square6, Square7, Square8, Square9]).
```

で、それぞれ、各列、各行、および、各ブロックがすべて異なる数字からなることをチェックしている。

```
s([_,_,_,7,1,2,4,6,_,
   _,_,_,_,3,_,1,_,8,
   _,_,_,_,_,_,_,_,7,3,
   4,_,_,3,_,_,_,_,5,
   6,5,_,_,_,_,_,_,1,7,
   8,_,_,_,_,6,_,_,2,
   2,9,_,_,_,_,_,_,_,_,
   5,_,1,_,9,_,_,_,_,
   _,3,8,1,2,4,_,_,_]).
p([_,1,8,5,_,_,_,_,_,_,
   _,_,_,9,_,_,8,7,_,
   _,_,4,_,_,2,_,_,_,
   6,_,_,8,_,_,7,_,_,
   _,_,_,_,7,_,_,_,_,
   _,_,2,_,_,5,_,_,4,
   _,_,_,4,_,_,1,_,_,
```

```
   _,4,9,_,_,1,_,_,_,
   _,_,_,_,_,_,7,6,4,_]).
q([_,_,3,7,_,8,_,_,_,
   _,_,_,_,_,_,_,_,_,
   6,_,_,3,1,_,_,_,_,
   9,_,8,_,_,2,_,_,5,
   _,_,4,_,_,_,3,_,_,
   2,_,_,1,_,_,4,_,9,
   _,_,_,_,2,9,_,_,_,
   _,_,_,_,_,_,_,_,_,
   _,_,_,8,_,4,7,_,_]).
```

は、三つの数独の問題を示していて、_ は変数の一種（無名変数）で、その値が何であるか興味がないときに使います。いちいちキーボードから打ち込むと失敗するので、定義に事実として定義しておき、

```
| ?- p(P),sudoku(P,S).
```

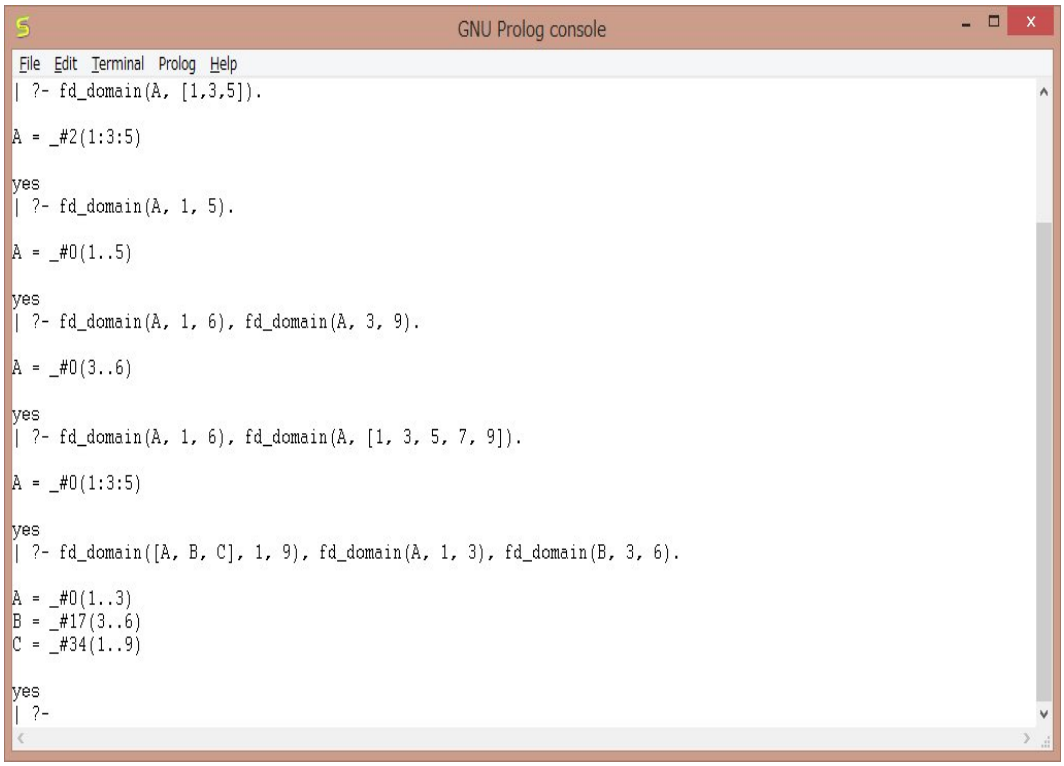
と質問することで、問題をセットしている。

fd_domain(A, B, C) という述語がどんなものか色々実験してみましょう。

```
| ?- fd_domain(A, [1, 3, 5]).

A = _#2(1:3:5)

yes
| ?- fd_domain(A, 1, 5)

A = _#0(1..5)

yes
| ?- fd_domain(A, 1, 6), fd_domain(A, 3, 9).

A = _#0(3:6)

yes
| ?- fd_domain(A, 1, 6), fd_domain(A, [1, 3, 5, 7, 9]).

