TP Preuve. Logique de Hoare.

L'objectif de cette séance est de découvrir les différents types d'outils qui permettent de simplifier la preuve de programmes impératifs en utilisant la logique de Hoare et le calcul d'obligations de preuve.

Nous avons étudié en TD les règles de déduction de la logique de Hoare pour la correction partielle et totale de programmes. Nous les avons appliquées sans construire l'arbre formellement pour éviter les débordements (de tableau noir).

Nous allons exploiter l'outil *Why3*[1] qui calcule les obligations de preuve, c'est-à-dire les formules logiques qui doivent être valides pour assurer qu'un programme annoté partiellement est correct. Nous prouverons ces obligations manuellement en utilisant l'assistant de preuve *Coq* puis automatiquement avec plusieurs démonstrateurs automatiques.

Le calcul des plus faibles pré-conditions et la génération d'obligations de preuve est une technique qui évite l'annotation explicite de tout le programme ou la construction explicite de l'arbre de preuve.

L'utilisateur doit donner les pré-conditions et post-conditions de chaque fonction ainsi que les invariants et variants de chaque boucle. Il est ensuite possible de générer une formule logique dont la validité est équivalente à la correction totale du programme.

Cette formule est appelée obligation de preuve (*proof obligation* ou *verification condition* en anglais). Cette formule doit ensuite être prouvée manuellement avec des assistants de preuve (tel l'outil *Coq*) ou automatiquement.

Nous allons utiliser l'outil *Why3* pour construire les obligations de preuve et les traduire vers le format de *Coq* ou utiliser des outils de preuve automatique.

Les programmes doivent être écrits dans le langage interne de *Why3* (suffixe `.mlw`) semblable au langage ML.

1 Préliminaires

1. Si vous ne l'avez pas fait dans la séance précédente, lancer la commande `why3 config detect` depuis la même fenêtre de commande pour configurer *Why3* en fonction des outils de preuve disponible dans votre environnement. Si vous avez déjà utilisé cet outil les années précédentes, il est préférable de supprimer le fichier de configuration avec la commande `rm $HOME/.why3.conf`.

2 Preuve assistée avec l'outil Coq

1. Lancer l'outil `why3 ide` depuis une fenêtre de commande avec en paramètre le nom du fichier contenant le programme qui vous souhaitez vérifier.
2. Prouver la correction de l'algorithme de calcul de la factorielle ascendant (fichier `ascendant.mlw`) :
   (a) Sélectionner les obligations de preuve (`VC for factorielle`) ;
   (b) Utiliser la stratégie `Split VC` qui décompose en plusieurs obligations (initialisation de l'invariant, préservation de l'invariant, décroissance du variant et postcondition) ;
   (c) Utiliser le *Prover* coq pour générer un fichier contenant un squelette de preuve pour chaque partie qui lancera `coqide` sur le fichier généré ;
   (d) Construire la preuve comme lors des séances précédentes et sauvegarder celle-ci dans un fichier séparé. Il est conseillé d'introduire comme hypothèses les différents axiomes contenus dans le fichier qui seront nécessaires plus tard pour la preuve (utiliser `generalize`, puis `intro` pour disposer en hypothèses des axiomes pour faire ensuite les preuves). Ceci permettra aux tactiques évoluées comme `omega` de les utiliser. Vous pouvez utiliser les commandes `intuition` ou `firstorder` (logique des prédicats), `ring` (anneau égalitaire associatif commutatif des entiers munis de l'addition et la multiplication, elle permet de prouver des égalités sur de telles structures) et `omega` (aithmétique de Presburger : preuve d'existence de solutions pour des systèmes d'équations/inéquations linéaires) pour automatiser une partie de la preuve. La commande `apply` permet d'utiliser les axiomes définissant factorielle quand l'objectif de preuve possède la même forme qu'une des deux équations définissant factorielle.

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

3 Preuve automatique

Lorsque la commande `why3 ide` appliquée sur un fichier en langage interne de *Why3* a démarré l'interface graphique de l'outil de preuve, vous pouvez ensuite utiliser les différents outils de preuve automatique disponibles (alt-ergo, z3, ...) pour essayer de prouver la correction totale de votre programme.

Tous les outils n'arrivent pas à prouver la correction de tous les programmes corrects. En effet, l'arithmétique étant incomplète et la preuve automatique indécidable, les outils appliquent des heuristiques correctes mais incomplètes qui peuvent boucler à la recherche d'une preuve, ou tout simplement prendre un temps extrêmement long ou demander des ressources dont votre machine ne dispose pas.

Il est en général nécessaire de donner la spécification de la fonction que l'on souhaite vérifier sous la forme d'axiomes qui décrivent sa sémantique.

1. Prouver la correction de l'algorithme de calcul de la factorielle ascendant (fichier `ascendant.mlw`).
2. Compléter les annotations et prouver la correction de l'algorithme de calcul de la factorielle descendant (fichier `descendant.mlw`).

   Attention, il faut compléter le fichier avant de lancer la commande `why3 ide`.
3. Compléter les annotations et prouver la correction de l'algorithme de calcul de la division entière (fichier `division.mlw`).

   Attention, il faut compléter le fichier avant de lancer la commande `why3 ide`.
4. Compléter les annotations et prouver la correction de l'algorithme de calcul du PGCD (fichier `pgcd.mlw`).

   Attention, il faut compléter le fichier avant de lancer la commande `why3 ide`.