Conception Formelle

TD-TP : Un peu de théorie et de pratique.

GALELA Nayar

2022-2023

Ce travail est à réaliser en binôme. La date de rendu est fixée au 2 mai 2023.

Les exercices sont par difficulté croissante. Les deux premiers demandent uniquement d'appliquer le calcul de WP et de trouver des invariants de boucle. Les réaliser parfaitement devrait valoir autour de 12-13 (4 pour le premier, 8-9 pour le second). Le troisième demande de démontrer une fonction avec deux boucles imbriquées, et consiste à déterminer les invariants et les démontrer avec frama-c. Il devrait être noté sur 4 ou 5. Le dernier demande de donner un code correct et sa spécification à partir d'une spécification informelle. Il est noté sur 2-3 points.

Ce barème n'est qu'indicatif, mais le message de celui-ci est de traiter les questions dans l'ordre.

Structure du devoir et rendu Le devoir est constitué du présent document, ainsi que de fichiers de code à compléter en parallèle des questions posées. Le code fourni est découpé ainsi : un fichier .c et .h pour chaque fonction. Ces fichiers sont à remplir au fur et à mesure.

Pour répondre aux questions, vous avez deux choix :

— Soit compléter le présent document .tex en plaçant vos réponses dans les blocs solutionorbox prévues à cet effet. Pour cela, référez-vous aux macros définies dans les sujets de TD. Si vous avez besoin de compiler avec la police OpenDyslexic, faites-le avec la ligne suivante :
`latexmk -xelatex -usepretex='\def\dyslexic{}' TD-TP.tex`

— Soit produire un document pdf par d'autres moyens (autre logiciel, scan), tant que c'est lisible, ça me conviendra.

Vous devrez rendre (sur la page moodle du cours) :

— Le présent document de réponse (ou votre version).

— Le dossier `code` complété. Dans les fichiers qui le composent, vous pourrez si besoin rajouter des commentaires expliquant comment prouver certaines spécifications si ce n'est pas immédiat, ou expliquer ce qui vous bloque.

Rappels de logique : On rappelle que :

— $p \Rightarrow q \equiv \neg p \vee q$

— $(p \wedge q) \Rightarrow r \equiv p \Rightarrow q \Rightarrow r$.

Vous aurez à manipuler des expressions comprenant des $\forall$. Pour ces expressions-là, lorsque vous appliquez un pas de WLP, grossièrement il vous faut découper votre formule entre partie modifiée et partie non-modifiée, et appliquer le calcul. Plus concrètement ici,

ce que vous devriez avoir, c'est quelque chose du genre :

$$\begin{aligned}&\mathrm{WLP}(t[i] = x, \forall j; 0 \leq j \leq i ==> \varphi(t[j]))\\ \equiv&\mathrm{WLP}(t[i] = x, (\forall j; 0 \leq j < i ==> \varphi(t[j])) \wedge \varphi(t[i]))\\ \equiv&(\forall j; 0 \leq j < i ==> \varphi(t[j])) \wedge \varphi(x)\end{aligned}$$

La règle la plus générale d'où vient cela est la suivante : pour φ et ψ des formules quelconques :

$$\forall j; \varphi(j) \equiv \forall j; \psi(j) ==> \varphi(j) \wedge \forall j; \neg\psi(j) ==> \varphi(j);$$

Un autre point, c'est «soyez paresseux» : si vous arrivez à un moment sur un calcul équivalent à $\bot \Rightarrow \mathrm{WLP}(i - j, \phi)$, il est inutile de calculer $\mathrm{WLP}(i - j, \phi)$ pour déterminer que l'implication est vraie.

On attendra que vos calculs de WP soient suffisamment détaillés, mais vous pouvez sauter quelques étapes si elles sont faciles (comme par exemples, appliquer plusieurs substitutions d'un coup). Évidemment, tant que cela est correct.

