université de BORDEAUX

Année Universitaire 2021/2022
Session de Printemps 2022

Parcours : Master 1 Informatique
Code UE : 4TIN801U
Épreuve : Conception Formelle
Date : Mercredi 11 mai 2022 **Heure :** 9 heures
Durée : 1 heure 30
Documents : non autorisés
Épreuve de M. Vincent Penelle

Collège Sciences et Technologies

Indiquez votre code d'anonymat :

Avertissement

- Les deux exercices sont indépendants, et dans chacun d'entre eux, certaines questions dépendent des précédentes.
- L'espace laissé pour les réponses est suffisant (sauf si vous utilisez ces feuilles comme brouillon, ce qui est fortement déconseillé).
- Le barème est donné à titre indicatif, et pourra donc être différent d'indiqué. Chaque question d'un exercice comporte un nombre d'étoiles représentant sa longueur/difficulté et donc son poids relatif envisagé dans la note. Elles n'ont pas la même signification dans les deux exercice (le premier étant plus court et facile que le second).
- Les questions bonus sont facultatives (mais peuvent rapporter des points).

Question	Points	Score
Rappels et compréhension	6	
findCloserToZero	14	
Total:	20	

Rappel:

Un triplet de Hoare est un tuple de la forme $\{\varphi\}$ `P` $\{\psi\}$ où `P` est un *programme* (dans la suite, on considérera toujours qu'il s'agit d'une suite d'instructions en C, ou d'une fonction complète), et ψ et φ sont des formule de la logique du premier ordre dont les *variables libres* (i.e., non capturées par un quantificateur) représentent des variables du programme (qui doivent exister à la position où la formule est évaluée). φ est la *pré-condition* du triplet et ψ est sa *post-condition*. Sur Frama-C, la pré-condition d'une fonction est la conjonction des clauses `requires`, et la post-condition est la conjonction des clauses `ensures`, ainsi que des clauses `assigns`. Une clause `assert` indique une post-condition du *début* de la fonction jusqu'à l'instruction après laquelle apparaître la clause `assert`.

La *vérification* d'un programme consiste à démontrer la validité d'un triplet de Hoare grâce au calcul de plus faible précondition en calculant WP(P,ψ), puis en démontrant que $\varphi \Rightarrow WP(\texttt{P}, \psi)$ est une formule valide (i.e., vraie quelque soit la valeur des variables libres).

On rappelle les règles du calcul de weakest precondition, dans laquelle `t` représente une expression arithmétique, $\texttt{e}_1$ et $\texttt{e}_2$ des expressions en C, et φ une formule de la logique du premier ordre ayant pour variables libres les variables du programme :

- WP(`skip`, φ) = φ.
- WP(`int x`, φ) = φ.

— WP(`x=t`, φ) = φ[x ← t].
— WP(`return t;`, φ) = φ[\result ← t].
— WP(e_1`;`e_2, φ) = WP(e_1, WP(e_2, φ)).
— WP(`if(t)` e_1`; else` e_2`;`,φ) = t ⇒ WP(e_1, φ) ∧ ¬ t ⇒ WP(e_2, φ).

On rappelle que $\varphi[x \leftarrow t]$ est la formule φ dans laquelle on a remplacé toutes les occurrences de x par le terme t. Par exemple $(x == 2 \vee 3 * x \geq y - x)[x \leftarrow y + x])$ est la formule $(y + x == 2 \vee 3 * x + 3 * y \geq -x)$.

On rappelle également que si la formule φ contient un \old(x), alors dans WP(P,φ), on remplace \old(x) par x. De même, \at(x,i) est remplacé par x dans WP(i-j,φ).

Pour les boucles, il est impossible de calculer la plus faible précondition directement. Il faut fournir (manuellement) un *invariant de boucle* et démontrer qu'il permet, c'est-à-dire une formule préservée par l'effet de la boucle (i.e., si la formule est vraie à l'entrée d'un tour de boucle, elle sera vraie après). Pour démontrer le contrat, il faut de plus montrer que, avec la condition de sortie de boucle, cet invariant permet de déduire la post-condition que l'on cherche à prouver, et enfin démontrer qu'il est vérifié au tout début de la boucle (il est établi).

Plus formellement, on peut voir cela comme la règle suivante :
$WP($`while(test)@I@ B;`$, \psi)) = I$, si les deux propriétés suivantes sont vraies :
— `test` $\wedge I \Rightarrow WP(B, I)$ est valide (invariant)
— ¬`test` $\wedge I \Rightarrow \psi$ est valide (implique la post-condition).

