PROJET INTERPRETEUR PROLOG

en Java

Programmation comparée – LU3IN032

Camille Palisoc et François-Xavier Drouard

I. Introduction

Ce projet consiste à programmer un interprète de Prolog. À partir d'un fichier écrit en Prolog contenant des règles et des buts, nous devons renvoyer la solution trouvée à partir de l'unification de règles et de buts. L'interprétation d'un programme s'est faite petit à petit, en commençant par un interprète restrictif, n'acceptant qu'un seul fait et un seul but, pour finir par un interprète qui peut chercher plusieurs solutions si l'utilisateur le souhaite.

Pour coder ce projet, nous avons choisi le langage objet Java. Nous sommes tous les deux très familiers avec ce langage, nous permettant de manipuler avec aise les éléments crées pour résoudre le problème et implémenter une interface graphique. La division en classes nous a permis de bien structurer notre code et attribuer un rôle précis à chaque objet pour la programmation de chaque étape du projet.

Notre programme peut donc être divisé en quatre ensemble de classes avec des rôles précis. Nous allons les présenter dans la partie suivante.

II. Structure du programme

1) Les classes pour l'unification

Nous avons créé les classes Environnement, pour associer une variable à un terme, et Equation, assemblant deux termes pour leur unification. Ces deux classes sont utilisées dans la classe Systeme dans laquelle les méthodes pour l'application des règles d'unification sont implémentées. Ces classes utilisent les classes fournies des interfaces Term et TermVisitor ainsi que la classe Predicate. Elles forment l'ensemble de classes le plus « bas », servant pour les opérations de base et manipulant les Term directement pour la résolution d'un problème.

2) Les classes pour l'interprétation d'un programme Prolog

Les classes énoncées précédemment peuvent être considérées comme un travail en amont pour l'interprétation des programmes. La classe « statique » Interprete nous sert à rassembler tous les algorithmes pour l'interprétation et la résolution d'un programme Prolog. Les algorithmes *interprete0* à *interprete5* ainsi que les méthodes *choose* et *solve* sont toutes statiques car elles ne dépendent pas d'une instance particulière d'interprète.

Cette classe manipule les classes fournies Program, Predicate ainsi que celles implémentant les interfaces Decl et DeclVisitor. Elle se sert aussi de la classe Systeme pour sa méthode *unify* qui renvoie une solution du problème. Comme l'unification n'est pas directement applicable sur l'arbre syntaxique abstrait, chaque interpréteur effectue un travail de transformation pour pouvoir manipuler des Term, qui sont ensuite passés au Systeme.

3) Les classes pour l'exécution des jalons

Une autre catégorie de classes créées pour le projet sont les classes exécutables pour les jalons et tests pour quelques classes. Elles servent à montrer l'exécution et le résultat fourni par les

algorithmes implémentés. L'exécution est surveillée par l'affichage et non encadrées par des tests Junit, que l'on aurait eu l'avantage de faire proprement avec le langage choisi.

L'unification était dans un premier temps très détaillée dans la sortie standard lors de l'exécution de la classe Jalon1. Pour voir ces affichages, il suffit de décommenter les instructions dans les méthodes d'application de règles dans la classe Systeme.

4) Les classes pour l'interface graphique

Pour l'interface graphique, les classes implémentées sont dans les packages pcomp.Gui et pcomp.IO. La classe pcomp.Gui.Gui lance l'interface graphique.

Nous donnons la possibilité à l'utilisateur de passer un fichier .pl à l'interprète mais aussi celle d'écrire directement dans la zone de texte supérieure qui pourra à la suite être enregistré. Nous pouvons interpréter plusieurs programmes Prolog à la fois avec l'option de dupliquer la fenêtre.

L'interpréteur utilisé dans l'interface graphique est le dernier implémenté, c'est-à-dire l'interprete5 qui permet de trouver plus d'une solution. L'output étant affiché une fois pour toutes, les solutions trouvées au fur et à mesure sont affichées dans la fenêtre qui apparaît pour demander à l'utilisateur s'il veut chercher une autre solution. Pour tester d'autres interprètes implémentés dans des jalons précédents, il faut modifier l'appel à la méthode interprete dans la classe Lanceur et veiller à remplacer tous les `System.out.println` par `Tools.addText` pour retrouver les affichages sur la sortir output.

III. Conclusion

Nous sommes conscients que Java n'est pas spécialement le langage le plus adapté pour avoir un code concis. Beaucoup de nos fonctions utilisent le système de délégation car les objets sont encapsulés dans d'autres. Cela peut contribuer à de la répétition de code, le rendant verbeux, ce qui n'arriverait pas avec d'autres langages. Néanmoins nous pensons que la répétition due à la délégation du travail des fonctions aux autres objets aide à la clarté du code.