Séance 1 du Projet 4IF ALIA – Septembre 2014
Découverte de Prolog

J-F. Boulicaut – M. Kaytoue

Nous vous proposons des exercices pour découvrir et maîtriser certains des mécanismes assez particuliers de la programmation en Prolog. Votre effort d'appropriation des concepts du langage (explication du codage des prédicats, usage de l'unification, approche rigoureuse dans les tests avec notamment l'étude des différentes configurations d'appel) est indispensable pour s'engager dans la réalisation d'un projet de programmation sur les deux séances suivantes.
La prise en main du langage peut s'appuyer sur les exemples montrés pendant la présentation et codés pour vous dans le fichier « intro_prolog.pl ». Il est important de savoir consulter la documentation en ligne de swi-prolog pour comprendre les tâches réalisées par les prédicats fournis (Built-in Predicates).

1. Thème de la généalogie

Créer une base de connaissances contenant des personnes, divers attributs de celles-ci (par exemple le genre) et la définition de certaines relations familiales (par exemple, « parent »). On peut alors étudier le codage de nouvelles relations comme « ancêtre » ou « frère ou sœur » ou encore « oncle ou tante ». Ce domaine simple est très bien pour découvrir les mécanismes d'aide à la mise au point (« debug » avec activation des traces – trace, notrace, spy, nospy, …) et les prédicats prédéfinis indispensables (comme, par exemple, pour les E/S avec read, write, nl, etc).

2. Manipulation de listes

Modifiez « element(Obj,List) » de façon à ce qu'un troisième argument représente « List\Ob ».
Ainsi :
?-element(b,[a,b,c],R). devrait se prouver avec R=[a,c] ;
?-element(X,[a,b,c],R). devrait se prouver avec X=a et R=[b,c], etc …
?-element(a,L,[b,c]). devrait se prouver avec L=[a,b,c], puis L=[b,a,c], etc …
Tester tous les modes d'utilisation « convenables » et observer comment l'interprète Prolog procède.

A partir de « element/3 », écrire un prédicat « extract(L,Extrait) » qui doit notamment permettre d'extraire des éléments de L. Ainsi on devrait pouvoir l'utiliser comme suit :
?- extract([a,b,c,a,c],[X,Y]). devrait se prouver avec [X,Y]=[a,b], puis [a,c], etc

Ecrire la concaténation de listes (le prédicat « append » a été présenté), tester et observer les modes de fonctionnement. A partir de ce prédicat de concaténation, réaliser l'inversion d'une liste. Ainsi inv([a,b,c],X) devrait réussir avec X=[c,b,a]. Tester les divers modes d'appel et constater que certains posent des problèmes. Trouver une solution en testant à l'appel la nature des arguments (usage de « var » et « novar » qui testent si une variable est libre ou liée).

Ecrire un prédicat qui permette l'expression de substitutions dans une liste. Ainsi « subsAll(a,x,[a,b,a,c],R) » devrait réussir avec R=[x,b,x,c]. Modifier sa définition pour qu'il soit reprouvable en ne faisant chaque fois qu'une substitution, puis le modifier pour qu'il ne fasse que la première substitution et ce sans utilisation de la coupure (« cut »).

Le thème sur les opérations ensemblistes permet aussi de mieux maîtriser les manipulations de listes.

3. Thème arithmétique

On souhaite que le prédicat « element » permette un accès par le rang.
Ainsi :
?-element(3,X,[a,b,c,d]). devrait réussir avec X=c.
On veut pouvoir aussi retrouver le rang :
?-element(I,a,[a,b,a]). devrait fournir les réponses I=1 et I=3.
Il serait bien que le même prédicat puisse être utilisé pour ces différents modes d'appel (c'est en fait le travail réalisé par le prédicat prédéfini « nth1/3 »).

4. Thème « ensembles »

On représentera des ensembles (en extension) par des listes (sans répétition) d'objets. Ecrire un prédicat pour tester si une liste est bien un ensemble (pas de répétition). Ecrire un prédicat qui produise un ensemble (enlève les doublons). Ainsi « list2ens([a,b,c,b,a],E) » devrait réussir avec E=[a,b,c]. Evidemment on s'interdira l'utilisation de « sort » et de « setof ». On peut ensuite décrire l'union, l'intersection, la différence, l'égalité (sans utiliser « sort ») de deux ensembles. Quel aurait été l'avantage, dans ce dernier cas, d'utiliser sort/2 (tri selon « @< ») ?

5. Thème sur les chaînes de caractères et opérateurs

En prolog 'Bonjour' est un symbole et les « quotes » masquent le fait que ce devrait être pris pour une variable. Mais "bonjour" est considéré comme une liste de codes ASCII.
Ainsi ?-"bonjour"=[98,111,110,106,111,117,114] réussit.
Le prédicat « name(Nom,Chaine) » permet d'éclater un nom en la liste des codes ASCII et réciproquement. Par exemple, on peut concaténer deux noms ainsi :
?- name(jean,N1),name('Paul',N2),append(N1,N2,N3),name(Result,N3), Result==jeanPaul. Réussit.

Ecrire un prédicat qui teste si un nom est avant l'autre dans l'ordre alphabétique. Ainsi, « ?- avant(adam,eve). » doit réussir, mais « ?- avant(adam,ada). » doit échouer. Faire que « avant » soit reconnu comme opérateur, c'est-à-dire que l'on puisse écrire indifféremment « avant(adam,eve) » ou « adam avant eve » (voir le prédicat prédéfini « op/3 »).

NB. Pour cette phase d'appropriation, cette liste de thèmes n'est pas exhaustive, vous pouvez aussi regarder les prédicats de parcours de graphes qui ont été présentés, essayer de construire et d'exploiter une table arborescente pour gérer efficacement un dictionnaire, développer l'arborescence des coups possibles dans un jeu à 2 joueurs simple comme un morpion, ou encore regarder des problèmes typiques de programmation sous contraintes (colorations de cartes/graphes et ordonnancements, puzzle logiques comme les sudokus, etc).