Examen d'Approches formelles pour la vérification de programmes

Master CNS

jeudi 5 novembre 2020

Durée : 3h.
Documents autorisés sauf livres. Aucun appareil électronique n'est autorisé.
Ce sujet comporte 4 exercices indépendants, qui peuvent être traités dans l'ordre voulu.
Il contient 4 pages.
Les questions précédées par le symbole ($\star$) sont des questions bonus qui pourront être traitées à la fin.
L'exercice 1 doit être rédigé sur une copie séparée des autres exercices.

Exercice 1 : EACL - Vérification à runtime

On s'intéresse à la fonction C find dont l'entête est la suivante :

```
int find(int *a, int n, int val)
```

Son contrat ACSL précise, entre autres, les clauses suivantes :

```
requires n > 0;
requires \valid(a + (0..n-1));
assigns \nothing;
```

1. Expliquez en quelques mots ce que chacune de ces deux clauses signifie.
2. Complétez le contrat de manière à spécifier que la fonction retourne l'index d'**une occurrence** de `val` dans le tableau `a` de longueur `n` si `val` est dans le tableau, `n` sinon. Vous pourrez utiliser le prédicat `HasValue(a, n, val)` pour exprimer que la valeur `val` apparaît dans le tableau `a` de longueur `n`.
   On appellera ce contrat le contrat C_{one}.
3. Complétez le contrat de manière à spécifier que la fonction retourne l'index de la **première occurrence** de `val` dans le tableau `a` de longueur `n` si `val` est dans le tableau, `n` sinon. Ici aussi, vous pourrez utiliser le prédicat `HasValue(a, n, val)` pour exprimer que la valeur `val` apparaît dans le tableau `a` de longueur `n`.
   On appellera ce contrat le contrat C_{first}.
4. Les deux contrats C_{one} et C_{first} sont-ils exécutables ? Justifiez votre réponse.
5. Regardons maintenant l'implémentation de la fonction `find` donnée ci-dessous.

```
int find(int *a, int n, int val){
   for (int i = n-1; i >= 0; i--) {
     if (a[i] == val) {
     return i;
   }
   }
   return n;
}
```

Soit le programme de test ci-dessous. Que se passe-t-il si on exécute cette fonction (sans utiliser le plugin EASCL) ?

```
int main(void) {
    int tab[4] = {1, 2, 0, 0};
    printf("%i\n", find(tab, 4, 0));
    return 0;
}
```

6. Que se passe-t-il si on exécute le programme de test précédent en utilisant le plugin EACSL et le contrat C_{one} ?
7. Que se passe-t-il si on exécute le programme de test précédent en utilisant le plugin EACSL et le contrat C_{first} ?
8. ($\star$) Dans le cadre du contrat C_{one}, donnez le code, généré par le plugin ACSL, associé à la postcondition dans le cas où `val` est présent dans le tableau. Vous écrirez ce code sous la forme d'un pseudo-code.

Exercice 2 : Preuve

Soient les formules suivantes :

(a) $(a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c))$

(b) $(a \vee b) \Rightarrow (b \vee a)$

(c) $(\forall X.\ p(X)) \Rightarrow (\forall X.\ p(X))$

(d) $(\exists X.\ (p(X) \wedge q(X))) \Rightarrow ((\exists X.\ p(X)) \wedge (\exists X.\ q(X))$

Par chacune d'entre elles :

1. Calculer les clauses correspondant à la négation de la formule.
2. Donner une preuve par résolution. (Cf. fig. 2 page 4.)
3. Donner une preuve en déduction naturelle. (Cf. fig. 1 page 4.)

Par ailleurs,

4. ($\star$) Donner deux programmes OCaml dont le type est celui associé respectivement à la formule (a) et à la formule (b) via la correspondance de Curry–Howard–De Bruijn.