A = _#0(1:3:5)

yes
| ?- fd_domain([A,B,C], 1, 9), fd_domain(A, 1, 3), fd_domain(B, 3, 6).

A = _#0(1..3)
B = _#17(3..6)
```

```
C = _#34(1..9)

yes
```

です。



fd_domain のような GNU Prolog の述語は

http://www.gprolog.org/manual/gprolog.html

の GNU PROLOG の MANUAL を見ればいいです。

member(X, L) は X がリスト L の要素であるか確かめる述語です。

```
| ?- member(3 ,[1,2,3,4,5]).

true ? ;

no
| ?- member(X, [1,2,3,4]).

X = 1 ? ;

X = 2 ? ;

X = 3 ? ;
```

```
X = 4

yes
| ?- member(1, L).

L = [1|_] ? ;

L = [_,1|_] ? ;

L = [_,_,1|_] ? ;

L = [_,_,_,1|_] ? ;

L = [_,_,_,_,1|_] ? ;

L = [_,_,_,_,_,1|_] ? ;

L = [_,_,_,_,_,_,1|_] ?

(63 ms) yes
```

のように使います。member 述語はリストのすべての要素をバックトラッキングで調べるときにも使いますが、上の数独を解くプログラムで使った範囲を与える `fd_domain(Solution, 1, 9)` 述語の代わりに member(X, [1,2,3,4,5,6,7,8,9]) を使って単純な虱潰ししようとすると膨大な計算時間がかかり、これでは数独の問題は解けません。

1, 2, 3 の順列をすべて求めるには

```
perm([A,B,C])
    :- member(A,[1,2,3]),member(B,[1,2,3]),member(C,[1,2,3]),fd_all_different([A,B,C]).
```

と定義し、

```
| ?- ['d:/prolog/permutation.pl'].
compiling d:/prolog/permutation.pl for byte code...
d:/prolog/permutation.pl compiled, 1 lines read - 1229 bytes written, 11 ms

yes
| ?- perm([A,B,C]).

A = 1
B = 2
C = 3 ? ;

A = 1
B = 3
```

```
C = 2 ? ;

A = 2
B = 1
C = 3 ? ;

A = 2
B = 3
C = 1 ? ;

A = 3
B = 1
C = 2 ? ;

A = 3
B = 2
C = 1 ? ;

(31 ms) no
```

または

```
| ?- bagof([A,B,C], perm([A,B,C]), List).

List = [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

yes
```

としても良い。

```
perm([A,B,C]) :- fd_domain([A,B,C],1,3),fd_all_different([A,B,C]).
```

と定義しても

```
| ?- ['d:/prolog/permutation.pl'].
compiling d:/prolog/permutation.pl for byte code...
d:/prolog/permutation.pl compiled, 1 lines read - 837 bytes written, 17 ms

yes
| ?- perm([A,B,C]).

A = _#3(1..3)
B = _#24(1..3)
C = _#45(1..3)

(16 ms) yes
```

となって、旨くいかない。

例題のプログラムを以下に幾つか示しておきますので、Prolog が気に入ったなら読んでみてください。解説はしません。

八人の女王の問題を解くプログラム

八人の女王の問題：チェス盤上に８個のクイーンを縦横斜めに重ならないように配置する

```
queen(Q) :- perm([1,2,3,4,5,6,7,8], Q), safe(Q).

perm([],[]).
perm(Xs, [Z | Zs]) :- select(Z, Xs, Ys), perm(Ys, Zs).

safe([Qt | Qr]) :- \+(attack(Qt, Qr)), safe(Qr).
safe([]).

attack(X, Xs) :- attack_sub(X, 1, Xs).
attack_sub(X, N, [Y|Ys]) :- (X =:= Y + N ; X =:= Y - N).
attack_sub(X, N, [Y|Ys]) :- N1 is N + 1, attack_sub(X, N1, Ys).
```

次のような質問をする。

