Université de Strasbourg
Certification du logiciel Feuille de TP

Preuve de programmes avec Frama-C

I) Nouveaux élements de syntaxe

Revoir le TP1 si besoin.

Syntaxe des ensembles non ordonnés de valeurs :

```
set ::=
    | expr                          // singleton
    | { expr, expr, ..., expr }     // ensemble contenant les éléments
    | \union(set, set, ..., set)    // union d'ensembles
    | (expr .. expr)                // intervale entre les deux valeurs
```

Pour exprimer l'appartenace d'un élément à un ensemble on peut utiliser la syntaxe `expr \in set`. On peut également exprimer le fait que deux ensembles sont égaux avec la syntaxe `set == set`.

Au lieu d'enchainer des implications, on peut spécifier qu'une fonction se comporte différemment selon les entrées en distinguant plusieurs comportements avec la syntaxe `behavior`. En voici un exemple pour une fonction calculant la valeur absolue :

```
/*@
    requires val > INT_MIN;
    assigns \nothing;
    ensures \result >= 0;

    behavior pos:
        assumes 0 <= val;
        ensures \result == val;

    behavior neg:
        assumes val < 0;
        ensures \result == -val;

    complete behaviors; // Au moins un comportement par entrée valide
    disjoint behaviors; // Au plus un comportement par entrée valide
*/
```

Les constantes `INT_MIN` et `INT_MAX` sont prédéfinies et peuvent être utilisées dans les spécifications après avoir importé `stdlib.h` (avec une ligne `#include "stdlib.h"`).

Pour que Frama-C échoue plus rapidement le lancer avec la commande :
`frama-c -rte -wp -wp-timeout=<nb_secondes> moncode.c`
Attention si la valeur du temps en secondes est trop faible, il pourra se mettre échouer sur des assertions vraies. La valeur par défaut est à 10 secondes.

II) Appels de fonctions

1. (a) On donne le code suivant :

```
/*@
  assigns \nothing;
  ensures \result <==>
    c \in \union(('a' .. 'z'), ('A' .. 'Z'), ('0' .. '9'));
*/
int is_alpha_num(char c) {
  return is_lower_alpha(c) || is_upper_alpha(c) || is_digit(c);
}
```

Coder et spécifier les fonctions `is_lower_alpha`, `is_upper_alpha` et `is_digit` de façon à ce que Frama-C accepte la spécification de `is_alpha_num`.

(b) On donne l'énumération suivante :

```
enum kind { LOWER, UPPER, DIGIT, OTHER };
```

Écrire une fonction qui classifie un caractère dans l'une de ces 4 catégories, la spécifier et vérifier que Frama-C peut statiquement valider la fonction de test suivante :

```
//@ assigns \nothing;
void test() {
    enum kind k = get_char_kind('h');
    //@ assert k == LOWER;
    k = get_char_kind('T');
    //@ assert k == UPPER;
    k = get_char_kind('5');
    //@ assert k == DIGIT;
    k = get_char_kind('#');
    //@ assert k == OTHER;
}
```

2. (a) Écire et spécifier une fonction `void swap(int* a, int* b)` qui prend deux pointeurs et échange leur contenu.

(b) En utilisant `swap` écrire et spécifier une fonction `void order2(int* a, int* b)` qui s'assure les contenus de a et b sont dans l'ordre croissant quitte à les inverser.

(c) En utilisant `order2` écrire une fonction `void order3(int* a, int* b, int* c)` qui s'assure que les contenus de a, b et c sont dans l'ordre croissant quitte à les inverser.

(d) Spécifier `order3`. On pourra utiliser l'égalité entre l'ensemble des nouvelles valeurs a, b et c, et celui des anciennes valeurs. Dans ce cas il faudra tout de même faire attention à distinguer les cas selon la valeur finale de b. Attention également à la séparation des pointeurs.

(e) Vérifier que vos spécifications sont suffisantes pour valider statiquement la fonction de test suivante :

```
//@ assigns \nothing;
void test() {
    int a = 1, b = 3, c = 2;
    order3(&a, &b, &c);
```

```
    //@ assert a == 1 && b == 2 && c == 3;

    a = 2, b = 1, c = 0;
    order3(&a, &b, &c);
    //@ assert a == 0 && b == 1 && c == 2;

    a = 2, b = 2, c = 1;
    order3(&a, &b, &c);
    //@ assert a == 1 && b == 2 && c == 2;

    a = 1, b = 2, c = 1;
    order3(&a, &b, &c);
    //@ assert a == 1 && b == 1 && c == 2;
}
```

III) Dépassements d'entier

1. (a) Programmer une fonction en langage C qui calcule la somme des entiers de 0 à n (sans faire de multiplication).

   (b) Montrer qu'elle vérifie cette spécification :

```
/*@
 requires 0 <= n <= 1000;
 ensures 2*\result == n*(n+1);
*/
```

   (c) Essayer d'enlever la borne supérieure de n. Commenter.

2. (a) Programmer une fonction en langage C qui prend en arguments trois tableaux de même taille et qui modifie le troisième tableau afin qu'il contienne les sommes des éléments des deux premiers tableaux deux à deux.

   (b) Spécifier le comportement de votre fonction, et vérifier la spécification avec Frama-C.

IV) Boucles multiples

1. (a) Programmer une fonction `concat` en langage C qui prend en arguments trois tableaux d'entier et modifie le troisième afin qu'il contienne la concatenation des deux premiers.

   (b) Spécifier le comportement de votre fonction, et vérifier la spécification avec Frama-C.

2. Programmer et vérifier une fonction `min_diff` qui calcule la différence minimale entre deux éléments d'un tableau et admet la spécification suivante :

```
#include "stdlib.h"
/*@
    requires size >= 2;
    requires \valid(&tab[0..size-1]);
    requires
        \forall int i; 0 <= i < size ==>
```

```
        \forall int j; 0 <= j < size ==>
        -INT_MAX <= tab[i] - tab[j] <= INT_MAX;
    assigns \nothing;
    ensures
        \forall int i; 0 <= i < size ==>
        \forall int j; 0 <= j < size ==>
        i != j ==>
        \result <= \abs(tab[i] - tab[j]);
    ensures
        \exists int i; 0 <= i < size &&
        \exists int j; 0 <= j < size &&
        i != j &&
        \result == \abs(tab[i] - tab[j]);
*/
int min_diff(int tab[], int size) {
    ...
}
```

On pourra utiliser la fonction `abs` de la bibliothèque standard.