Programation logique

PROLOG

Prolog (PROgrammer en LOGique, 1970, Robert A. Kowalski (Edinburgh) & Alain Colmerauer (Marseille)) est né du besoin de pouvoir traiter la langue naturelle par ordinateur et, en particulier, la grammaire.

Télécharger et installer SWI Prolog

Faits

Créer un nouveau fichier (tests.pl) qui va constituer votre base de faits.

Les faits : « Jean aime Marie » ou « Anne aime Jean » sont traduits en Prolog par :

```
aime(jean, marie). % car Jean aime Marie
aime (paul , marie). % Paul est amoureux de Marie
aime (marie , paul). % et Marie aime Paul
```

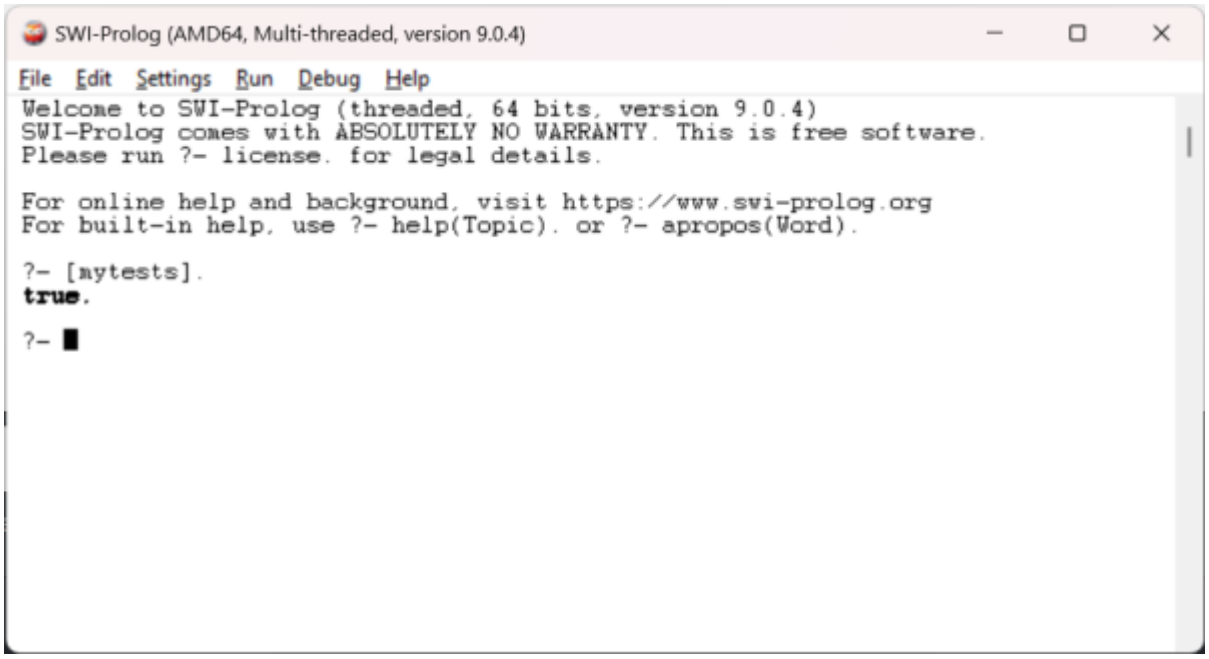
Avec

1. le nom de la relation ou prédicat (débutant par une minuscule),
2. le/les arguments (ici « jean » et « marie ») séparés par une virgule, dans un ordre qui possède un sens (qui est le sujet et le complément de l'action décrite),
3. le fait se termine par un point « . ».

Les espaces ne jouent pas de rôle et le % indique le début du commentaire. (Le commentaire peut aussi être placé entre /* … */). Le nom de la relation (ici « aime ») dépend du programmeur / analyste.

Questions

Les questions permettent d'interroger la base de fait. Elles se posent dans la fenêtre d'exécution de Prolog.



