Master 1 Informatique – PEP
Frama-C / WP

Guillaume Petiot

13-14 février 2014

L'objectif de ce TP est l'utilisation du greffon WP de Frama-C pour la vérification de programmes C. Un compte-rendu contenant vos réponses (au format TXT ou PDF uniquement) devra être envoyé par mail à `guillaume.petiot@cea.fr` à la fin de la séance.

Cet examen comprend 10 questions sur un total de 24 points.

Point ACSL : `invariant` et `variant` de boucle

L'`invariant` de boucle permet d'énoncer des propriétés sur les boucles sans forcément connaître le nombre d'itérations. En géénral, il exprime la partie de la propriété à prouver après la boucle et qui est vérifiée après les n premières itérations. Il doit être donné juste avant la boucle dans un commentaire ACSL :

```
loop invariant <condition>;
```

Le `variant` de boucle permet de prouver la terminaison de la boucle sans forcément connaître le nombre d'itérations (on parle alors de correction totale). Il doit être :

- une expression, fonction des variables du programme
- positif lorsqu'on commence une itération (lorsque la condition de la boucle et l'invariant de boucle sont vérifiés)
- strictement décroissant à chaque itération

Comme il ne peut diminuer infiniment, la terminaison de la boucle en découle. Il doit figurer dans le même commentaire ACSL que l'invariant de boucle et s'écrit :

```
loop variant <expression>;
```

Un exemple complet est donné par le fichier getMin.c avec une fonction qui retourne le minimum d'un tableau `t` de taille `n`.

Exercice WP6

Nous allons spécifier et prouver uniquement la terminaison des boucles des fonctions du fichier ex6.c.

Question1 (1 point)
Complétez les variants de boucles, faites les preuves et vérifiez qu'elles passent. Donnez vos variants de boucles et expliquez votre choix.

Exercice WP7

Nous allons spécifier et prouver le programme du fichier getMinSubarray.c. La fonction `getMinSubarray` est une version de `getMin` qui cherche le minimum dans le sous-tableau des indices $k..n-1$ du tableau `t` de taille `n`.

Question1 (1 point)
Complétez la spécification du programme en vous inspirant de getMin.c. Faites la preuve et vérifiez qu'elle passe. Donnez le programme complet.

Question2 (1 point)
Vous avez effectué la preuve de la **correction totale** (le programme termine et il vérifie sa spécification), qui est plus forte que la **correction partielle** (si le programme termine, alors il vérifie sa spécification). Quelle partie de la spécification vous permet d'établir la correction totale ?

Exercice WP8

Nous allons spécifier et prouver le programme du fichier all_zeros.c qui retourne 1 si tous les éléments du tableau `t` de taille `n` sont nuls, ou 0 sinon.

Question1 (3 points)
Complétez la spécification du programme, faites la preuve et vérifiez qu'elle passe. Donnez le programme complet.

Question2 (1 point)
Quelle partie de la spécification permet de prouver la post-condition pour un nombre quelconque d'itérations ?

Point ACSL : \at

Pour faire référence à la valeur d'un terme `t` au label `L` : `\at(t,L)`. Utilisez le label spécial `Pre` pour faire référence à la valeur d'un terme au début de la fonction.

Exercice WP9

Nous allons spécifier et prouver le programme du fichier ex9.c.

Question1 (4 points)
Complétez la preuve et vérifiez qu'elle passe. Donnez le programme complet.

Point ACSL : énoncés intermédiaires

Parfois un prouveur automatique peut ne pas réussir à prouver une propriété. Différentes possibilités d'indiquer une propriété intermédiaire (un axiome, un lemme, une assertion) existent dans le langage ACSL pour l'aider. Si ce résultat intermédiaire n'est pas prouvé par le prouveur, on devra s'assurer par ailleurs de sa validité (par exemple en utilisant un assistant de preuve comme Coq). Pour aider WP, un lemme est défini pour l'exercice WP10 :

```
/*@ lemma div_by_2_def:  \forall integer n; 0 <= n - 2 * (n / 2) <= 1; */
```

Exercice WP10

Nous allons spécifier et prouver le programme du fichier binary_search.c qui effectue une recherche binaire d'un élément donné (`query`) dans un tableau trié `arr` de taille `length`. Cette fonction retourne -1 si cet élément n'est pas trouvé, et son indice sinon. Pour cette preuve, nous aurons besoin de donner au prouveur un lemme sur la division par 2 qui peut être prouvé par ailleurs si le prouveur n'arrive pas à le prouver, et une assertion supplémentaire au milieu de la fonction pour diriger la preuve.

Question1 (4 points)
Complétez la spécification du programme, faites la preuve et vérifiez qu'elle passe (vous pourrez avoir besoin d'utiliser l'option `-wp-timeout T` pour laisser plus de temps à WP). Donnez le programme complet.

Exercice WP11

On appelle racine entière de $n \geq 0$, le nombre entier $r \geq 0$ tel que $r*r \leq n$ et $(r+1)*(r+1) > n$.

Question1 (4 points)
Spécifiez et prouvez le programme du fichier RacineInc.c qui calcule la racine entière.

Exercice WP12

Nous allons maintenant spécifier deux implémentations de fonctions calculant une valeur de la suite de Fibonacci. Une fonction logique `fib` a été définie pour vous aider à spécifier les fonctions `fibo` et `fibo2` :

```
/*@ logic integer fib(integer n) = (n <= 1) ?  n :  fib(n-1) + fib(n-2); */
```

Question1 (2 points)
Complétez la spécification de la fonction `fibo`. Faites la preuve et vérifiez qu'elle passe.

Question2 (3 points)
Complétez la spécification de la fonction `fibo2` : le contrat de fonction est le même que celui de `fibo`, écrivez le contrat de boucle. Faites la preuve et vérifiez qu'elle passe.