Master 1 Informatique – PEP
Frama-C / WP / Value Analysis

Guillaume Petiot

06-07 février 2014

L'objectif de ce TP est la découverte des greffons preuve de programme (WP) et d'analyse de valeurs (Value) de Frama-C et du langage de spécification ACSL.

Point ACSL (ANSI C Specification Language)

Les spécifications (annotations) ACSL sont des commentaires C de deux types :

- `/*@ ... */`
- `//@ ...`

Annotations de base pour spécifier une fonction :

- `requires <condition>;` donne une pré-condition (supposée vraie)
- `ensures <condition>;` donne une post-condition (à prouver)
- `\result` fait référence à la valeur retournée par la fonction
- `\old(var)` fait référence à la valeur de `var` en entrée de la fonction

La post-condition et la pré-condition se terminent par un `;` et sont optionnelles. Si elles sont présentes, elles doivent apparaître dans le même commentaire et la pré-condition doit précéder la post-condition.

Les conditions (formules logiques) de base sont définies comme suit :

⟨*formule*⟩ ::= ⟨*expr*⟩
| ⟨*expr*⟩ ⟨*rel*⟩ ⟨*expr*⟩
| ⟨*formule*⟩ '`==>`' ⟨*formule*⟩
| ⟨*formule*⟩ '`<==>`' ⟨*formule*⟩
| ⟨*formule*⟩ '`&&`' ⟨*formule*⟩
| ⟨*formule*⟩ '`||`' ⟨*formule*⟩
| '`\forall`' ⟨*type*⟩ ⟨*ident*⟩ '`;`' ⟨*formule*⟩
| '`\exists`' ⟨*type*⟩ ⟨*ident*⟩ '`;`' ⟨*formule*⟩

⟨*rel*⟩ ::= '`==`'
| '`!=`'
| '`<`'
| '`<=`'
| '`>`'
| '`>=`'

La validité des locations mémoire peut être exprimée de la façon suivante :

- `\valid(p)` : validité de `*p`
- `\valid(p+(x..y))` validité de l'ensemble des {`*(p+x)`; ...; `*(p+y)`}

Vous trouverez la description exhaustive du langage ACSL dans le manuel (aussi présent dans le répertoire frama-c/doc/manuals) nommé acsl-implementation.pdf

Exercice WP0

Nous allons spécifier et prouver les fichiers ex0a.c, ex0b.c et ex0c.c. Pour lancer le greffon de preuve WP, utilisez la commande :

```
frama-c-gui ex0a.c -wp
```

Question1
Lancez la preuve pour le fichier ex0a.c (fonction `absval`).

Question2
Complétez la spécification du fichier ex0b.c (fonction `ToCm`). Lancez WP pour le prouver.

Question3
Complétez la spécification du fichier ex0c.c (fonction `PosOrZero`). Lancez WP pour le prouver.

Exercice WP1

Question1
Lancez la preuve pour le fichier ex1.c, est-il pouvé ?

Question2
Pourquoi la spécification de la fonction est-elle incomplète ? Commentez le corps de la fonction et donnez une autre version de la fonction qui satisfait cette spécification, mais ne retourne pas le maximum de ses deux entrées `x` et `y`. Vérifiez que cette version erronée passe aussi la preuve.

Question3
Corrigez la spécification et refaites la preuve du code initial (correct). La version erronée de la question précédente passe-t-elle la preuve maintenant ? Donnez le programme corrigé et expliquez la correction.

Question4
Conclusion : expliquez l'importance de la précision de la spécification.

Exercice WP2

Question1
Lancez la preuve pour le fichier ex2.c, est-il prouvé ? Rajoutez l'option `-wp-rte` pour vérifier l'absence d'erreurs à l'exécution et relancez la preuve. Expliquez l'échec de la preuve, le risque d'erreur est-il réel ?

Question2
Ajoutez une pré-condition telle que `n` est compris entre -100 et 100. Refaites la preuve et vérifiez qu'elle passe.

Exercice WP3

Question1
Lancez la preuve pour le fichier ex3.c avec l'option `-wp-rte`, pourquoi n'est-il pas prouvé ?

