Spécification et preuve de programmes
Solution du devoir 2 : Spécification et preuves avec micro-C

Alain Giorgetti

Licence d'Informatique de l'Université de Franche-Comté 2020-21

Ce devoir doit être rendu sous la forme d'un unique fichier texte *NomPrenom*.c, où ***NomPrenom*** sont les 8 premières lettres de votre nom de famille, suivies des 8 premières lettres de votre prénom, sans espaces ni accents. Ce fichier doit commencer par un commentaire entre `/*` et `*/` indiquant vos nom et prénom. Ensuite, des commentaires entre `/*` et `*/` doivent indiquer le numéro de chaque question traitée, par exemple

```
/* Exercice 2 question 3 */
```

juste avant la réponse à la question 3 de l'exercice 2.

La preuve et l'exécution du contenu complet de ce fichier doivent être possibles avec l'interface de micro-C en ligne. Dans ce but, les parties incorrectes ou incomplètes doivent être entourées par `/*` et `*/`.

Lire l'énoncé jusqu'à la fin avant de commencer à traiter les exercices.

1 Spécification d'une propriété de tableau (2 points)

Reproduire et compléter la définition

```
predicate p1(int a[], int i) =
```

pour définir en micro-C la propriété que le tableau `a` est croissant jusqu'à l'indice `i` inclus, puis décroissant ensuite, en supposant que `i` est compris entre `0` et `length(a)-1`.

(2 points)

```
/*@
predicate p1(int a[], int i) = (forall k. 0 <= k < i -> a[k] <= a[k+1]) &&
  forall k. i <= k < length(a)-1 -> a[k] >= a[k+1];
*/
```

2 Remplissage d'un tableau d'entiers (5 points)

1. Reproduire et compléter la définition de fonction

```
void f2 (int b[], int k) {
  /*@ requires length(b) == 2*k;
      ensures ..; */
  int j = 0;
  while (..) {
    ..
  }
}
```

pour que le tableau `b`, supposé de longueur paire, contienne alternativement les entiers `0` et `1` : `b = { 0, 1, 0, 1, ..., 0, 1 }` (sans utiliser `%`).

2. Spécifier cette propriété sous la forme d'une postcondition.

3. Spécifier un invariant de boucle assez précis pour permettre de démontrer cette postcondition.

4. Spécifier un variant de boucle permettant de démontrer la terminaison de cette fonction.

5. Prouver ce contrat avec l'interface de micro-C.

(code : 2 points, postcondition : 1 point, invariant : 1 point, variant : 1 point)
Toutes les réponses sont dans le listing suivant :

```
1  void f2 (int b[], int k) {
2    /*@ requires length(b) == 2*k;
3        ensures forall i. 0 <= i < k -> b[2*i] == 0 && b[2*i+1] == 1; */ // Q2: 1pt
4    int j = 0;
5    while (j < k) {
6      /*@ invariant 0 <= j <= k;                                          // Q3: 0,5 pt
7          invariant forall i. 0 <= i < j -> b[2*i] == 0 && b[2*i+1] == 1; // Q3: 0,5 pt
8          variant k-j; */                                                // Q4: 1 pt
9      b[2*j] = 0;
10     b[2*j+1] = 1;
11     j++;
12   }
13 }
```

Listing 1: Solution de l'exercice 2.

3 Taille d'un entier naturel (13 points)

Par définition, la taille d'un entier naturel n est le nombre de chiffres minimal nécessaire pour écrire cet entier en base 10. Par exemple, l'entier $n = 2021$ est de taille 4. Cette taille est le plus petit exposant c tel que $n < 10^c$. L'objectif de cet exercice est de spécifier cette définition, puis de programmer et de vérifier une fonction C qui calcule cette taille pour tout entier naturel non nul.

1. Dans un commentaire micro-C, entre `/*@` et `*/`, reproduire les déclarations du symbole fonctionnel `power`, puis des axiomes et des lemmes demandés dans les questions 3 et 4 de l'exercice 4 du devoir 1.

(1 point)

```
/*@
function int power(int a, int b);

axiom power0 : forall x. power(x,0) == 1;
axiom powerS : forall x. forall n. n >= 0 -> power(x,n+1) == x * power(x,n);

lemma power1 : forall x. power(x,1) == x;
lemma power_sum : forall a. forall b. forall c. b >= 0 && c >= 0
  -> power(a,b+c) == power(a,b) * power(a,c);
*/
```

Compte-tenu de l'incomplétude de la preuve automatique, il est possible que ces lemmes ne soient pas démontrés par l'interface en ligne de micro-C.

2. Programmer une fonction

   ```
   int exp(int a, int b)
   ```

   qui calcule a^b selon l'algorithme donné dans l'énoncé de la question 1 de l'exercice 4 du devoir 1.

   (1 point, code dans le listing 2)

3. À l'aide du symbole fonctionnel `power`, spécifier que la fonction `exp` calcule a^b, après avoir ajouté en précondition que b doit être positif ou nul.

   (2 points, lignes 2 et 3 dans le listing 2)

4. Spécifier un variant de boucle permettant de démontrer la terminaison de cette fonction.

   (1 point, ligne 9 dans le listing 2)

5. Spécifier un invariant de boucle assez précis pour permettre de démontrer automatiquement ce contrat avec l'interface en ligne de micro-C.

   (2 points, lignes 7 et 8 dans le listing 2)

```
1  int exp(int a, int b) {
2  /*@ requires b >= 0;                 // Q3: 1 pt
3      ensures result == power(a,b); */ // Q3: 1 pt
4    int p = 1;
5    int i = 0;
6    while (i < b) {
7      /*@ invariant 0 <= i <= b;     // Q5: 1 pt
8          invariant p == power(a,i);   // Q5: 1 pt
9          variant b-i; */              // Q4: 1 pt
10     p = a*p;
11     i = i+1;
12   }
13   return(p);
14 }
```

Listing 2: Solution des questions 2 à 5 de l'exercice 3.

6. Reproduire et compléter la définition

   ```
   predicate has_size(int n, int s) =
   ```

afin que `has_size(n,s)` soit vrai si est seulement si `s` est la taille de l'entier `n`, supposé strictement positif.

(1 point, première ligne du listing 3)

7. Programmer une fonction

```
int getSize(int n) {
  //@ requires n >= 1;
```

qui calcule la taille de tout entier `n` strictement positif à l'aide d'un boucle qui calcule les puissances croissantes de 10 uniquement par des multiplications successives à partir de 1.

(2 points, code du listing 3)

8. Spécifier complètement cette fonction (postcondition, invariants et variant) pour que la démonstration de ce contrat soit automatique avec l'interface en ligne de micro-C.

(3 points : postcondition : 1 point, invariants : 1 point, variant : 1 point)

```
1  /*@ predicate has_size(int n, int s) =
2        power(10,s-1) <= n < power(10,s); */          // Q6: 1 pt
3
4  int getSize(int n) {                                // Q7 (C code): 2pts
5    /*@ requires n >= 1;
6        ensures has_size(n,result); */                // Q8: 1 pt
7    int s = 0;
8    int p = 1;
9    while (p <= n) {
10     /*@ invariant 1 <= p && 0 <= s;                 // Q8: 1 pt
11         invariant p == power(10,s);                 // for all the
12         invariant s > 0 -> power(10,s-1) <= n;      // invariants
13         variant n+1-p; */                           // Q8: 1 pt
14     p = 10*p;
15     s = s+1;
16   }
17   return(s);
18 }
```

Listing 3: Solution des questions 6 à 8 de l'exercice 3.