```
| ?- queen(Q).
```

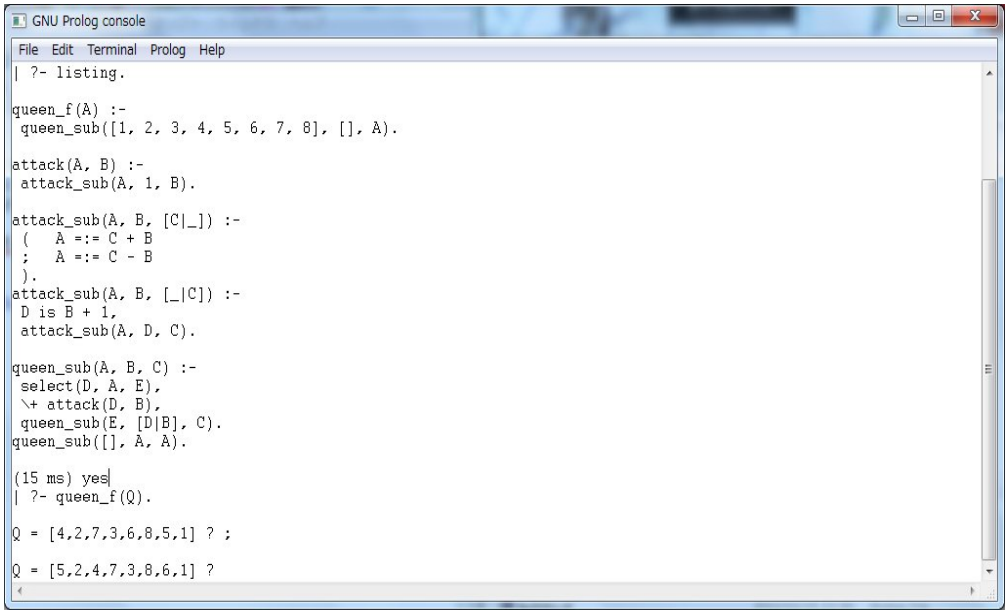
次のような答えを得る。

```
Q = [1,5,8,6,3,7,2,4] ?
```

八人の女王の問題は

```
queen_f(Q) :- queen_sub([1,2,3,4,5,6,7,8], [], Q).

queen_sub(L, SafeQs, Q) :-
        select(X, L, RestQs),
        \+(attack(X, SafeQs)),
        queen_sub(RestQs, [X | SafeQs], Q).
queen_sub([], Q, Q).

attack(X, Xs) :- attack_sub(X, 1, Xs).
attack_sub(X, N, [Y|Ys]) :- (X =:= Y + N ; X =:= Y - N).
attack_sub(X, N, [Y|Ys]) :- N1 is N + 1, attack_sub(X, N1, Ys).
```

と定義しても良い。この方が速く動く。



select 述語は

```
| ?- select(X, [1,2,3], Y).

X = 1
Y = [2,3] ? ;

X = 2
Y = [1,3] ? ;

X = 3
Y = [1,2] ? ;

(15 ms) no
```

のように使う。

ソートのプログラム

インサートソートのプログラム：ソートとは昇順または降順に並べ直すことです

```
insertionsort([X|Xs],Ys) :- insertionsort(Xs,Zs), insert(X,Zs,Ys).
insertionsort([],[]).
insert(X,[],[X]).
insert(X,[Y|Ys],[Y|Zs]) :- X > Y, insert(X,Ys,Zs).
insert(X,[Y|Ys],[X,Y|Ys]) :- X =< Y.
```

次のような質問をする。

```
| ?- insertionsort([5,8,3,1,8,4,7],X).
```

次のような答えを得る。

```
X = [1,3,4,5,7,8,8] ?
```

クイックソートのプログラム：ソートとは昇順または降順に並べ直すことです

```
quicksort([X|Xs],Ys) :-
    partition(Xs,X,Littles,Bigs),
    quicksort(Littles,Ls),
    quicksort(Bigs,Bs),
    append(Ls,[X|Bs],Ys).
quicksort([],[]).
partition([X|Xs],Y,[X|Ls],Bs) :- X =< Y, partition(Xs,Y,Ls,Bs).
partition([X|Xs],Y,Ls,[X|Bs]) :- X > Y, partition(Xs,Y,Ls,Bs).
partition([],Y,[],[]).
```

次のような質問をする。

```
| ?- quicksort([5,8,3,1,8,4,7],X).
```

次のような答えを得る。

```
X = [1,3,4,5,7,8,8] ?
```

ここで append(L1, L2, L3) は L1 と L2 を合わせたリストが L3 となることを示す述語で

```
| ?- append([1],[2,3],X).

X = [1,2,3]

yes
| ?- append([1,2],[3,4,5],X).

X = [1,2,3,4,5]

yes
| ?- append([],[1,2],X).

X = [1,2]

yes
| ?- append([1],[],X).

X = [1]
```

```
yes
| ?- append([1], X, [1,2,3]).

X = [2,3]

yes
| ?- append(X,Y,[1,2,3]).

X = []
Y = [1,2,3] ? ;

X = [1]
Y = [2,3] ? ;

X = [1,2]
Y = [3] ? ;

X = [1,2,3]
Y = []

yes
```

となります。Scheme や Visual C++ のプログラムも参照して下さい。

今では手に入りませんが、Leon Sterling, Ehud Shapiro 著 「Prolog の技芸」松田利夫訳　共立出版が Prolog の一番良い参考書です。図書館にはあります。英語版：Leon Sterling and Ehud Shapiro : The Art of Prolog second Edition, The MIT Press 1994 ならまだ手に入ると思います。

参考文献：　Problem Solving With Prolog : John Stobo (Kindle 版)
Prolog プログラミング　：　W. F. Clocksin/ C.S. Mellish 著　1983 年
Leon Sterling, Ehud Shapiro 著 「Prolog の技芸」松田利夫訳　共立出版 1988 年
Bruce A. Tate 著　７つの言語　７つの世界、　Ohmsha　平成２３年