Vérification et Validation

Année 2019-2020

www.lri.fr/∼wolff/teach-material/2019-20/PolyTech-VnV

Prof. Burkhart Wolff
wolff@lri.fr

Delphine Longuet
longuet@lri.fr

TP - Preuve de programmes avec Why3

Semaine du 1 avril 2020

L'objectif de ce TP est de spécifier et de prouver les programmes fournis à l'aide de l'environnement de preuve Why3 et du prouveur automatique Alt-Ergo. On utilisera une version en ligne de Why3 disponible à l'adresse suivante :

`http://why3.lri.fr/try/`

Les programmes à prouver sont à l'adresse :

`http://www.lri.fr/~blsk/AProuver.zip`

Dans l'environnement Why3, les programmes sont écrits dans un langage similaire à OCaml appelé WhyML. Le programme à prouver est d'abord annoté par sa spécification sous forme de pré et post-conditions. L'exécution de Why3 sur ce programme traduit le code annoté en un ensemble de conditions à prouver, exprimées sous la forme de triplets de Hoare. Les règles d'inférence du calcul de Hoare sont ensuite appliquées à ces conditions. Enfin, un prouveur (automatique ou interactif, ici Alt-Ergo) est appelé sur les formules de logique classique restantes (provenant de l'application de la règle de conséquence par exemple).

Premier essai Dans un programme, les spécifications sont introduites de la manière suivante :

```
let max (x y:int) : int =
  ensures { result >= x /\ result >= y }
  if (x > y) then x else y
```

La spécification d'un programme est donnée par une pré-condition (introduite par le mot clé `requires`) et une post-condition (introduite par `ensures`). Le mot clé `result` représente le résultat de la fonction.

▶ Ouvrez dans Why3 le fichier `max.mlw` contenant le programme précédent, puis lancez la preuve de ce programme en cliquant sur les engrenages. Le programme annoté est traduit en un ensemble de conditions à prouver (les obligations de preuve) et le prouveur automatique Alt-Ergo est exécuté sur ces formules.

À droite apparaît la liste des obligations de preuve (VC pour *verification conditions*). Pour les voir toutes, il faut faire un clic droit sur la dernière et choisir Split and prove. Why3 découpe alors l'obligation de preuve en un ensemble de sous-buts à prouver. On peut voir l'instruction du code et la formule logique correspondant à une obligation de preuve en cliquant dessus.

▶ L'outil réussit à prouver toutes les propriétés demandées. Peut-on en déduire que le programme est correct ?

▶ Changez la spécification pour capturer le fait que le résultat doit être le maximum de x et y et lancez de nouveau la preuve.

Exercice 1 (Racine carrée)
On considère le programme `sqrt.mlw` qui calcule la racine carrée entière d'un entier a.

1. *Spécification.* Annotez le programme par sa spécification sous forme de pré et post-conditions.

2. *Invariant.* Ajoutez l'invariant nécessaire pour prouver la boucle `while`.
3. *Terminaison.* Ajoutez une mesure de terminaison à la boucle (aussi appelé variant), c'est-à-dire une expression dont la valeur décroisse à chaque tour de boucle et soit bornée par une valeur minimale (atteinte à la sortie de la boucle). Il faut que la valeur de cette expression soit toujours positive ou nulle, sauf peut-être à la sortie de la boucle.

Exercice 2 (All zeros)
On considère le programme `all_zeros.mlw` qui renvoie `true` si les valeurs de toutes les cases d'un tableau d'entiers sont nulles et `false` sinon.

1. *Spécification.* Annotez le programme par sa spécification sous forme de pré et post-conditions.
2. *Invariant et terminaison.* Ajoutez l'invariant nécessaire pour prouver la boucle `while`, ainsi que la mesure de terminaison.

Exercice 3 (Minimum d'un tableau)
On considère le programme `mintab.mlw` qui renvoie le minimum d'un tableau d'entiers.

1. *Spécification.* Annotez le programme par sa spécification sous forme de pré et post-conditions.
2. *Invariant et terminaison.* Ajoutez l'invariant nécessaire pour prouver la boucle `while`, ainsi que la mesure de terminaison.

Exercice 4 (Recherche dichotomique)
On considère le programme `binarysearch.mlw` qui effectue une recherche dichotomique dans un tableau d'entiers : il renvoie l'indice de l'élément s'il existe dans le tableau et -1 sinon.

1. *Spécification.* Annotez le programme par sa spécification en termes de pré et post-conditions.
2. *Invariant et terminaison.* Ajoutez ensuite au programme l'invariant nécessaire pour prouver la boucle `while` ainsi que la mesure de terminaison.

Exercice 5 (Tri par sélection)
On considère le programme `tri_selection.mlw` qui trie un tableau selon l'algorithme de tri par sélection. On recherche le plus petit élément du tableau et on l'échange avec le premier élément, puis on cherche le minimum des éléments restants et on l'échange avec le deuxième élément, et ainsi de suite jusqu'à ce que le tableau soit trié.

Une fonction auxiliaire `swap` permet d'échanger les deux éléments se trouvant aux indices `i` et `j` d'un tableau `t`. On remarque que cette fonction n'est pas implantée. En effet, sa spécification est suffisante pour prouver la correction de toute fonction qui utilise `swap`.

Annotez les deux fonctions avec leurs spécifications, puis ajoutez les invariants de boucle afin de prouver la correction du programme. On remarque qu'ici l'utilisation de boucles `for` simplifie l'écriture des invariants, et assure la terminaison (il n'est donc pas nécessaire de trouver un variant).

Pour spécifier la fonction `swap`, il est nécessaire de comparer les valeurs de `t[i]` et de `t[j]` avant et après l'opération ; on utilisera l'opération `old` qui permet de récupérer la valeur d'une variable avant l'application d'une fonction.

Remarque : pour des raisons de simplicité, on ne cherchera à démontrer que la propriété relative au tri du tableau, et on omettra volontairement celle relative à la permutation des éléments.

`/\`	et	`result`	résultat de la fonction
`\/`	ou	`old a`	valeur de `a` avant l'appel de la fonction
`->`	implication	`forall i:int. 0<=i<n -> ...`	quantification universelle
`<->`	équivalence	`exists i:int. 0<=i<n /\ ...`	quantification existentielle

FIGURE 1 – Syntaxe des annotations dans Why3