Conception Formelle

202-2022

TP2: Première complication, les pointeurs

Vincent Penelle

Points abordés

- Pointeurs et contrats.

Récupérer l'archive suivante[1] et décompressez-la.

Exercice 1: Pointeurs et tableaux

1. Écrivez un contrat de fonction pour la fonction `max_ptr` de `exo1.c`, et prouvez-le (ça ne devrait pas être trop dur, on a fait la même fonction sans pointeurs au TP1).

2. Affichez les gardes RTE. Que pensez-vous qu'elles indiquent ? Rajoutez-les comme préconditions, et observez que Frama-C considère que tout est correctement prouvé.

> **Point technique:**
>
> Le prédicat `\valid(a)` indique que la zone mémoire est correctement allouée, et `\valid_read(a)` que l'on peut y lire, mais pas nécessairement y écrire. La distinction permet de désigner des pointeurs `const`, qui sont typiquement `\valid_read`, mais pas `\valid`.
>
> À chaque fois qu'on manipule des pointeurs, RTE vous demandera de prouver que ceux-ci sont bien alloués si l'on souhaite y lire ou écrire. En pratique, on les utilisera toujours comme précondition, la fonction malloc n'étant pas commode à manipuler et spécifier avec Frama-C.

3. Même question pour la fonction `sum_first_last`. Attention aux valeurs possibles de n.

[1] `https://www.labri.fr/perso/vpenelle/Enseignement/ConceptionFormelle/tp2.tar.gz`

Point technique:

Dans un contrat de fonction, si on veut faire référence explicitement à une valeur avant de l'appel de la fonction, on doit la mettre dans la fonction `\old(...)`. Par défaut, c'est ce qui est fait sur les arguments passés par valeurs (puisqu'ils ne peuvent pas être modifiés par la fonction), mais pour les valeurs pointées, ce n'est pas le cas, puisqu'elles peuvent subir des effets de bord. Par défaut, leur valeur est évaluée après l'appel dans une post-condition.

Exemple: `/*@ ensures \old(a[0]) == a[0];*/`

Vous noterez d'ailleurs que Frama-C place des `\old` autour des arguments de la fonction dans ce qu'il vous affiche : comme c'est la seule interprétation possible pour eux, les `\old` sont implicites.

4. Mêmes questions pour les fonction `swap_first_last` et `swap`. Attention, on veut maintenant écrire dans le tableau/pointeur, il doit donc être `\valid` et pas seulement `\valid_read` (et c'est bien ce que RTE vous demandera).

Exercice 2: Pointeurs ++

1. Écrivez un contrat de fonction pour la fonction `sort_ptr` de `exo2.c`, qui doit permettre de satisfaire les assertions générées par RTE. Attention, il ne doit PAS être satisfait par les implémentations présentes dans `wrong_sort_ptr1.c` et `wrong_sort_ptr2.c`.

2. Écrivez un contrat de fonction pour la fonction `sum_in_pointer`, qui doit permettre de satisfaire les assertions générées par RTE. De plus, vous assurerez que la fonction ne modifie pas la valeur de *b. À votre avis, pourquoi Frama-C n'arrive pas à prouver cette assertion ? Dans quel cas peut-elle être fausse ? (ne trichez pas, essayez de répondre à cette question avant de regarder la suite)

Point technique:

Le mot clé `\separated` permet d'assurer que deux pointeurs ne pointent pas vers la même zone mémoire (ce qui est a priori possible). Il est également possible d'assurer que plusieurs adresses sont disjointes deux à deux grâce au même mot clé.

Exemple: `/*@ \separated(a,b+(0 .. n));*/`.

3. Ajouter la clause `\requires` nécessaire pour que la fonction précédente soit maintenant prouvée.

Exercice 3: Assigns

1. Exécutez le plugin WP sur le fichier `assigns.c` et observez le résultat produit. Curieusement, les assertions concernant les pointeurs ne sont pas prouvés.
2. Cela est dû au fait que par défaut, Frama-C considère qu'une fonction peut modifier n'importe quelle valeur en mémoire globale (pas les variables locales d'autre fonctions, donc). Grâce à l'indication ci-dessous, modifiez les contrats des fonctions `add_one` et `dummy` pour que WP réussisse à prouver toutes les assertions.
3. L'assertion `*b == \old(*b);` n'est toujours pas prouvée. Pourquoi ? (cf exercice précédent)

Point technique:

La clause `assigns` permet d'indiquer quelles valeurs la fonction peut être amenée à modifier lors de son exécution. Si elle ne modifie rien, on peut écrire `assigns \nothing`. Attention, WP réussira à prouver tout sur-ensemble des valeurs que la fonction modifie vraiment. Ainsi, une fonction ne modifiant que la valeur `*a` aura les clauses `assigns *a,*b;` et `assigns *a;` prouvées par WP, mais pas la clause `assigns *b;`.

Exercice 4: Utiliser Frama-C pour trouver des bugs

1. Lancez Frama-C sur le fichier write.c.
2. Identifiez les bugs grâce à WP et RTE et corriger le code de la fonction writes pour qu'elle satisfasse sa spécification.

Exercice 5: Produire du code sûr

1. Implémentez les fonctions présentes dans `bonus.h` en respectant leurs spécifications informelles. Vous prendrez soin de donner un contrat de fonction dont les seuls clauses `requires` indiqueront que les pointeurs manipulés sont bien alloués (i.e., `\valid`) et qui seront prouvés par WP lorsque l'on inclue les gardes générées par RTE. Incluez également les clauses `assigns` de vos fonctions.

   Si besoin, pour augmenter la lisibilité de vos spécification, utilisez des comportements et des prédicats.