Chapitre 8 : Modules Prolog

Carole Porrier
carole.porrier@univ-paris13.fr

Département d'informatique
IUT de Villetaneuse

6 mai 2024
Paradigmes de développement universels

Déclaration et utilisation d'un module

- La **modularité** en Prolog est relativement **simple**;
- Il existe des mécanismes d'**importation** et d'**exportation** similaires à Haskell;
- On utilise la **règle**

```
:- module(NomModule, ListePredicatsExportes).
```

- Exemple

```
:- module(tp2, [solve/3]).
```

- Comme il n'y a pas de notion de **privé**/**public**, on peut obtenir des comportement semblables en **restreignant** l'exportation.

Charger une base de connaissance

- Il s'agit du prédicat `consult/1`;
- Exemple:

```
?- consult(fichier).
?- consult('/chemin/vers/fichier').
```

- Raccourci avec la notation **liste**:

```
?- [fichier1, fichier2, '../fichier3'].
```

Identification de termes

- Les prédicats **var**/1, **nonvar**/1; **atom**/1; number/1; **atomic**/1;
- Exemples:

```
?- var(X).
yes
?- var(8).
no
?- nonvar(8).
yes
?- atom(23).
no
?- number(23).
yes
?- atomic(23).
yes
```

Afficher des informations (1/2)

- Le prédicat `listing/1` permet d'afficher des informations sur un **atome donné**;
- Exemple:

```
?- consult(famille).
true.
?- listing(has_mother).
has_mother(bart, marge).
has_mother(lisa, marge).
has_mother(maggie, marge).
has_mother(marge, jackie).
has_mother(homer, mona).
has_mother(patty, jackie).
has_mother(selma, jackie).
has_mother(abraham, yuma).
has_mother(yuma, theodora).
true.
```

Afficher des informations (2/2)

- Fonctionne aussi pour afficher des **règles**:
- Exemple:

```
 $ swipl famille.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).

?- listing(are_siblings).
are_siblings(A, B) :-
        has_mother(A, C),
        has_mother(B, C),
        has_father(A, D),
        has_father(B, D).

true.
```

Assertions dynamiques

- Dans certains cas, il est utile d'**ajouter** de façon dynamique des nouveaux **faits**;
- Deux prédicats utiles: `asserta/1` et `assertz/1`;
- Les deux jouent le même rôle, mais un ajoute le fait au **début** de la base (`asserta/1`) alors que l'autre l'ajoute à la **fin** (`assertz/1`).
- Pour **annuler** une assertion, on utilise `retract/1`.
- **Attention!** Dans un programme, il faut déclarer les prédicats comme **"dynamiques"** pour utiliser des assertions.

Exemple

```
?- assert(happy(alice)).
true.
?- assertz(happy(bob)).
true.
?- listing(happy).
:- dynamic happy/1.
happy(alice).
happy(bob).
true.
?- retract(happy(alice)).
true.
?- listing(happy).
:- dynamic happy/1.
happy(bob).
true.
?- retractall(happy(_)).
true.
?- listing(happy).
:- dynamic happy/1.
true.
```

Décomposer des structures

- Dans certains cas, il peut être utile de **déconstruire** des structures pour manipuler leur **foncteur** et leurs **arguments**;
- Deux prédicats utiles: `functor/3` et `arg/3`;

```
?- functor(happy(alice), F, N).
F = happy,
N = 1.
?- functor(4 + 5, F, N).
F =  (+),
N = 2.
?- arg(2, soeurs(alice, diane), X).
X = diane.
?- arg(1, soeurs(alice, diane), X).
X = alice.
```

Manipulation d'atomes

- Il est possible de **convertir** des atomes et des nombres en **liste de caractères**;
- Deux prédicats utiles: `atom_chars/2` et `number_chars/2`;
- Exemples:

```
?- atom_chars(heureuse, X).
X = [h, e, u, r, e, u, s, e].
?- number_chars(123.5, X).
X = ['1', '2', '3', '.', '5'].
```

Entrées et sorties

- `get_char(X)`: Récupère un **caractère** sur `stdin`;
- **`read`**`(X)`: Récupère un **terme** (suivi d'un **point**);
- `put_char(X)`: Écrit un **caractère** sur `stdout`;
- **`nl`**: Écrit **\n** sur `stdout`;
- **`write`**`(X)`: Écrit un **terme** sur `stdout`.
- etc.

Fichiers

- `open(X, Y, Z)`: Ouvre le fichier `X`, en mode `Y` (soit **`read`** ou **`write`**) et alors `Z` est instantié en **flux**;
- **`close`**`(Z)`: Ferme le flux `Z`.
- Ensuite, il suffit de **rediriger** l'entrée et/ou la sortie;
- `set_input(X)`: Redirige `stdin` vers le flux `X`;
- `set_output(X)`: Redirige `stdout` vers le flux `X`.

Débogage

- Pour déboguer un programme Prolog, on peut utiliser des `write` bien placées;
- Sinon, il existe des prédicats permettant de le faire de façon **dynamique**: `trace`, `notrace`, `spy`, `debugging`, `nodebug`, `nospy`, etc.