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 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). En premier lieu, 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. Nous allons ensuite construire ces preuves formellement en utilisant un assistant de preuve *j-Algo*.

1 Vérification à base d'obligations de preuve

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 l'appel d'outils de preuve automatique.

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

- Exécuter la commande `source /mnt/n7fs/ens/gamba/opam.sh` pour rendre visible les outils de preuve dans votre fenêtre de commande (vous pouvez ajouter cette commande dans votre fichier de configuration `.bashrc` pour éviter de l'exécuter régulièrement.

  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.

1.1 Preuve assistée avec l'outil Coq

Lancer l'outil `why3 ide` depuis une fenêtre de commande avec en paramètre le nom du fichier contenant le programme qui vous souhaitez vérifier.

1. Prouver la correction de l'algorithme de calcul de la factorielle ascendant (fichier `ascendant.mlw`. Pour cela, vous devez sélectionner les obligations de preuve (`VC for factorielle`) et utiliser la stratégie `split` qui décompose en plusieurs obligations (initialisation de l'invariant, préservation de l'invariant, décroissance du variant et postcondition). Vous devez ensuite utiliser le *Prover* coq pour générer un fichier contenant un squelette de preuve pour chaque partie puis utiliser l'outil `Edit` qui lancera `coqide` sur le fichier généré. Vous pouvez ensuite construire la preuve comme lors des séances précédentes et sauvegarder celle-ci dans un fichier séparé. Vous pouvez utiliser les commandes `ring` (anneau associatif commutatif des entiers munis de l'addition et la multiplication) et `omega` (systèmes d'équations/inéquations linéaires) pour automatiser une partie de la preuve.

1.2 Preuve automatique

La commande `why3 ide` appliquée sur un fichier en langage interne de *Why3* démarre l'interface graphique de l'outil de preuve. Vous pouvez ensuite utiliser les différents outils de preuve automatique disponibles pour essayer de prouver la correction totale de votre programme. Tous les outils

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

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`).
3. Compléter les annotations et prouver la correction de l'algorithme de calcul de la division entière (fichier `division.mlw`).
4. Compléter les annotations et prouver la correction de l'algorithme de calcul du PGCD (fichier `pgcd.mlw`).

2 Construction d'arbres de déduction

Nous nous appuyons sur l'outil *j-Algo* pour construire des arbres de déduction corrects en exploitant la logique de Hoare. Cet outil impose que les règles de la logique soient appliquées correctement, ainsi l'arbre de déduction est correct par construction. Vous devez saisir les annotations des propriétés (pré-conditions, post-conditions, invariants) sous la forme exacte attendue par chaque règle. L'outil vérifie automatiquement que les renforcements et affaiblissements sont corrects.

Celui-ci se présente sous la forme d'une fenêtre principale qui contient l'arbre de preuve à gauche, d'une pile de fenêtres à droite contenant le programme en cours de preuve et les annotations, et enfin d'une pile de boutons permettant de choisir la règle appliquée sur le nœud courant.

Le point essentiel est de donner aux annotations exactement la forme nécessaire à l'application des règles et d'introduire pour cela par renforcement/affaiblissement les propriétés intermédiaires nécessaires.

Lancer l'outil `sh /mnt/n7fs/ens/tp_pantel/j-Algo/j-Algo.sh` depuis une fenêtre de commande.

1. Prouver la correction de l'algorithme de calcul de la factorielle ascendant (fichier `ascendant.jalgo`). Charger pour cela dans l'outil le fichier correspondant.
2. Prouver la correction de l'algorithme de calcul de la factorielle descendant (fichier `descendant.jalgo`). Charger pour cela dans l'outil le fichier correspondant.
3. Prouver la correction de l'algorithme de calcul de la division entière (fichier `division.jalgo`). Charger pour cela dans l'outil le fichier correspondant.