Charger votre base de connaissances avec le nom du fichier entre crochets :

```
[tests].
```

En Prolog, les questions débutent par le point d'interrogation suivi du nom de la relation et du/des argument(s) (objet). On parle également de but pour désigner une question. Par exemple, on peut se poser la question de savoir si « Est-ce que Jean aime Marie ? ». Cette question se traduit par :

```
?- aime(jean, marie).
```

Pour y répondre, l'interprète Prolog va essayer d'unifier la question posée avec un des faits de sa base de connaissance. S'il réussit, il répond « Yes » et « No » dans le cas contraire.

Variables

Une variable commence par une Majuscule.

Les variables permettent d'introduire des inconnues, et sont utilisables dans les questions :

```
?- aime(jean, X).
X = marie
Yes
```

Il est possible d'obtenir les réponses suivantes, en utilisant le `;` :

```
?- aime(jean, X).
X = marie ;
X = peche ;
No
```

Dans un fait, une variable représente tout (quelque chose de quelconque) :

`anne aime tout` s'écrira :

```
aime(anne, _) % _ est une variable quelconque
```

Conjonctions

Soit la base de connaissances suivante :

```
aime(jean, peche).
aime(jean, marie).
aime(jean, paris).
aime(marie, paul).
aime(marie, paris).
aime(paul, marie).
aime(paul, biere).
aime(anne, bretagne).
roi(tintin, belgique).
```

Il est possible de vérifier la véracité simultannée de plusieurs faits avec un ET : `,` :

```
?- aime(jean, marie) , aime(marie, jean).
No
```

« Existe-t-il quelque chose que Jean et Paul aiment ? ».

```
?-aime(jean, X), aime(paul, X)
```

Règles

Les règles permettent de généraliser les faits, pour éviter d'avoir à saisir tous les faits dans la base de connaissances.

Une règle correspond à une affirmation générale sur les objets et leurs relations.

Par exemple, on sait que « Paul aime tous ceux qui aiment la bière » que l'on écrit en Prolog comme suit :

```
aime(paul, X) :- aime(X, biere).
```

Et cette règle se compose :

- d'une tête (aime(paul, X)) ;
- du symbole « :- » pour indiquer le « SI » ;
- d'un corps (aime(X, biere)) ;

- et d'un « . » final

L'ensemble des règles possédant le même nom (foncteur) et le même nombre d'arguments (arité) doivent se suivre dans votre programme (elles doivent former un paquet de clauses).

Soient les règles suivantes :

- « Anne aime tous les rois ».
- « Arthur aime ceux qui l'aiment ».
- « Jean aime toutes les femmes ».
- « Paul aime les gens qui aiment la biere et Londres ».
- « Il faut taxer les riches ».
- « Marie aime toutes les villes ».

Elles s'écriront en Prolog :

```
aime(anne, UnRoi) :- roi(UnRoi, Pays).
aime(arthur, X) :- aime(X, arthur).
aime(jean, X) :- femme(X).
aime(paul, X) :- aime(X, biere), aime(X, londres).
taxer(UnePersonne) :- riche(UnePersonne).
aime(marie, UneVille) :- ville(UneVille).
```

Exercices

Soit la base de connaissances :

```
masculin(hubert).
masculin(denis).
masculin(robert).
masculin(joseph).
masculin(georges).
masculin(henri).
feminin(nelly).
feminin(martine).
feminin(anne).
feminin(jeanne).
% parent(X,Y) est vrai si Y est le pere/mere de X
parent(robert, hubert).
parent(robert, georges).
parent(robert, anne).
parent(joseph, nelly).
parent(hubert, denis).
parent(hubert, martine).
parent(nelly, denis).
parent(nelly, martine).
parent(georges, jeanne).
parent(georges, henri).
```

Définissez une règle pere(Pere, Enfant) qui est vrai si Pere est le père de Enfant.

Faites de même avec la relation mere(Mere, Enfant).

Définissez une règle fils(Fils, Parent) qui est vrai si Fils est le fils du parent Parent. Faites de même avec la relation fille(Fille, Parent).

Définissez les règles

- frere(Frere, X) qui est vrai si Frere est le frère de X.
- oncle(Oncle, X) qui est vrai si Oncle est l'oncle de X.
- neveu(Neveu, X) qui est vrai si Neveu est le neveu de X.
- grand-pere(GP, X) qui est vrai si GP est le grand-père de X.

