Séance 1 du Projet 4IF ALIA – Septembre et Octobre 2013
Découverte de Prolog

J-F. Boulicaut – M. Kaytoue

Vous devrez remettre, au plus tard en semaine 43 un compte-rendu court (6 pages maximum) qui, sur l'ensemble des thèmes proposés, montrera votre effort d'appropriation des concepts du langage (explication du codage des prédicats, usage de mécanismes particuliers (voir la liste de compétences jointe), discussion des difficultés, approche rigoureuse dans les tests avec notamment l'étude des différentes configurations d'appel). Ce document comportera également le sujet que vous vous proposez de traiter dans votre projet de programmation (description sur une page maximum).
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 « @< ») ?

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).

Projet 4IF ALIA – Prolog - 2013

Vous devez proposer un sujet de projet qui pose un problème dont la résolution s'effectuera en Prolog de manière non triviale. On mettra l'accent sur les jeux à un ou deux joueurs, sur des problèmes de programmation sous contraintes, ou encore des problèmes de génération de labyrinthe que l'on parcourt.

Dans tous les cas, on doit pouvoir évaluer la solution du problème choisi selon des critères quantitatifs que vous définissez. L'évaluation qualitative sera basée sur les notions comprises et utilisées à bon escient dans votre programme Prolog, et votre aptitude à les restituer dans votre rapport et pendant votre présentation. Le projet se réalise en hexanôme, il est donc attendu que plusieurs solutions soit proposées et comparées entre elles.

Prenons par exemple le cas du jeu de Tic-tac-toe (morpion) : vous décrivez plusieurs manières de le résoudre : une solution "aléatoire", une solution qui explore l'arbre complet de l'espace de recherche, et deux solutions basées sur des heuristiques.

Le code Prolog écrit doit être modulaire : sa structuration doit permettre de pouvoir intégrer toute nouvelle solution de manière immédiate selon des signatures de prédicats bien définies. Le coeur du programme gère les structures de données (par exemple comment représenter un plan, l'état du jeu). Il appelle alors diverses solutions, chacune écrite dans un module séparé. Un module de tests unitaires et un module d'évaluation/profilage des solutions sont également attendus. Ce dernier peut par exemple "faire jouer" 100 parties entre chaque couple de méthodes de résolution. On mettrait en avant alors le pourcentage de victoires d'une solution, le temps d'exécution de chaque "coup" (somme, moyen, etc.), etc. Cela vous permettra de mettre en exergue vos résultats et de les expliquer dans votre rapport de projet qui accompagnera la soutenance.

Chaque hexanôme désigne un chef de projet. Il n'est pas attendu une gestion de projet excessive (pas de livrable spécifique), car la priorité reste, pour ce projet court, la pratique du Prolog par tous les membres avec l'utilisation et l'écriture de prédicats non triviaux. Par exemple, on peut répartir, de manière chevauchante ou non les différentes tâches suivantes : gestion de projet, architecture et structure de données, approche qualité pour les tests unitaires et la gestion des exceptions, développement de solutions, évaluation des solutions.

Quelques idées. Puissance 4, bataille navale, Tic-Tac-Toe, le compte est bon, des chiffres et des lettres, le problème du voyageur de commerce, trouver un plus court chemin, génération de labyrinthe, pac-man, Starcraft Broodwar (voir sujet annexe), et tout autre idée que les enseignants valideront en seconde séance.

Dates importantes.
Semaine 40 : Séance d'exercices
Semaine 41 : Choix du sujet et attribution des rôles dans le rendu des exercices – début du travail de développement sur le projet de programmation
Semaine 42 : Séance complète dédiée au projet de programmation
Semaine 43 : Rendu des rapports (support papier à remettre dans le casier de l'un des enseignants).
Semaine 43 : Rendu du projet de programmation avec démonstration (salle à définir)