Question2
Ajoutez une pré-condition assurant la validité du pointeur `p`. Refaites la preuve et vérifiez qu'elle passe.

Point ACSL : `assigns`

Pour préciser les (locations mémoire des) variables que la fonction a le droit de modifier en dehors de ses variables locales, on utilise la clause `assigns v1, v1, ..., vN;`
Si la fonction ne doit modifier aucune variable non locale : `assigns \nothing;`

Exercice WP4

Nous allons spécifier et prouver le programme du fichier ex4.c en utilisant trois versions erronées : ex4_error1.c, ex4_error2.c et ex4_error3.c.

Question1
Examinez les trois versions erronées du fichier ex4.c, la preuve passe-t-elle pour ces versions erronées ?

Question2
Complétez les spécifications pour faire en sorte que la preuve réussisse pour ex4.c, mais pas pour les versions erronées.

Question3
Expliquez l'importance de la modification que vous venez de faire.

Point ACSL : spécification par cas, `behaviors`

Pour préciser plusieurs cas possibles, il peut être pratique d'utiliser les `behaviors`. En plus des pré-conditions et post-conditions communes pour tous les cas, chaque `behavior` peut avoir sa propre pré/post-condition. La clause `assumes` détermine dans quel cas un `behavior` s'applique. Par exemple :

```
/*@ requires -100 <= x <= 100;
assigns \nothing;
behavior pos:  assumes x >= 0; ensures \result == x;
behavior neg:  assumes x < 0; ensures \result = -x; */
```

Exercice WP5

Nous allons maintenant prouver les fonctions du fichier ex5.c.

Question1
Vérifiez que la preuve passe pour la fonction `absval`.

Question2
Écrivez la spécification de la fonction `PosOrZero` avec des `behaviors` et prouvez-la.

Question3
Écrivez la spécification de la fonction `max3` sans behaviors et prouvez-la.

Question4
Écrivez la spécification de la fonction `max3encore` avec des behaviors et prouvez-la.

Exercice VA0

Lancez l'analyse de valeurs sur le fichier max_VA.c avec la commande :

```
frama-c-gui -val max_VA.c share/builtin.c
```

Question1
Quel est le domaine de `M` après l'appel de la fonction `max` ? Cliquez sur une variable pour voir son domaine à un point du programme.

Question2
Pourquoi une partie du code est-elle inatteignable ?

Question3
Définissez le domaine de B comme [-200; 200] et relancez l'analyse. Que constatez-vous ?

Exercice VA1

Lancez l'analyse de valeurs sur le fichier divide_VA.c avec la commande :

```
frama-c-gui -val divide_VA.c share/builtin.c
```

Question1
Regardez quels sont les domaines des variables dans différentes instructions. Pouvez-vous les expliquer ?

Question2
Pourquoi une alarme (`/*@ assert ... */`) est-elle générée ?

Question3
Cette alarme est-elle réelle ou une fausse alarme ?

Question4
Relancez l'analyse avec l'option `slevel` qui permet de garder plusieurs états en parallèle :

```
frama-c-gui -val divide_VA.c share/builtin.c -slevel 3
```

Regardez de nouveau les domaines. L'alarme est-elle générée ? Pourquoi ?

Exercice VA2

Lancez l'analyse de valeurs sur le fichier RacineInc_VA.c avec la commande :

```
frama-c-gui -val RacineInc_VA.c share/builtin.c
```

Question1
Regardez quel est le domaine de `B` après l'appel de la fonction `RacineInc`.

Question2

Relancez l'analyse avec l'option `ulevel` qui permet de déplier les boucles 10 fois :

```
frama-c-gui -val RacineInc_VA.c share/builtin.c -ulevel 10
```

Quel est le nouveau domaine de `B` calculé ? Pourquoi une partie du code est-elle devenue inatteignable ?

Question3

Modifiez le domaine de `A` en mettant l'intervalle [0,4], puis [17,65]. Essayez différentes valeurs des options `ulevel` et `slevel`, observez le domaine du résultat.