Exercice 3 : Conditions de vérification

1. Calculer les conditions de vérification pour les programmes et post-conditions suivantes :
   a) VC(`{a = 42; b = a - b; a = b - a;}`, $a = b$)
   b) VC(`if (x > 100) x = 100; else ;`, $x > 42$)
2. Montrez que les pré-conditions suivantes sont valides pour les programmes et post-conditions suivantes :
   a) Pré-condition : $a > 0$ Programme `b = a + 2;` Post-condition : $b \geq 0$
   b) Pré-condition : $a = b$ Programme `if (a > b) m = a; else m = b;` Post-condition $m = b$

Exercice 4 : Preuve de programme

On considère le programme suivant :

```
int sum(int n) {
  int i, s;
  i = 0;
  s = 0;
  while (i != n) {
    s = s + i;
    i = i + 1;
  }
  return s;
}
```

On cherche à montrer que si n est positif ou nul, alors `sum(`n`)` retourne $\frac{n(n-1)}{2}$.

1. Donner les spécifications en ACSL de la fonction.
2. Calculer la condition de vérification pour le corps de la boucle `while` avec la post-condition $s = \frac{i(i-1)}{2}$.
3. Quelles sont les variables modifiées par le corps de la boucle ?
4. En déduire la condition de vérification de la boucle en entier, avec comme invariant de boucle $s = \frac{i(i-1)}{2}$ et comme post-condition $s = \frac{n(n-1)}{2}$.
5. En déduire la condition de vérification du corps de la fonction en entier, avec la même post-condition.
6. Quelle famille de prouveurs automatiques serait a priori capable de la démontrer ?
7. Déduire de la condition de vérification qu'avec la précondition $n \geq 0$, le programme est correct.
8. ($\star$) Que se passe-t-il si on change la troisième ligne en `i = 1;` ? Peut-on toujours prouver la correction du programme ? On étudiera en particulier le cas où $n = 0$.

$$\frown \frac{}{\Gamma, A \vdash A} \qquad \bot\text{-e}\ \frac{\Gamma \vdash \bot}{\Gamma \vdash A}$$

$$\wedge\text{-e}_g\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \wedge\text{-e}_d\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \qquad \wedge\text{-i}\ \frac{A \qquad B}{A \wedge B}$$

$$\Rightarrow\text{-e}\ \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \qquad \Rightarrow\text{-i}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

$$\forall\text{-e}\ \frac{\Gamma \vdash \forall x.\ A[x]}{\Gamma \vdash A[t]} \qquad \forall\text{-i}\ \frac{\Gamma \vdash A[x]}{\Gamma \vdash \forall x.\ A[x]}\ x \text{ non libre dans } \Gamma$$

$$\neg\text{-e}\ \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot} \qquad \neg\text{-i}\ \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}$$

$$\vee\text{-e}\ \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \qquad \vee\text{-i}_g\ \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad \vee\text{-i}_d\ \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$\exists\text{-e}\ \frac{\Gamma \vdash \exists x.\ A[x] \qquad \Gamma, A[x] \vdash C}{\Gamma \vdash C}\ x \text{ non libre dans } \Gamma, C \qquad \exists\text{-i}\ \frac{\Gamma \vdash A[t]}{\Gamma \vdash \exists x.\ A[x]}$$

Vous pouvez également utiliser ces deux règles dérivées :

$$\vee\text{-d}\ \frac{\Gamma, A \vee B, A \vdash C \qquad \Gamma, A \vee B, B \vdash C}{\Gamma, A \vee B \vdash C}$$

$$\exists\text{-d}\ \frac{\Gamma, \exists x.\ A[x], A[x] \vdash C}{\Gamma, \exists x.\ A[x] \vdash C}\ x \text{ non libre dans } \Gamma, C$$

FIGURE 1 – Règles de la déduction naturelle

$$\text{Resolution}\ \frac{P \vee C \qquad \neg Q \vee D}{\sigma(C \vee D)} \qquad \text{Factoring}\ \frac{P \vee Q \vee C}{\sigma(P \vee C)}$$

σ est l'unificateur le plus général de P et Q

FIGURE 2 – Règles de la résolution