On peut découper l'invariant en plusieurs invariants qu'on prouvera indépendamment dont on prendra la conjonction (en utilisant que $\varphi \Rightarrow (\psi_1 \wedge \psi_2)$ et $(\varphi \Rightarrow \psi_1) \wedge (\varphi \Rightarrow \psi_2)$ sont équivalentes).

En Frama-C, l'invariant est la conjonction des clauses `loop invariant` (plus la clause `loop assigns`, qu'il vous sera demandé d'expliciter dans les questions).

Un variant de boucle est un terme arithmétique V à valeur entière tel que :
— V est positif au début de chaque tour de boucle (il faut donc démontrer que I ∧ t ⇒ V ≥ 0).
— Un tour de boucle diminue strictement la valeur de V (il faut donc démontrer que `test`∧ I ⇒ WP(`B`,\at(V,i) - V > 0) où i est la première ligne du corps de la boucle B).

En Frama-C, un variant de boucle est indiqué par la clause `loop variant`.

Enfin, toujours en Frama-C, lorsqu'on déclare un *prédicat* (c'est-à-dire une formule logique qui sera utilisée ailleurs – dans différents contextes) il est possible de lui spécifier des labels comme arguments qui permettront d'évaluer ses arguments qui représentent des variables du programmes à différents labels. Par exemple, il sera possible de définir un prédicat `IsEqual{L,M}(integer x) = \at(x,L) == \at(x,M)`, et de l'utiliser dans le programme, par exemple comme `assert IsEqual{Pre,Here}(y);` si on veut démontrer que la variable `y` a la même valeur à la position courante qu'avant l'appel de la fonction. C'est notamment indispensable lorsqu'on veut utiliser les valeur pointées par un même pointeur (ou tableau) à différents labels.

Exercice 1 : Rappels et compréhension (6 points)

(a) (*) Expliquez rapidement ce que décrivent les formules du premier ordre qu'on manipule dans ce cours, i.e., qu'est-ce que représentent leurs variables.

Décrivez en français la signification de la validité d'un triplet de Hoare de la forme $\{\varphi\}$ `P` $\{\psi\}$ (i.e., quand le triplet est valide, que peut-on dire sur l'effet du programme P).

(b) (***) Qu'est-ce qu'un comportement indéterminé pour une instruction C ?

Donnez deux exemples d'instructions pouvant avoir un comportement indéterminé, ainsi que l'état de mémoire le provoquant : un provoquant une erreur à l'exécution si compilé avec gcc, et un ne provoquant pas d'erreur à l'exécution, et en donnant le résultat obtenu après le comportement indéterminé si compilé avec gcc.

Est-ce qu'avec un autre compilateur respectant la norme C vous auriez nécessairement le même résultat dans ces exemples ?

(c) (**) Que doit-on faire pour vérifier qu'un programme ne provoque pas de comportements indéterminés avec le calcul de weakest precondition ? Quel module de Frama-C est responsable de cela ?

(d) (*) Quelle est la signification, en terme d'invariant, de l'expression ACSL `loop assigns x,y,*t` (on demande une explication, pas nécessairement une formule FO, mais on acceptera une formule FO) ?

En l'absence d'une telle clause `loop assigns`, que considère Frama-C ?

(e) (**) Que sont la correction partielle et la correction totale d'une fonction ?
Si frama-c valide votre spécification, laquelle a-t'on ? Que faut-il faire pour avoir l'autre ?

(f) (**) Pourquoi la présence d'un variant de boucle assure t'elle la terminaison d'une boucle ?

(g) (*) Est-il possible de déterminer la terminaison de toute boucle ? Justifiez.
(Bonus) Si la réponse est non, donnez une boucle pour laquelle personne n'a la réponse, si vous en connaissez une.

Exercice 2 : findCloserToZero **(14 points)**

```
1  int findCloserToZero(int *t, int n)
2  {
3      if (n <= 0)
4        res = 0;
5      else
6        res = t[0];
7      int i = 1;
8      while (i < n)
9      {
10         if (|t[i]| < |res|)
11             res = t[i];
12         if (res < 0 && res == -t[i])
13             res = t[i];
14         i++;
15     }
16     return res;
17 }
```

Dans cet exercice, on considère qu'on dispose d'un terme arithmétique $|t|$ qui est la valeur absolue du terme arithmétique t ($|t| = t$ si $t \geq 0$ et $|t| = -t$ si $t < 0$). On l'utilisera tel quel dans les formules.

Informellement, cette fonction va trouver la valeur de plus proche de 0 présente dans le tableau. En cas d'égalité, elle prendra la valeur positive.

