# Spécification et preuve de programmes
# Solution du devoir 1 : Vérification déductive, logique de Hoare et logique de micro-C

Alain Giorgetti

Licence d'Informatique de l'Université de Franche-Comté 2020-21

Un barème est donné à titre indicatif.

Afin que ce devoir serve aussi d'entraînement aux examens, il est demandé que toutes les réponses soient rédigées **à la main** par l'étudiant.e, sur des feuilles au format A4 portant chacune le nom et le prénom de l'étudiant.e. Ces feuilles doivent être numérisées au format PDF, par scan ou photographie, et déposées sur Moodle dans le devoir prévu à cet effet.

## 1 Preuve propositionnelle avec micro-C (2 points)

Donner le contenu d'un fichier micro-C qui permet de démontrer avec l'interface en ligne que les formules $(p \Rightarrow q) \vee r$ et $p \Rightarrow (q \vee r)$ sont équivalentes.

(1 point pour la syntaxe, 1 point si la formule correspond à l'énoncé)

```
/*@
predicate p();
predicate q();
predicate r();

lemma f1: ((p → q) || r) ↔ (p → (q || r));
*/
```

## 2 Logique des prédicats avec micro-C (6 points)

Sachant que :
($H_1$) tout homme est un primate ;
($H_2$) les dauphins ne sont pas des primates ;
($H_3$) il y a des dauphins intelligents.
on veut aboutir à la conclusion ($C$) qu'on peut ne pas être un homme et être intelligent.

1. Les prédicats $h(x)$, $p(x)$, $d(x)$ et $i(x)$ formalisent respectivement les propositions "$x$ est un homme", "$x$ est un primate", "$x$ est un dauphin" et "$x$ est intelligent".

   Utiliser ces prédicats pour formaliser les hypothèses ($H_1$), ($H_2$) et ($H_3$) et la conclusion ($C$) du problème par des formules closes du premier ordre.

(2 points : 0,5 point par formule) Les hypothèses $(H_1)$, $(H_2)$ et $(H_3)$ et la conclusion $(C)$ du problème se formalisent respectivement par

$$\forall x \,.\, h(x) \Rightarrow p(x),$$

$$\forall x \,.\, d(x) \Rightarrow \neg p(x),$$

$$\exists x \,.\, d(x) \wedge i(x)$$

et

$$\exists x \,.\, \neg h(x) \wedge i(x).$$

2. Présenter le problème comme une seule formule du premier ordre dont on doit prouver la validité.

(1 point) Le problème peut s'exprimer sous forme de la formule du premier ordre

$$(\forall x \,.\, h(x) \Rightarrow p(x)) \wedge (\forall y \,.\, d(y) \Rightarrow \neg p(y)) \wedge (\exists z \,.\, d(z) \wedge i(z)) \Rightarrow (\exists t \,.\, \neg h(t) \wedge i(t))$$

dont on doit prouver la validité.

3. Formaliser ce problème dans la logique de micro-C.

(2 points)

```
/*@
predicate h(int x);
predicate p(int x);
predicate d(int x);
predicate i(int x);

lemma f1:
  (
    (forall x. h(x) → p(x)) &&
    (forall y. d(y) → !p(y)) &&
    (exists z. d(z) && i(z))
  ) → exists t. !h(t) && i(t);
*/
```

4. Décrire la réponse de l'interface en ligne de micro-C pour ce problème.

(1 point) L'interface en ligne de micro-C valide ce lemme.

# 3 Définitions de la valeur absolue d'un entier (4 points)

1. Donner la syntaxe micro-C pour déclarer un symbole fonctionnel `abs` tel que `abs(x)` représente la valeur absolue d'un entier `x`.

(1 point) Ligne 2 du listing 1.

2. La valeur absolue d'un entier $x$ (notée $|x|$ en mathématiques) est égale à $x$ si $x$ est positif ou nul. Sinon, elle est égale à $-x$. Avec la notation définie dans la question 1,

formaliser cette définition dans la logique de micro-C, en complétant les pointillés dans la définition suivante :

```
predicate def1(int x) = ...
```

(1 point) Ligne 4 du listing 1.

3. De même, définir un prédicat micro-C `def2` correspondant à la définition suivante : la valeur absolue d'un entier $x$ est égale à $x$ ou à $-x$, et c'est un entier positif ou nul.

   (1 point) Ligne 6 du listing 1.

4. Donner un code micro-C qui permet de vérifier avec l'interface en ligne que ces deux définitions sont équivalentes.

   (1 point) Lemme `eqdef` dans le listing 1.

```
1 /*@
2 function int abs(int x);
3
4 predicate def1(int x) = (x ≥ 0 → abs(x) == x) && (x < 0 → abs(x) == -x);
5
6 predicate def2(int x) = (abs(x) == x || abs(x) == -x) && abs(x) ≥ 0;
7
8 lemma eqdef: forall x. def1(x) ↔ def2(x);
9 */
```

Listing 1 – Solution de l'exercice 3.

## 4 Puissances d'un entier, en logique de Hoare (8 points)

1. Exprimer par un triplet de Hoare que, si $b$ est positif ou nul et si le programme

$$p := 1;\ i := 0;\ \text{while } i < b \text{ do } p := a * p; i := i + 1 \text{ od}$$

termine, alors il calcule $a^b$ dans $p$ pour tout entier $a$.