Comme dit souvent, pour les justification de vérité de formule, on n'attendra pas de preuves formelles, mais des justifications convaincantes (i.e., qui n'oublient pas de cas, mais on restera tolérant sur la forme). En gros, quand vous aurez une implication, une possibilité sera de dire un truc du genre «l'implication est vraie car telle et telle partie de la partie gauche impliquent bien la partie droite». Un «ben oui» (ou plutôt un remplacement par $\top$) sera admissible pour des propriétés du genre $\bot \Rightarrow \varphi$, $\varphi \Rightarrow \top$ ou $a < b \Rightarrow a \leq b$. Cependant, pour faire cela de manière convaincante, vous auriez intérêt à simplifier vos formules avant.

Dans le présent sujet, on donne une version normalisée du code pour vous faciliter preuves (if then else développés, un seul return, que des while). Évidemment, vous pouvez coder autrement, mais il sera plus aisé de faire ces restrictions sur papier.

Attention : on manipule beaucoup de unsigned int. En pratique, dans vos formules, *quantifiez sur des integer, et non des int.*

Un mot sur les preuves : À partir de l'exercice 3, certaines des preuves commencent à être difficiles pour les solveurs, aussi faites bien les trois points suivants :

- Vérifiez que le solveur Z3 4.8.10 (counter-examples) est activé (il parvient à démontré des propriétés où alt-ergo et les autres variantes de Z3 échouent – ne me demandez pas pourquoi). Attention, il y a trois variantes de Z3 installée au CREMI, et c'est bien celle avec counter-examples qui fonctionne dans certains cas. Pensez à cliquer sur «filter» dans l'onglet Provers de frama-c si vous ne le voyez pas.
- Si certaines propriétés ne sont pas prouvées, retentez la preuve : quand les solveurs tentent trop de preuves en même temps, il arrive que certaines timeout pour de mauvaises raisons. Les relancer peut régler le problème.
- Prouvez d'abord le contrat de fonction, avant d'ajoutez les gardes RTE, puis prouvez ces dernières. Dans certains cas, tenter de tout prouver d'un coup peut ne pas fonctionner.
- Si cela ne marche toujours pas (et que vous avez confiance en la propriété), cliquez sur le nom du but, puis sur la tactique «filter». Cela peut parfois régler le problème.

Si rien de tout cela ne marche, vous avez probablement oublié de spécifier certaines hypothèses (ou votre propriété est fausse), donc reprenez votre stylo.

Exercice 1 : Valeur absolue

On commence par une petit mise en jambe, la fonction valeur absolue. La formalisation

de `abs` vous est donnée dans `abs.h`. Il faut la lire comme si n est strictement positif, le résultat est n, sinon c'est $-n$ (comme on l'attend d'une valeur absolue).

Dans vos raisonnements, vous pourrez directement remplacer $abs(n)$ par n ou $-n$ quand il se trouve à droite d'une implication où le résultat est connu. Par exemple, si vous avez la formule $n > 42 \Rightarrow abs(n) >= 12$, vous pouvez la remplacer par $n > 42 \Rightarrow n >= 12$

```
1 /*@ ensures \result == abs(n);
2 */
3 int abs(int n)
4 {
5     int res = n;
6     if (n < 0)
7         res = -n;
8     return res;
9 }
```

(a) Si ce n'est pas déjà fait et que vous rendrez le présent .tex, mettez vos noms dans la balise author situé en haut de ce document (celle où il y a écrit un message assez explicite). (0 points)

(b) Calculez WP(`abs`, ψ) pour ψ la post-condition fournie, et déduisez-en un triplet de HOARE valide.

Réponse:

$$\begin{aligned}
&\text{WP}(5, \text{WP}(6-7, \text{WP}(8, \psi))) \\
\equiv\ &\text{WP}(5, \text{WP}(6-7, \psi[\texttt{\textbackslash result} \leftarrow res))) \\
\equiv\ &\text{WP}(5, n < 0 ==> \psi[\texttt{\textbackslash result} \leftarrow res][res \leftarrow -n] \\
&\wedge n \geq 0 ==> \psi[\texttt{\textbackslash result} \leftarrow res]) \\
\equiv\ &n \leq 0 ==> \psi[\texttt{\textbackslash result} \leftarrow res][res \leftarrow -n][res \leftarrow -n] \\
&\wedge n \geq 0 ==> \psi[\texttt{\textbackslash result} \leftarrow res][res \leftarrow n] \\
\equiv\ &n < 0 ==> abs(n) == -n \wedge n \geq 0 ==> abs(n) == n \\
\equiv\ &n < 0 ==> -n == -n \wedge n \geq 0 ==> n == n \\
\equiv\ &\top \\
&\top \texttt{abs} \psi est un triplet de Hoare valide.
\end{aligned}$$

(c) Évidemment, on a ici une correction partielle, dans le sens où on n'a pas tenu compte des comportements indéterminés. Quelle information manque-t'il pour assurer qu'il n'y aura pas d'erreur à l'exécution ?