Plus formellement, on considère les post-conditions suivantes :

`MinAbsVal` := $\forall k; 0 \leq k < n \Rightarrow |t[k]| \geq |\backslash\texttt{result}|$;

`PositivePriority` := $(\exists k1, k2; 0 \leq k1 < n \wedge 0 \leq k2 < n \wedge \backslash\texttt{result} == t[k1] \wedge \backslash\texttt{result} == -t[k2]) \Rightarrow \backslash\texttt{result} \geq 0$

Le but de cet exercice est de trouver une pré-condition permettant au programme de valider les post-conditions précédentes, et d'en démontrer la correction.

Dans vos calculs de weakest precondition, vous pourrez représenter des instructions par leur numéros de lignes (e.g., écrire WP(14,φ) à la place de WP(i++,φ)), et vous pourrez également utiliser les numéros de lignes pour représenter des sections du programmes, par exemple WP(10-14,φ) désignera le sous-programme commençant avant l'instruction de la ligne 10 et s'arrêtant après l'instruction de la ligne 14.

Comme nous sommes dans une preuve sur papier, nous faisons les mêmes hypothèses que dans le cours, à savoir que nous considérons que les valeurs manipulées par le programme sont de vrais entiers de $\mathbb{Z}$ et pas des entiers machine. On considèrera également qu'aucun comportement indéterminé ne peut survenir.

(a) (*) Commencez par calculer WP(16,`MinAbsVal`) et WP(16,`PositivePriority`) (comme le résultat est la simple application d'une règle, il est inutile de détailler ici).

(b) (*) Nous allons maintenant commencer à définir des invariants, et démontrer qu'ils impliquent bien notre post condition.

Donnez la clause `loop assigns` appropriée à la boucle.

(c) (**)

Déterminez trois invariants :

I1 qui est l'invariant lié à la condition de fin de boucle

IMin qui est un invariant permettant d'impliquer WP(16,`MinAbsVal`)

IPos qui est un invariant permettant d'impliquer WP(16,`PositivePriority`)

Justifiez que $\neg(i < n) \wedge I1 \wedge IMin \Rightarrow WP(16, MinAbsVal)$ et que $\neg(pos < size) \wedge I1 \wedge IPos \Rightarrow WP(16, PositivePriority)$ (la justification est commune aux deux et est très syntaxique).

On va maintenant chercher à démontrer que les formules déterminées précédemment sont des invariants.

Si on calcule WP(10-14,φ) pour une formule quelconque φ, on obtient :

$$
\begin{array}{rrll}
 & (|t[i]| < |res| \wedge t[i] < 0 \wedge t[i] == -t[i]) \Rightarrow & \varphi[i \leftarrow i+1][res \leftarrow t[i]][res \leftarrow t[i]]) & \text{(A)} \\
\wedge & (|t[i]| < |res| \wedge \neg(t[i] < 0 \wedge t[i] == -t[i])) \Rightarrow & \varphi[i \leftarrow i+1][res \leftarrow t[i]]) & \text{(B)} \\
\wedge & (|t[i]| \geq |res| \wedge res < 0 \wedge res == -t[i]) \Rightarrow & \varphi[i \leftarrow i+1][res \leftarrow t[i]]) & \text{(C)} \\
\wedge & (|t[i]| \geq |res| \wedge (\neg(res < 0 \wedge res == -t[i]))) \Rightarrow & \varphi[i \leftarrow i+1]) & \text{(D)}
\end{array}
$$

On a nommé chacune de ces implications (à droite), et on les a simplifié partiellement pour vous simplifier la tâche.

(d) (*) Simplifiez (si possible) les parties gauches des implications. Laquelle est trivialement vraie ? Dans la suite on n'utilisera pas cette implication (puisqu'elle est déjà vraie).

Dans la suite du sujet, vous traiterez chacune de ces trois implications indépendamment dans vos calculs et justifications. Je vous conseille fortement de les écrire de la manière suivante : «pour (A), $IMin$[...] donne la formule [...]. Elle est bien impliquée par IMin et a et b (et ..) car [...]».

Ce que vous avez à montrer en faisant ça, c'est une formule du genre $i < n \wedge I \Rightarrow ((|t[i]| \geq |res| \wedge res < 0 \wedge res == -t[i]) \Rightarrow \varphi[...])$ (pour (C)). Mais, comme $P \Rightarrow (Q \Rightarrow R)$ est équivalente à $(P \wedge Q) \Rightarrow R$, vous avez sacrément intérêt à considérer que tout ce qui est à gauche de l'implication la plus à droite sont des hypothèses avec lesquels vous allez démontrer φ[...] (par exemple, grâce à $IPos$ et $|t[i]| \geq |res|$, je déduis que [...]). Pour ça vous avez deux

cas types : soit les hypothèses contiennent une contradiction (donc la formule est vraie), soit les hypothèses permettent bien de démontrer $\varphi[...]$.