(1 point)

$$\{b \geq 0\}\ p := 1;\ i := 0;\ \text{while } i < b \text{ do } p := a * p; i := i + 1 \text{ od } \{p = a^b\}$$

2. En appliquant une par une les règles de la logique de Hoare, démontrer formellement **dans un tableau** que ce triplet de Hoare est valide.

(5 points : 1 point pour $\boxed{1}$-$\boxed{3}$ (sequence), 1 point pour $\boxed{3}$-$\boxed{5}$ (sequence), 1 point pour $\boxed{5}$-$\boxed{8}$ (consequence), 1 point pour $\boxed{9}$ (règle (while) et invariant de boucle), 1 point pour $\boxed{9}$-$\boxed{11}$ (sequence), si les justifications hors tableau sont présentes)

Dans la logique de Hoare, une preuve de validité de ce triplet de Hoare est :

| No | Triplet ou formule | Justification |
|---|---|---|
| 1 | $\{b \geq 0\}\ p := 1;\ i := 0;$ while $i < b$ do $p := a * p; i := i + 1$ od $\{p = a^b\}$ | (*seq.*) 2 3 |
| 2 | $\{b \geq 0\}\ p := 1\ \{b \geq 0 \wedge p = 1\}$ | (*assign.*) |
| 3 | $\{b \geq 0 \wedge p = 1\}\ i := 0;$ while $i < b$ do $p := a * p; i := i + 1$ od $\{p = a^b\}$ | (*seq.*) 4 5 |
| 4 | $\{b \geq 0 \wedge p = 1\}\ i := 0\ \{b \geq 0 \wedge p = 1 \wedge i = 0\}$ | (*assign.*) |
| 5 | $\{b \geq 0 \wedge p = 1 \wedge i = 0\}$ while $i < b$ do $p := a * p; i := i + 1$ od $\{p = a^b\}$ | (*conseq.*) 6 7 8 |
| 6 | $b \geq 0 \wedge p = 1 \wedge i = 0 \Rightarrow i \leq b \wedge p = a^i$ | (voir (1)) |
| 7 | $\{i \leq b \wedge p = a^i\}$ while $i < b$ do $p := a * p; i := i + 1$ od $\{i \geq b \wedge i \leq b \wedge p = a^i\}$ | (*while*) 9 |
| 8 | $i \geq b \wedge i \leq b \wedge p = a^i \Rightarrow p = a^b$ | évident |
| 9 | $\{i < b \wedge i \leq b \wedge p = a^i\}\ p := a * p; i := i + 1\ \{i \leq b \wedge p = a^i\}$ | (*seq.*) 10 11 |
| 10 | $\{i < b \wedge p = a^i\}\ p := a * p\ \{i + 1 \leq b \wedge p = a^{i+1}\}$ | (voir (2)) |
| 11 | $\{i + 1 \leq b \wedge p = a^{i+1}\}\ i := i + 1\ \{i \leq b \wedge p = a^i\}$ | (voir (3)) |

L'invariant de boucle est $i \leq b \wedge p = a^i$.
(1) L'implication

$$b \geq 0 \wedge p = 1 \wedge i = 0 \Rightarrow i \leq b \wedge p = a^i$$

est évidente : il suffit de remplacer $i$ par $0$ et $p$ par $1$ à droite de l'implication, puisque $a^0 = 1$.
(2) Le triplet

$$\{i < b \wedge p = a^i\}\ p := a * p\ \{i + 1 \leq b \wedge p = a^{i+1}\}$$

se justifie comme suit :

$$\overline{\{i + 1 \leq b \wedge a * p = a^{i+1}\}\ p := a * p\ \{i + 1 \leq b \wedge p = a^{i+1}\}}$$

selon la règle (*assignment*). Or $i + 1 \leq b$ est équivalent à $i < b$ et $a * p = a^{i+1}$ se simplifie en $p = a^i$.
(3) Le triplet

$$\{i + 1 \leq b \wedge p = a^{i+1}\}\ i := i + 1\ \{i \leq b \wedge p = a^i\}$$

est une instance de la règle (*assignment*).

3. La déclaration

```
function int power(int a, int b);
```

introduit un symbole fonctionnel `power` d'arité 2. Reproduire et compléter les axiomes

```
axiom power0 : forall x. power(x,0) == ...;
axiom powerS : forall x. forall n. ... → power(x,n+1) == ... power(x,n);
```

afin que `power(x,y)` formalise $x^y$ pour tout entier `x` et tout entier `y` positif ou nul.

(1 point)

```
axiom power0 : forall x. power(x,0) == 1;
axiom powerS : forall x. forall n. n ≥ 0 → power(x,n+1) == x * power(x,n);
```

4. Exprimer sous forme de lemmes micro-C les propriétés $x^1 = x$ et $a^{b+c} = (a^b) * (a^c)$ pour tous les entiers $b$ et $c$ positifs ou nuls.

   Essayez de démontrer ces deux lemmes avec l'interface en ligne de micro-C. Justifiez clairement et simplement le résultat obtenu.

   (1 point)

```
lemma power1 : forall x. power(x,1) == x;
lemma power_sum : forall a. forall b. forall c. b ≥ 0 && c ≥ 0
  → power(a,b+c) == power(a,b) * power(a,c);
```

   Le premier lemme est facile à démontrer, mais il est possible que le prouveur de l'interface en ligne de micro-C échoue dans cette démonstration.
   Le second lemme n'est pas démontré, mais c'est compréhensible : sa démonstration à partir des axiomes requiert une preuve par récurrence, et peu de prouveurs automatiques savent faire des preuves par récurrence.