PLC – Cours 5

Thi-Bich-Hanh Dao

Université d'Orléans

M1 Informatique

Plan

I. Prédicats d'entrée/sortie

Prédicats de base

- Prolog communique avec des flux : un flux d'entrée (clavier) et un flux de sortie (écran).
- Les prédicats de base :
  - `see(fichier).` change le flux d'entrée vers fichier
  - `see(user).` met le flux d'entrée standard
  - `tell(fichier).` change le flux de sortie vers fichier
  - `tell(user).` met le flux de sortie standard
  - `seen` ferme le fichier d'entrée courant
  - `told` ferme le fichier de sortie courant
  - `read(X).` lit le prochain terme (délimité par .) du flux d'entrée et unifie avec X
  - `write(X).` écrit le terme sur le flux de sortie

I. Prédicats d'entrée/sortie

Exemple 1

```
cube :- write('Nouveau chiffre, s.v.p. (ou stop): '),
        read(X),
        process(X).

process(stop).

process(N) :- integer(N), C is N * N * N,
        write('Le cube de '), write(N),
        write(' est: '),
        write(C), nl, cube.
```

I. Prédicats d'entrée/sortie

Exemple 2

triangle(N) imprime un triangle de N lignes chaque ligne i<N composée de i étoiles

```
triangle(N) :- dessine(0,N).

dessine(N,N).
dessine(M,N) :- M<N, L is M+1,
          ligne(L), nl, dessine(L,N).

ligne(0).
ligne(N) :- N>0, write('*'),
          M is N-1, ligne(M).
```


Plan

Prédicats du second ordre

`setof, bagof, findall, call`

- `findall(X,P,L)` produit la liste `L` de tous les termes `t` tels que le but `P[t/X]` est satisfait.
- `bagof(X,P,L)` comme `findall` mais sépare les instanciations des variables libres dans `P`.
- `setof(X,P,L)` comme bagof sauf que la liste `L` est ordonnée et les éléments en double sont éliminés.
- `call(G)` exécute le but `G`, succède si `G` est satisfait, si `G` contient une coupure cette coupure n'a pas d'effet en dehors de `G`
- Exemple : voir tableau

Plan

Manipulation de termes

L'ordre sur les termes

Les termes en Prolog sont ordonnés totalement par l'ordre suivant (du plus petit au plus grand) :

- variables ordonnées par l'"age", du plus petit au plus grand
- nombres flottants, en ordre numérique
- entiers, en ordre numérique
- atomes, en ordre lexicographique
- termes composés, ordonnés d'abord par l'arité, puis par le nom du principal foncteur, puis par les arguments de gauche à droite. Les listes sont considérées comme des termes composés (fonction '.').

Prédicats de comparaison de termes (d'arité 2)

`==` `\==` `@<` `@<=` `@>` `@>=`

Manipulation de termes

Prédicats de test

- `integer/1` teste si l'argument est un entier
- `number/1` teste si l'argument est un nombre
- `float/1` teste si l'argument est un nombre flottant
- `atomic/1` teste si l'argument est un atome
- `var/1` teste si l'argument est une variable non instanciée
- `nonvar/1` teste si l'argument n'est pas une variable non instanciée
- `compound/1` teste si l'argument est un terme composé
- `ground/1` teste si l'argument est un terme sans variable

Manipulation de termes

Prédicats de manipulation

- `functor(Terme, Foncteur, Arité)` succède si le foncteur de `Terme` est `Foncteur` et son arité est `Arité`. Mode d'utilisation $(+,-,-)$ et $(-,+,+)$ seulement

```
?- functor(f(a,g(Y,b)),F,A).
A = 2
F = f
?- functor(T,f,2).
T = f(_,_)
```

- `arg(N,Terme,X)` unifie `X` avec le `N`-ième argument de `Terme`

```
?- arg(2,f(a,g(Y,b)),T).
T = g(Y,b)
```

- univ `Terme =.. Liste` transforme un terme en une liste

```
?- f(a,X,g(Y,b)) =.. L.
L = [f,a,X,g(Y,b)]
?- T =..  [f,a,X,g(Y,b)].
T = f(a,X,g(Y,b))
```


Manipulation de termes

Exemples

- Aplatir une liste
- Chercher tous les sous-termes d'un terme

Plan

Gestion dynamique de clauses

Directive `dynamic/1`

- Une procédure est soit statique soit dynamique.
- Les prédicats pré-définis sont statiques.
- Les prédicats définis par l'utilisateur sont par défaut statiques.
- Lorsqu'on veut ajouter/supprimer des clauses dans un prédicat, il faut déclarer que le prédicat est dynamique.
- `dynamic(Nom/Arité)` précise que le prédicat nommé `Nom` d'arité `Arité` est dynamique.

Gestion dynamique de clauses

Prédicats pour modifier dynamiquement un programme logique

- `asserta(C).` ajoute le prédicat (atomique) `C` au début du programme.
- `assertz(C).` ajoute le prédicat (atomique) `C` à la fin du programme.
- `retract(C).` supprime le prédicat (atomique) `C` du programme.
- `abolish(P,N).` supprime les règles du prédicat `P` d'arité `N`.
- Exemple : programme 28-assert.pl