

TD Feuille 3 — Preuve formelle

Exercice 1 (Inférences formelles). On manipule ici rigoureusement le formalisme des règles d'inférence de Hoare sur l'IR des commandes gardées vue en CM.

▷ **Question 1.** Avec les règles (y compris dérivées) vues en cours, montrer la règle dérivée de calcul de WP ci-dessous. C'est la règle qu'on utilise quand on cherche à inférer l'invariant de boucle à partir de la postcondition.

$$D \frac{\{I \wedge C\} S \{I\}}{\{I\} \mathbf{while} C \mathbf{do} S \{Q\}} I \wedge \neg C \Rightarrow Q$$

On considère la définition de fonction suivante dans l'IR des commandes gardées.

```
pr rem(x, &mut y)[
    x ∈ ℕ ∧ y ∈ ℕ ∧ x ≠ 0,
    y = @y mod x,
    while x ≤ y do y := y - x
]
```

On va montrer que cette définition est valide. Autrement dit, d'après le cours, il faut montrer ce lemme :

$$D \frac{}{\{@x \in \mathbb{N} \wedge @y \in \mathbb{N} \wedge @x \neq 0\} (x := @x; y := @y); \mathbf{while} x \leq y \mathbf{do} y := y - x \{y = @y \bmod @x\}}$$

▷ **Question 2.** On commence par supposer qu'on a un invariant I . Construire un arbre de preuve complet (en propagation par WP) avec les obligations de preuves sur I . Pour abrégé la construction de l'arbre, on note P (resp. Q) la précondition (resp. postcondition) du triplet de Hoare ci-dessus. Et on note W la boucle. Quand on pourra parcourir certains sous-arbres en propagation arrière (WP), on appliquera la technique vue à l'exo 2 de la Feuille 2 pour laisser ces sous-arbres implicites. A la fin, lister les obligations de preuve en fonction de I , avec leur noms usuels.

▷ **Question 3.** Il faut maintenant trouver le I sur lequel on peut prouver les obligations de preuves précédentes.

▷ **Question 4.** En utilisant la commande gardée de l'appel de procédure, montrer

$$D \frac{}{\{x = 19\} \mathbf{rem}(x - 15, \&mut x) \{x = 3\}}$$

On pourra abrégé la construction de l'arbre en combinant propagation avant et arrière.

Exercice 2 (Recherche linéaire dans un tableau). On considère le programme en figure 1, spécifié en FRAMA-C. Le tableau est ici donné sous la forme d'un pointeur (qui peut être alloué dynamiquement) et dont la taille n'est pas connu statiquement. Par rapport au schéma vu en cours, un accès en lecture à $t[i]$ produit un assert RTE "assert(\valid_read(t+i))" (dans

une mémoire implicite qui sera explicitée au moment du codage dans l'IR de vérification). Et la précondition sur le tableau est équivalente à

$$\forall i, 0 \leq i \leq \text{size} - 1 \Rightarrow \text{\textbackslash valid_read}(t + i)$$

La présence des 3 postconditions correspond au fait qu'elles sont établies en conjonction.

```

1 /*@ requires \valid_read(t+(0..size-1)) ;
2   @ assigns \nothing;
3   @ ensures -1 <= \result;
4   @ ensures 0 <= \result ==> \result < size && t[\result] == v;
5   @ ensures \result == -1 ==> \forall int k; 0 <= k < size ==> t[k] != v; */
6 int linear_search(int v, int* t, int size) {
7   int i ;
8   i=size-1;
9   while (i >= 0 && t[i] != v) i--;
10  return i;
11 }
```

FIGURE 1 – Recherche linéaire à compléter pour la preuve en FRAMA-C

▸ **Question 1.** Ajouter les assertions RTE.

▸ **Question 2.** On veut prouver les assertions RTE - sans chercher à prouver forcément les postconditions pour l'instant, sauf si ça simplifie la vie. Modifier le programme si besoin (la spécification, elle, ne doit pas être changée) et trouver l'invariant de boucle FRAMA-C qui permet de prouver ces assertions.

▸ **Question 3.** Compléter l'invariant afin de pouvoir établir les postconditions. On prouvera que la formule proposée est bien un invariant et on explicitera comment les 3 postconditions sont alors établies.

Exercice 3 (Théorie des tableaux en SMT-solving). On étudie comment raisonner sur les tableaux dans la logique. Ici, on ne s'occupe pas de la question débordements de tableaux, gérée au niveau des `assert RTE`, comme on l'a vu au TD précédent. On se base sur la théorie des tableaux définie par les 3 axiomes suivants :

$$A1: \forall t, i, j, x, (i = j \Rightarrow \text{select}(\text{store}(t, i, x), j) = x)$$

$$A2: \forall t, i, j, x, (i \neq j \Rightarrow \text{select}(\text{store}(t, i, x), j) = \text{select}(t, j))$$

$$A3: \forall t_1, t_2, (\forall i, (\text{select}(t_1, i) = \text{select}(t_2, i)) \Rightarrow t_1 = t_2)$$

▸ **Question 1.** La valeur d'un tableau est donc représenté comme une suite de store. En partant de la valeur initiale t_0 donner :

— la valeur t de t après la suite d'affectations `C "t[2]=0; t[5]=0; t[2]=1;"`

— le calcul des valeurs $t[2]$ et $t[5]$ pour le t calculé précédemment.

▸ **Question 2.** On définit l'échange de 2 éléments d'un tableau par :

$$\text{swap}(t, i, j) \hat{=} \text{store}(\text{store}(t, i, \text{select}(t, j)), j, \text{select}(t, i))$$

Prouver la propriété suivante : $\forall k, \text{swap}(t, k, k) = t$

-
- ▷ **Question 3.** En supposant qu'une théorie associée au type `int` définisse un ordre total \leq , proposer une définition du prédicat `sorted(t, l, h)` qui exprime si `t` est trié par ordre croissant sur les indices compris dans `l..h`
 - ▷ **Question 4.** Proposer une axiomatisation du prédicat $Permut(t_1, t_2)$ signifiant que t_2 est une permutation de t_1 .
 - ▷ **Question 5.** Proposer une postcondition exprimant le tri du tableau dans une variable `t` de type `int t[100]`. On notera `@t` la valeur initiale logique du tableau `t`.