Chapitre 5 : Introduction à la programmation logique

Carole Porrier
carole.porrier@univ-paris13.fr

Département d'informatique
IUT de Villetaneuse

30 avril 2024
Paradigmes de développement universels

Table des matières

Table des matières

Programmation logique

- Paradigme de programmation basé sur la **logique**;
- Repose sur le **calcul des prédicats**;
- Peut être vu comme un cas **particulier** de la **programmation par contraintes**;
- Le langage le plus **populaire** dans cette famille est sans doute **Prolog**;
- Très populaire dans les **années 80**;
- En revanche, une approche basée sur l'utilisation de **bibliothèques dédiées** (par exemple Aleph) est généralement favorisée par rapport à l'emploi d'un **langage** comme Prolog.

Prolog

- Créé vers **1972**, par **Alain Colmerauer** et **Philippe Roussel**, à l'Université de la Méditerranée Aix-Marseille;
- **Motivé** par des applications en
  - **intelligence artificielle**;
  - **traitement** linguistique;
  - conception de **systèmes experts**;
  - planification **automatique**;
  - systèmes à base de **règles**;
  - **preuve** de théorèmes, etc.

Références

- SWI-Prolog, le principal compilateur Prolog libre disponible.
- Programming in Prolog, par William F. Clocksin et Christopher S. Mellish, Springer. La principale ressource sur laquelle je me baserai. Un pdf est disponible en ligne.
- Prolog, a tutorial Introduction, par James Lu et Jerud J. Mead. Aussi disponible en ligne.

Hello, world!

- Un exemple minimal de **script Prolog**:

```
#!/usr/bin/swipl -s

:- initialization(main).

main :- write('Hello, world!'), nl, halt.
```

- On peut rendre le **script** exécutable:

```
chmod +x hello.pl   # Pour rendre le script exécutable
./hello.pl          # Pour l'exécuter
```

L'énigme d'Einstein

```
/*
* L'énigme d'Einstein (ou l'énigme des cinq maisons), attribuée au physicien
* Einstein s'énonce comme suit:
*
* Il y a cinq maisons de cinq couleurs différentes. Dans chacune de ces maisons
* vit une personne de nationalité différente. Chacune de ces personnes boit une
* boisson différente, fume un cigare différent et a un animal domestique
* différent.
*
*  1. L'Anglais vit dans la maison rouge.
*  2. Le Suédois a des chiens.
*  3. Le Danois boit du thé.
*  4. La maison verte est à gauche de la maison blanche.
*  5. Le propriétaire de la maison verte boit du café.
*  6. La personne qui fume des Pall Mall a des oiseaux.
*  7. Le propriétaire de la maison jaune fume des Dunhill.
*  8. La personne qui vit dans la maison du centre boit du lait.
*  9. Le Norvégien habite dans la première maison.
* 10. L'homme qui fume des Blend vit à côté de celui qui a des chats.
* 11. L'homme qui a un cheval est le voisin de celui qui fume des Dunhill.
* 12. Le propriétaire qui fume des Blue Master boit de la bière.
* 13. L'Allemand fume des Prince.
* 14. Le Norvégien vit juste à côté de la maison bleue.
* 15. L'homme qui fume des Blend a un voisin qui boit de l'eau.
*
* Question : qui a le poisson ?
*/
```

Modélisation en Prolog

```
/*
* On représente une maison par une liste de la forme
* `[couleur,nationalite,boisson,cigare,animal]`
*
* Si on entre la question `resoudre(Maisons)`, alors
* `Maisons` contient l'unique configuration satisfaisant
* toutes les contraintes.
*/
gauche(X,Y,L) :- append(_,[X,Y|_],L).

voisin(X,Y,L) :- gauche(X,Y,L) ; gauche(Y,X,L).

resoudre(Maisons) :- length(Maisons,5),
    member([rouge,anglais,_,_,_],Maisons),
    member([_,suedois,_,_,chien],Maisons),
    member([_,danois,the,_,_],Maisons),
    gauche([verte,_,_,_,_],[blanche,_,_,_,_],Maisons),
    member([verte,_,cafe,_,_],Maisons),
    member([_,_,_,pallmall,oiseaux],Maisons),
    member([jaune,_,_,dunhill,_],Maisons),
    Maisons=[_,_,[_,_,lait,_,_],_,_],
    Maisons=[[_,norvegien,_,_,_],_,_,_,_],
    voisin([_,_,_,blend,_],[_,_,_,_,chat],Maisons),
    voisin([_,_,_,_,cheval],[_,_,_,dunhill,_],Maisons),
    member([_,_,biere,bluemaster,_],Maisons),
    member([_,allemand,_,prince,_],Maisons),
    voisin([_,norvegien,_,_,_],[bleue,_,_,_,_],Maisons),
    voisin([_,_,_,blend,_],[_,_,eau,_,_],Maisons),
    member([_,_,_,_,poisson],Maisons).
```

