TD 2 : Quelques théories utiles en programmation

Yann Régis-Gianas (yrg@pps.jussieu.fr)
GL6 - Licence 3 - Université Paris 7

17 février 2009

Les structures de données courantes de la programmation séquentielle sont intégrées à Why sous la forme de théories pré-existantes. Parmi elles, la **théorie des tableaux** sert à raisonner sur des cellules indexées séquentiellement et la **théorie des entiers relatifs** est utilisée pour dénoter l'ensemble $\mathbb{Z}$. La manipulation de ces deux théories est l'objet de ces travaux dirigés.

Au terme de cette séance, vous devrez :

- ✓ avoir révisé la syntaxe des formules logiques de Why ;
- ✓ connaître les opérations et les axiomes de la théorie des tableaux ;
- ✓ avoir pris connaissance des opérations de $\mathbb{Z}$ fournies par Why ;
- ✓ savoir déterminer si une formule de Why se trouve dans le fragment de l'arithmétique linéaire ;
- ✓ être capable d'écrire des spécifications fonctionnelles à l'aide de ces deux théories pour des programmes simples.

1 Écrire une spécification à l'aide de Why

Nous avons appris, lors du dernier TD, la syntaxe de Why dédiée à la définition de théories et d'énoncés dans ces théories. Ces énoncés peuvent être déclarés comme des buts à prouver ou des hypothèses sur le modèle, mais aussi en tant que **spécifications** de programmes.

La spécification d'une fonction est un couple formé de deux prédicats. Le premier, la **précondition**, porte sur les paramètres de la fonction et représente son domaine d'utilisation. Le second, la **postcondition**, spécifie la relation entre l'entrée et la sortie de la fonction.

Dans Why, on peut supposer l'existence de fonctions – sans en donner la définition – qui vérifient des spécifications données. Ces fonctions font parties du modèle, elles ne sont pas vérifiées. Par exemple, si on se donne la théorie suivante pour modéliser des ensembles finis munis de relations d'ordre :

```
type set
type element
logic mem : element, set → prop
predicate equal_sets (s1, s2) = forall x: element. mem (x, s1) ↔ mem (x, s2)
logic pick : element, set → set
logic less_than : element, element → prop
logic empty : set
axiom pick_remove : forall x: element. forall s: set. not (mem (x, pick (x, s)))
```

on peut alors supposer l'existence de la fonction min qui calcule le plus petit élément d'un ensemble non vide :

```
parameter min : s: set → { s ≠ empty } element { forall x: element. mem (x, s) → less_than (result, x) }
```

On peut aussi écrire et vérifier des programmes s'appuyant sur ces fonctions. Par exemple, voici un programme qui teste si deux ensembles finis sont égaux :

```
let rec check_equals_set (s1 : set) (s2 : set) : bool =
  ((s1 = empty) ∧ (s2 = empty))
  ∨
  (let min1 = min (s1) in
   let min2 = min (s2) in
     (min1 = min2) ∧ check_equals_set (pick min1 s1) (pick min2 s2))
```

Nous écrirons de tels programmes dans le TD suivant. Pour le moment, nous allons nous contenter d'écrire des spécifications et de vérifier des compositions de spécifications.

2 Théorie des entiers relatifs

Présentation

Exercice 1 (Propriétés vraies ou fausses ?)
❶ □

Fragment décidable de l'arithmétique linéaire

Exercice 2 *Calcul de la racine carée entière*
□

3 Théorie des tableaux

Présentation
Exercice 3 (Calcul de l'indice du minimum d'un tableau) □
Exercice 4 (Échange entre deux cellules) □
Exercice 5 (Tri par sélection) □