Vous pourrez écrire $\ll IMin[...]\gg$ sans remettre les substitutions (elles sont écrites dans cette question, inutile de le recopier partout). Quand deux substitutions donnent le même résultat (ça arrive souvent), n'écrivez la formule qu'une fois. Le message principal est que toutes ces questions sont similaires, économisez-vous du temps d'écriture. On n'attendra pas de preuve entièrement formelle, mais une justification en français convaincante. Vous pouvez cela dit, découper vos formule pour exhiber des sous-formules triviales. N'oubliez pas de simplifier $\varphi[...]$ le plus possible. Les preuves sont à peu près toutes assez syntaxiques.

(e) (*) Donnez $I1[...]$ dans les trois implications précédentes.
Justifier que $i < n \wedge I1 \Rightarrow WP(9 - 13, I1)$.

(f) (*) Donnez $IMin[...]$ dans les trois implications de la question d.
Justifier que $IMin$ implique bien les implications obtenues (vous n'avez pas besoin des autres hypothèses).

(g) (***) Donnez $IPos[...]$ dans les trois implications de la question d.
Justifiez que deux des implications obtenues sont impliquées par $IPos$. Attention, $IPos[...]$ contient elle-même une implication contenant un $\exists$ dans sa partie gauche (si ce n'est pas le cas, vous êtes en train de vous planter). Pour démontrer qu'elle est vraie, montrez soit que cette partie gauche est contradictoire avec les hypothèses que vous avez, soit que sa partie droite est vraie avec ces mêmes hypothèses.

On admettra pour le moment la dernière implication (pour les besoins de l'exercice, en réalité, il nous manque une information, que la fin de l'exercice sera consacrée à découvrir). Il est normalement assez aisé d'identifier les deux implications que l'on peut traiter à cette question (d'autant qu'elles se ressemblent plus entre elles que la troisième).

On va pour le moment admettre que l'invariant précédent en est un (c'est bien le cas, mais il nous manque un invariant supplémentaire pour le prouver – on fera trouver cet invariant en fin de sujet).

(h) (**) Calculer WP(3-7,φ) pour une formule φ quelconque.

Déduisez-en WP(3-7,I) pour chaque invariant et déduisez-en le triplet de Hoare que nous venons de démontrer (ahem). Simplifiez vos formules au maximum (et notamment justifiez les formules vraies).

(i) (**) On va maintenant montrer que la fonction termine. Déterminez un variant Var pour la boucle, et démontrez que $i < n \Rightarrow Var \geq 0$ et que $WP(10 - 14, Var - \text{at}(Var, 10) < 0)$ est vraie (pas besoin des invariants ici). Pour cette dernière question, vous pouvez comme aux questions précédente vous contenter de donner $(Var - \text{at}(Var, 8) < 0)[...]$ pour les trois

implications de la question d et justifier qu'il est vrai (cela devrait donner la même formule pour les trois implications).

(j) (*) Si on se place pour cette question en vrai C, quels comportements indéterminés peuvent survenir, et que faut-il rajouter comme précondition pour les éviter ? (on ne demande pas de la démontrer par le calcul)

(k) (*) Trouvez une fonction `fakeFindCloserToZero` très simple (une ligne) pour laquelle le même triplet sera valide. Prouvez-le en calculant WP(fakeFindCloserToZero, MinAbsVal $\wedge$ PositivePriority). On rappelle qu'un triplet $\{\varphi\}P\{\psi\}$ est valide si $WP(P, \psi) \Rightarrow \varphi$ (et qu'il est donc possible que le WP calculé ne soit pas le même que précédemment sans contredire ce qu'on veut prouver).

(l) (**) Quelle post-condition manque-t'il pour séparer ces deux fonctions ? Exprimez-la en français (moitié des points), puis en logique. Dans la suite on appellera ψ cette nouvelle post-condition.

(m) (*) Calculez WP(16,ψ) et donnez un invariant $I\psi$ permettant de l'impliquer en fin de boucle (justifiez que c'est bien le cas).

(n) (**) Démontrer que $I\psi$ est bien un invariant (même suppositions et raccourcis qu'au questions précédentes).

(o) (*) Obtient-on une précondition plus restrictive que ci-dessus ? Démontrez-le en calculant $WP(3-7, I\psi)$ et donnez le nouveau triplet de HOARE valide que l'on vient de démontrer.

(p) (bonus) (*****) Démontrez que $IPos$ est un invariant, grâce à $I\psi$. Attention, ici, il faut un peu plus réfléchir et faire différent cas pour démontrer que c'est correct.