Traduction de spécifications algébriques vers Prolog et Coq

Nicolas Magaud et Pascal Schreck

Sujet TER M1 2008-2009

Les spécifications algébriques en utilisant un langage de la famille OBJ peuvent se voir comme une manière fonctionnelle de décrire des structures de données et des algorithmes par l'utilisation de systèmes de réécriture et de filtrage. C'est une vision un peu partielle des choses qui ne traduit qu'un fragment des spécifications algébriques, mais celui-ci peut s'implanter en Prolog en utilisant l'unification et le filtrage pour implanter les systèmes de réécriture. On peut également traduire de telles spécifications vers un système d'aide à la preuve formelle comme Coq.

1 Traduction de spécifications algébriques en programme Prolog

Ce sujet de TER propose d'étudier et d'écrire diverses manières de traduire des spécifications algébriques simples en Prolog. Plusieurs angles d'attaque sont proposés :

- implanter des systèmes généraux de réécriture de termes en Prolog (un chapitre de livre qui étudie la question sera inclu dans l'étude) : ceci revient, en gros, à écrire un interprète de spécifications algébriques en Prolog ;
- implanter un traducteur cafeOBJ vers Prolog dans le cas simple où il n'y a pas de relations entre les constructeurs (et où de tels constructeurs son explicitement désignés) : le programme Prolog obtenu est un vrai programme Prolog dans lequel on pourra poser des questions qui ne seraient pas envisageable en OBJ ;
- essayer de généraliser ceci à des spécifications algébriques équationnels ou conditionnelles-équationnels quelconques.

Exemple classique : les entiers de Peano !
Le programme Prolog suivant (ou un programme similaire) devrait pouvoir être produit automatiquement à partir de la spécfication classique des entiers naturels vue un certain nombre de fois dans le cours de Sémantique et Spécifications algébriques.

```
nat(0).
nat(s(X)) :- nat(X).
```

```
plus(0,X,X).
plus(s(X),Y,s(Z)) :- nat(X), nat(Y), plus(X,Y,Z).
```

2 Traduction de spécifications algébriques vers l'assistant de preuve Coq

Ce sujet de TER propose d'étudier comment traduire des spécifications algébriques vers le langage de description de programmes fonctionnels de l'outil de preuve interactive Coq (http://coq.inria.fr). Dans un premier temps, on s'intéressera aux spécifications algébriques simples où les constructeurs de la sorte décrite sont clairement distingués des autres opérations. On étudiera également la manière de décrire les fonctions structurellement récursives telle que l'addition sur les entiers. Dans un deuxième temps, on cherchera à voir comment traduire des descriptions de fonctions récursives générales, et notamment les problèmes liés à la terminaison des fonctions dans Coq.

Un des avantages du système Coq est qu'il permet de faire des démonstrations formelles sur les structures de données et les fonctions définies. En particulier, à chaque déclaration d'une structure de données inductive, un principe d'induction est automatiquement généré par le système. On pourra donc étudier comment établir des preuves formelles à partir des preuves informelles réalisables sur la spécification algébrique considérée.

Pour reprendre l'exemple des entiers de Peano, l'objectif serait de produire à partir de la spécification faite en OBJ, le code suivant qui est acceptable par Coq :

```
Inductive nat : Set := O : nat | S : nat -> nat.

Fixpoint plus (n m:nat) struct n  : nat
match n with
| O => O
| (S p) => S (plus p m)
end.
```

On pourra aussi établir formellement les propriétés de base de ces opérations :

```
Lemma plus_n_O : forall n : nat, plus n O = n.
induction n.
simpl; reflexivity.
simpl; rewrite IHn; reflexivity.
Qed.
```