Réponse:
requires -2147483647<=n ;

On considère un autre code pour calculer cette fonction :

```
1 /*@ ensures \result == abs(n);
```

```
2  */
3  int abs2(int n)
4  {
5      int aux = n % 2 + (n + 1) % 2;
6      return n * aux;
7  }
```

(d) Calculez WP(`abs2`, ψ) pour ψ la post-condition fournie. Justifiez que la formule obtenue est toujours vraie (on ne demande pas une preuve formelle mais une explication de la raison).

Réponse:

$$
\begin{aligned}
&\mathrm{WP}(5, \mathrm{WP}(6, \psi)) \\
&\equiv \mathrm{WP}(5, \psi([\backslash\texttt{result} \leftarrow n * aux])) \\
&\equiv \psi[\backslash\texttt{result} \leftarrow n * aux][aux \leftarrow n \mod 2 + (n+1) \mod 2] \\
&\equiv \psi[\backslash\texttt{result} \leftarrow n * (n \mod 2 + (n+1) \mod 2)] \\
&\equiv abs(n) = n * (n \mod 2 + (n+1) \mod 2) \\
&\equiv n = n(n \mod 2 + (n+1) \mod 2) \\
&\equiv 1 = n \mod 2 + (n+1) \mod 2 \\
&npair : 1 = 0 + 1 ==> 1 = 1 \\
&nimpair : 1 = 1 + 0 ==> 1 = 1 \\
&\top \texttt{abs2} \psi est un triplet de Hoare valide.
\end{aligned}
$$

(e) Quelle préconditions faut-il pour ne pas avoir de comportement indéterminé ? Justifiez.

Réponse:
requires n+1 <= 2147483647 ; requires -2147483647 <= n ;

(f) Compléter le fichier `abs.h` de manière à ce que Frama-C puisse démontrer la post-condition fournie, en ajoutant les assertions RTE. Il est bien évidemment possible (et encouragé) de répondre aux questions précédentes grâce à celle-ci. Attention, alt-ergo n'arrive pas à démontrer `abs2`, mais z3, oui.
Ajoutez également les clauses assigns et terminates pertinentes.

Exercice 2 : Max_dist

On veut maintenant une fonction qui calcule la plus grande différence entre deux éléments d'un tableau. On considère le code de la fonction, ainsi que la postcondition que l'on souhaite démontrer.

```
1          /*@
2          ensures \forall integer a,b; 0 <= a < b < n ==>
3          \result >= abs(tab[a]-tab[b]);
4      */
5      int max_dist(int *tab, unsigned int n)
6      {
```

```
7           int min = tab[0];
8           int max = tab[0];
9           unsigned int i = 1;
10          while (i < n)
11          {
12              if (tab[i] < min)
13                  min = tab[i];
14              if (tab[i] > max)
15                  max = tab[i];
16              i++;
17          }
18          return max - min;
19      }
```

(a) Quelle condition doit-être vraie à la sortie de la boucle (i.e, WLP$(18, \psi)$), pour ψ la post-condition ?

> **Réponse:**
> i==n ;

(b) Calculez WLP$(12 - 16, I)$ pour un invariant I quelconque. Mettez-la en forme de manière à avoir la conjonction de 4 implications (en appliquant que $P \Rightarrow (Q \wedge R)$ est équivalent à $(P \Rightarrow Q) \wedge (P \Rightarrow R)$, et $P \Rightarrow Q \Rightarrow R$ est équivalent à $(P \wedge Q) \Rightarrow R$).

> **Réponse:**
> METTEZ VOTRE RÉPONSE ICI.

