

TD Feuille 2 — Preuve d'absence de RTE

Exercice 1 (Preuve semi-formelle). Le but de cet exercice est de raisonner sur le fragment de code C ci-contre, de façon “intuitive”, c’est-à-dire sans chercher encore à appliquer le formalisme du cours—pour déjà construire l’intuition avant de formaliser dans la suite.

<code>z=0;</code>	1
<code>while (x>0) {</code>	2
<code> z=z+y;</code>	3
<code> x=x-1;</code>	4
<code>}</code>	5

► **Question 1.** On considère l’exécution de ce code sous la précondition ci-dessous

$$x \in \mathbb{Z} \wedge y \in \mathbb{Z} \wedge x \geq 0$$

Que calcule-t-il (notamment dans la variable z)? Donner une postcondition caractérisant les valeurs en sortie de z , x et y . On notera $@a$ la valeur initiale de la variable a .

► **Question 2.** Prouver la postcondition par deux méthodes :

1. Utiliser d’abord une récurrence sur le nombre de tours de boucle i . Il faut inventer une propriété P qui est vraie à chaque passage par la ligne 2 (quelque soit i) et qui permet de prouver la postcondition quand on passe à la ligne 5. Cette propriété P s’obtient en généralisant la postcondition souhaitée, de manière à être prouvable par récurrence.
2. Adapter cette preuve pour utiliser le concept d’invariant de boucle vu en cours – qui permet de faire une preuve **sans introduire i explicitement**. On rappelle qu’il y a 3 obligations de preuves :

initialisation de l’invariant qui correspond au *cas de base* de la récurrence ;

préservation de l’invariant qui correspond au *cas de récurrence* (utilisant l’hypothèse de récurrence) ;

établissement de la postcondition où on prouve la postcondition à partir du lemme prouvé par récurrence.

► **Question 3.** Reprendre les 2 questions précédentes pour x quelconque ($x \in \mathbb{Z}$).

On considère maintenant que les variables sont déclarées dans un contexte `int x, y, z`. On considère aussi comme FRAMA-C que le type `int` correspond au type des entiers signés 32 bits en complément à 2, avec $\text{MinInt} = -2^{31}$ et $\text{MaxInt} = 2^{31} - 1$.

► **Question 4.** Insérer les assertions dans le programme pour garantir l’absence d’UB (ici débordement arithmétique), comme le ferait le plugin RTE de FRAMA-C, c’est-à-dire de façon syntaxique. Comme dans FRAMA-C, on supposera que quand une **variable** est de type `int`, alors sa valeur est garantie être dans les bornes : c’est un invariant implicite du typage. Le but des assert à la RTE est précisément de garantir la **préservation** de cet invariant.

► **Question 5.** Adapter la preuve précédente pour prouver que ces assert à la RTE n’échouent pas : quelle précondition (la plus faible possible) faut-il prendre pour garantir l’absence de débordement ?

► **Question 6.** Si on voulait juste prouver l’absence de débordement, sans prouver la postcondition générale, mais en gardant la même précondition, pourrait-on simplifier l’invariant de boucle ?

Exercice 2 (Calcul de WP). On considère le code A_0 en langage C de la Figure 1, dans le contexte “`int tab[6], x, v, y, z, t;`”. Le but de l’exercice est de **calculer** la plus faible précondition (WP) qui garantit que ce code s’exécute sans erreur. Contrairement à l’exo 1, il s’agit ici d’appliquer une méthode **complètement automatique** – sans aucun raisonnement.

```

1 v = 2*x;
2 if (x>0) y=x+1;
3 else z=x-3;
4 if (v>0) t=y-5;
5 else t=z+1;
6 tab[t]=3;

```

FIGURE 1 – A_0

$$\begin{array}{c}
 \frac{}{\{Q[x \leftarrow T]\} x := T \{Q\}} \quad \text{D} \frac{}{\{C \wedge Q\} \text{assert } C \{Q\}} \\
 \frac{\{P\} S_1 \{I\} \quad \{I\} S_2 \{Q\}}{\{P\} S_1; S_2 \{Q\}} \\
 \text{D} \frac{\{P_1\} S_1 \{Q\} \quad \{P_2\} S_2 \{Q\}}{\{(C \Rightarrow P_1) \wedge (\neg C \Rightarrow P_2)\} \text{if } C \text{ then } S_1 \text{ else } S_2 \{Q\}}
 \end{array}$$

FIGURE 2 – Calcul de WP (plus faible précondition)

► **Question 1.** Insérer les assertions RTE dans le code de la Figure 1.

On note A_1 le code obtenu à la question précédente après avoir enlevé la dernière ligne (l’affectation de `tab`). On interprète maintenant directement ce code comme une commande gardée (avec commandes dérivées) telle que vue en cours.

► **Question 2.** Exprimer avec un triplet de Hoare que sous une certaine précondition P l’exécution n’atteint pas de **abort**.

► **Question 3.** Calculer la plus faible précondition P pour lequel ce triplet de Hoare est valide. Pour cela, on applique les règles de dérivations dont la postcondition est réduite à une méta-variable Q . Les règles dont vous avez besoin sont rappelées en Figure 2. Ce calcul consiste alors à parcourir le programme en propageant les préconditions **en arrière** (de la fin vers le début). Vous simplifierez ces conditions au fur et à mesure. Et vous les écrirez en commentaire dans le programme (e.g. à la fin de la ligne où elles apparaissent comme postcondition).