Exercice

On considère les affirmations suivantes (dues à Lewis Carroll, dans La logique symbolique) :

1. Tous les canards vivant dans ce village qui sont marqués d'un B appartiennent à Mrs. Bond.
2. Les canards vivant dans ce village ne portent pas de col en dentelle, à moins qu'ils n'appartiennent à Mrs. Bond.
3. Mrs. Bond ne possède aucun canard gris vivant dans ce village.

Traduisez ces affirmations sous la forme de règles Prolog dans un fichier canards.pl en vous appuyant sur les constantes **ce_village** et **mrs_bond**, les prédicats unaires **canard** et **porte_col_dentelle**, et les prédicats binaires **appartient_a**, **marque_avec**, **pas_gris** et **vit_dans** (et aucun autre prédicat ou symbole de fonction).

À présent, on considère George, un canard de ce village marqué AB et qui porte un col de dentelles et Augusta, une cane de ce village marquée B. Ajoutez les faits correspondant à votre fichier.

On se demande quels sont les canards de la base qui ne sont pas gris. Utilisez l'interprète Prolog pour répondre à cette question.

Récursivité

Liste des nombres de N à 1

```
/* de N à 1 */
decroissant(0).
decroissant(N) :- N>0, write(N), nl, N1 is N-1, decroissant(N1).
```

Liste des nombres de 1 à N

```
/* de 1 à N */
croissant(0).
croissant(N) :- N>0, N1 is N-1, croissant(N1), write(N), nl.
```

Nombres pairs

```
pair(0).
pair(X) :- X>0, X2 is X-2, pair(X2).
```

Somme des N premiers entiers

```
som(0,0).
som(N,X) :- N>0, N1 is N-1, som(N1,X1), X is N+X1.
```

Factorielle

```
fact(0,1).
fact(N,X) :- N>0, N1 is N-1, fact(N1,X1), X is N*X1.
```

Suite de Fibonacci

```
fibo(1,1).
fibo(2,1).
fibo(N,X) :- N>2, U is N-1, V is N-2, fibo(U,U1), fibo(V,V1),
         X is U1+V1.
```

Listes

En Prolog, Les tableaux sont des listes, dans lesquelles les indices des éléments ne sont pas disponibles.

En revanche, une liste L peut toujours être décomposée en L=[E|R] où E est le premier élément de la liste (E n'est pas une liste) et où R est le reste de la liste L (R est une liste : c'est en fait la tranche de L qui démarre apres E).

La liste vide est [].

Affichage

```
affiche([]).
affiche([X|R]) :- write(X), nl, affiche(R).
```

Premier élément

```
premier([X|_],X).
```

Dernier élément

```
dernier([X],X).
dernier([_|L],X) :- dernier(L,X).
```

Compte

```
compte([],0).
compte([_|R],N) :- compte(R,N1), N is N1+1, N>0.
```

Somme

```
somme([],0).
somme([X|R],N) :- somme(R,N1), N is N1+X.
```

Sudoku

Base de connaissances :

```
sudoku(Rows) :-
        length(Rows, 9), maplist(same_length(Rows), Rows),
        append(Rows, Vs), Vs ins 1..9,
        maplist(all_distinct, Rows),
        transpose(Rows, Columns),
```

```
        maplist(all_distinct, Columns),
        Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],
        blocks(As, Bs, Cs),
        blocks(Ds, Es, Fs),
        blocks(Gs, Hs, Is).
blocks([], [], []).
blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-
        all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),
        blocks(Ns1, Ns2, Ns3).
problem(1, [[_,2,7,1,_,_,_,_,8],
            [_,_,_,_,7,_,_,4,2],
            [_,8,_,_,_,_,_,9,_],
            [1,_,_,_,_,_,6,_,_],
            [5,_,_,_,2,_,_,_,_],
            [_,_,_,_,8,5,_,_,_],
            [_,7,1,4,_,_,_,_,_],
            [_,4,_,2,6,_,_,_,_],
            [_,_,_,_,_,_,_,_,3]]).
```

Résolution

```
use_module(library(clpfd)).
[sudoku].
problem(1, Rows), sudoku(Rows), maplist(portray_clause, Rows).
```