(c) Proposez des invariants de boucle pour votre boucle. Vous devriez en avoir trois :
- I1 qui encadre la variable de boucle (i).
- I2 qui correspond à la formule obtenue à la question a.
- I3 supplémentaire (qui permettra de prouver le précédent) qui dit que toute valeur déjà vue est comprise entre `min` et `max`.
- I4 et I5 qui disent que min et max sont des éléments du tableau. Ces deux derniers invariants ne servent (pour l'instant) qu'à démontrer l'absence d'erreur à l'exécution.

On appellera I leur conjonction.

Justifiez que $\neg(i < n) \wedge I \Rightarrow$ WLP$(18, \varphi)$.

Pour chaque Ii, précisez WLP$(12 - 16, Ii)$ (utilisez le calcul de la question b), et justifiez que $i < n \wedge I \Rightarrow$ WLP$(12 - 16, Ii)$.

Pour I2, vous pouvez faire une justification très informelle que la formule obtenue est correcte.

Pour chacun des invariants, comme elles sont similaires, vous pouvez n'expliquer qu'une seule des quatre implications

> **Réponse:**
> METTEZ VOTRE RÉPONSE ICI.

(d) Déduisez-en un triplet de Hoare valide pour votre fonction.

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(e) Démontrez la terminaison de la fonction en donnant un variant de boucle et en démontrant qu'il décroît à chaque tour de boucle et qu'il est toujours positif (vous pouvez utiliser le calcul de la question b).

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(f) Quelles sont les valeurs mémoires modifiées par cette fonction (et la boucle) ? Vous répondrez en donnant les clauses assigns et loop assigns correspondantes.

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(g) Que manque-t'il comme précondition pour que la fonction ne puisse pas faire d'erreur à l'exécution ?

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(h) Démontrez que la fonction suivante vérifie le même contrat de fonction :

```
1 int max_dist(int *tab, unsigned int n)
2 {
3     return INT_MAX;
4 }
```

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(i) Donnez une postcondition vérifiée par la fonction correcte qui n'est pas vérifiée par celle-ci. Vos invariants devraient permettre de la vérifier (si vous l'écrivez sans valeur absolue).

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(j) Complétez les fichiers `max_dist.c` et `max_dist.h` de manière à ce que le contrat reflète tous les points vus ici et que frama-c accepte de prouver le contrat. Vous pouvez bien évidemment vous servir de cette question pour travailler les précédentes (c'est recommandé, d'ailleurs). On attendra bien sûr que toutes les assertions RTE soient démontrée, ainsi qu'une clause `terminates \true` et une clause assigns.

Exercice 3 : Min_dist

On veut maintenant une fonction qui calcule la plus petite différence entre deux élé-

ments d'un tableau. On considère le code de la fonction, ainsi que la postcondition que l'on souhaite démontrer.

Dans cette fonction, on utilise la fonction `abs` définie précédemment. On l'utilisera en admettant que $\mathrm{WP}(x = \texttt{abs}(t), \psi) = \psi[x \to abs(t)]$ (ce qui est ce qu'on a démontré à l'exercice 1, au final).

Dans cet exercice on ne va pas faire les preuves sur papier entièrement. On se contentera de calculer pour chaque boucle l'effet de la boucle sur les invariants pour mieux inférer les invariants nécessaires.

```
1   /*@ ensures \forall integer i; 0 <= i < n ==>
2   (\forall integer j; i < j < n ==> \result <= abs(tab[i]-tab[j]));
3   */
4   int min_dist(int *tab, unsigned int n)
5   {
6       int min = abs(tab[0] - tab[1]);
7       unsigned int i = 0;
8       while (i < n - 1)
9       {
10          int min_i = abs(tab[i] - tab[i + 1]);
11          unsigned int j = i + 2;
12          while (j < n)
13          {
14              int d = abs(tab[i] - tab[j]);
15              if (d < min_i)
16                  min_i = d;
17              j++;
18          }
19          if (min_i < min)
20              min = min_i;
21          i++;
22      }
23      return min;
24  }
```

(a) Quelle condition doit-être vraie à la sortie de la boucle extérieure (i.e, $\mathrm{WLP}(23, \psi)$), pour ψ la post-condition ?