Interrogation

```
/* On lance d'abord l'interpréteur en chargeant le fichier */
 $ swipl einstein.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
    software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

/* Puis on pose la question */
/* Les variables doivent commencer par une majuscule */
?- resoudre(Maisons).
Maisons = [[jaune, norvegien, eau, dunhill, chat], [bleue,
    danois, the, blend, cheval], [rouge, anglais, lait,
    pallmall, oiseaux], [verte, allemand, cafe, prince, poisson
    ], [blanche, suedois, biere, bluemaster|...]] ;

/* Prolog retourne la première solution */
/* Si on entre ; il tente de trouver une autre solution */
/* Dans le cas présent, il n'y a qu'une unique solution */
false.
```

Table des matières

Propositions et prédicats (1/3)

- En logique, une **proposition** est un énoncé qui est soit **vrai**, soit **faux**:
  - "Il pleut";
  - "Alice est heureuse";
  - "Bob et Carl sont des frères";
  - "Le nombre 3 est pair".
- Un **prédicat** est simplement une **fonction** booléenne:

$$\text{heureuse}(x) = \begin{cases} vrai, & \text{si } x \text{ est heureuse;} \\ faux, & \text{sinon.} \end{cases}$$

$$\text{frères}(x, y) = \begin{cases} vrai, & \text{si } x \text{ et } y \text{ sont frères;} \\ faux, & \text{sinon.} \end{cases}$$

Propositions et prédicats (2/3)

- En Prolog:

```
il_pleut :- true.   % Doit commencer par une minuscule
heureuse(alice).    % Prédicat unaire heureuse(X)
freres(bob, carl).  % Prédicat binaire freres(X,Y)
pair(2).            % Prédicat unaire pair(X)
```

- Une proposition ou un prédicat **évalué en une valeur** sont appelés **faits** (*facts*).
- Par défaut, tout fait non précisé est considéré **faux** (principe du **tiers exclus**).

Propositions et prédicats (3/3)

- On appelle `alice`, `bob` et `carl` des **atomes**;
- `2` et `3` sont des **nombres**;
- `heureuse(alice)` et `freres(bob, carl)` sont des **termes composés**, `heureuse` et `freres` sont appelés **foncteurs**.
- L'**arité** d'un terme composé est donnée par son nombre d'**arguments**:
  - `il_pleut` est d'arité 0;
  - `heureuse` est d'arité 1;
  - `freres` est d'arité 2.
- On écrit `il_pleut/0`, `heureuse/1` et `freres/2` pour indiquer l'arité.

Règles (1/2)

- En logique, on s'intéresse en particulier à l'**inférence**.
- Par exemple:
  - "Tout rectangle possède **4 angles droits**",
  - "Un **carré** est un rectangle",
  - Donc, "un carré possède 4 angles droits".
- En **Prolog**:

```
est_rectangle(carre).
a_4_angles_droits(X) :- est_rectangle(X).
% Noter que les variables commencent par une majuscule
```

- La syntaxe

```
Head :- Body
```

  permet de définir une **règle**.

Règles (2/2)

```
 $ swipl carre.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- est_rectangle(carre).
true.

?- est_rectangle(rectangle).
false.

?- a_4_angles_droits(carre).
true.

?- a_4_angles_droits(rectangle).
false.
```

Connecteurs logiques

- En logique, les **connecteurs** $\wedge$ (et) et $\vee$ (ou) sont **fondamentaux**;
- En Prolog, on utilise la **virgule** pour la **conjonction** (et) et le **point-virgule** pour la **disjonction** (ou).

```
 $ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- true, true.
true.

?- true, false.
false.

?- true; false.
true .

?- false; false.
false.
```

Exercice

Le fichier **famille.pl** contient plusieurs faits ainsi que les règles `are_siblings`, `are_half_siblings` et `has_child`.

1. **Compléter** en définissant les règles `have_same_mother` et `have_same_father`;
2. **afficher** la liste des enfants de **Marge**.