> **Réponse:**
> $i == n$

(b) Calculez $\mathrm{WLP}(14 - 17, J)$ pour un invariant J quelconque.

> **Réponse:**
> METTEZ VOTRE RÉPONSE ICI.

(c) Calculez $\mathrm{WLP}(19 - 21, I)$ pour un invariant I quelconque (pour déterminer ce qui doit être vrai à la sortie de la boucle interne, en fonction de I).

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(d) Calculez WLP$(10-21, I)$ pour un invariant I quelconque, en admettant que l'invariant de la boucle interne (14-17) est J.

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(e) Donnez les clauses loop assigns et assigns pour cette fonction.

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(f) Proposez des invariants pour les deux boucles – on ne vous demande pas de démontrer la correction sur papier, mais uniquement de vérifier avec frama-c que vos invariants en sont et sont suffisants. Pour chaque boucle, deux invariants devraient suffire. Pour la boucle extérieure :
— I1 qui parle de i
— I2 qui parle de `min` et permet d'impliquer la post-condition quand $i == n-1$.
Pour la boucle intérieure :
— J1 qui parle de j.
— J2 qui parle de `min_i`. Il doit permettre d'impliquer WP$(19-21, I2)$, cependant il peut être beaucoup plus simple que cette formule (les loop assigns permettront de démontrer ce qui n'est pas modifié par la boucle).

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(g) Déterminez les pré-conditions qui permettent de démontrer ce contrat de fonction et que les gardes RTE sont satisfaites.

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(h) Démontrez la terminaison de la fonction en donnant un variant de boucle pour chaque boucle et en démontrant qu'ils décroissent à chaque tour de boucle et qu'ils sont toujours positifs (vous pouvez utiliser les calculs des questions précédentes).

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(i) Trouvez une fonction très simple (d'une instruction) qui satisfait le même contrat de fonction.

Réponse:
METTEZ VOTRE RÉPONSE ICI.

(j) Donnez une postcondition vérifiée par la fonction correcte qui n'est pas vérifiée par celle-ci. Vous le vérifierez avec frama-c, en ajoutant les invariants pertinents (un par boucle suffira).

> **Réponse:**
> METTEZ VOTRE RÉPONSE ICI.

(k) Complétez les fichiers `min_dist.c` et `min_dist.h` de manière à ce que le contrat reflète tous les points vus ici et que frama-c accepte de prouver le contrat. Vous pouvez bien évidemment vous servir de cette question pour travailler les précédentes (c'est recommandé, d'ailleurs). On attendra bien sûr que toutes les assertions RTE soient démontrée, ainsi qu'une clause `terminates \true` et une clause assigns.

Exercice 4 : Diameter

Cette question est volontairement plus ouverte et ne vous demandera rien sur papier.

Donnez une fonction et sa spécification, dont le prototype est le suivant :

```
1 int diameter(int* tab,unsigned int n);
```

Et sa spécification informelle est la suivante : la fonction renvoie la valeur minimale pour un i des valeur maximale pour les j des $\text{abs}(tab[i] - tab[j])$.

Dit autrement, c'est la plus petite des plus grandes différence à un élément du tableau.

Évidemment le code de la fonction va ressembler au cas précédent, mais il y a des différences qui compliquent un peu la tâche.

En particulier, si vous voulez la spécification la plus précise (i.e., celle que la fonction `return 0` ne vérifie pas), vous ne pouvez pas vous en sortir sans dérouler la boucle externe, et donc avoir un code plus long que nécessaire (en tous cas, je n'ai pas réussi à faire la preuve sans ça).

Une précondition peu évidente à trouver vous est fournie dans le fichier source. De même, la post-condition principale vous est donnée. Commencez par la démontrer, puis, comme dans les exercices précédents, observez que ce n'est pas suffisant et donnez et démontrez la post-condition manquante.

Vous pouvez cela dit noter les remarques, commentaires, arguments, explications que vous jugez utiles pour cet exercice dans le cadre qui suit (ou votre copie) :

> **Réponse:**
> METTEZ VOTRE RÉPONSE ICI.