Validation d'algorithmes
L3 MIAGE, UGA
2019/2020

raphael.charrondiere@inria.fr

# Examen session 2

---

**Modalités :** Selon les règles de l'université, un délai maximal de 24h est autorisé entre le dépôt du sujet et le rendu, passé ce délai, même à quelques minutes près, je n'accepterai pas votre copie. Vous avez reçu ce sujet par e-mail, et vous devez le rendre par e-mail. Vu la situation particulière, l'anonymisation des copies a été levée.

Vous n'avez pas le droit de vous faire aider pour cette épreuve, cet examen est **individuel**. Vous pouvez utiliser tout document du cours.

Pour chaque question, une indication sur le temps maximal à investir est indiquée, l'épreuve ne devrait pas vous prendre plus de 2 heures.

**Important :**

- Pensez à vous relire.
- Soyez clair, soignez la présentation. Un texte bref mais précis est plus utile qu'un discours long et confus.
- Ne jouez pas avec le feu, n'attendez pas la dernière minute pour rendre votre copie, laissez-vous au moins une heure de marge, il ne faudrait pas qu'un incident technique survienne à la dernière minute.
- Ne perdez pas de temps, faites votre devoir sur papier, puis scannez ou photographiez votre copie, faites attention à ce que l'image soit nette.
- Si vous trichez, ça se verra, ne jouez pas avec le feu.

---

## Rappel de cours

On rappelle d'abord les règles d'inférence pour la sémantique à grands pas de IMP.

$$\frac{}{\sigma, \textbf{skip} \Downarrow \sigma} \qquad \frac{\sigma, c_1 \Downarrow \sigma' \quad \sigma', c_2 \Downarrow \sigma''}{\sigma, c_1 \,\textbf{;} c_2 \Downarrow \sigma''} \qquad \frac{\sigma, a \rightsquigarrow k}{\sigma, x \,\textbf{:=} a \Downarrow \sigma'}\ \sigma' = \sigma[k/x]$$

$$\frac{\sigma, b \rightsquigarrow true \quad \sigma, c_1 \Downarrow \sigma'}{\sigma, \textbf{if b then } c_1 \textbf{ else } c_2 \Downarrow \sigma'} \qquad \frac{\sigma, b \rightsquigarrow false \quad \sigma, c_2 \Downarrow \sigma'}{\sigma, \textbf{if b then } c_1 \textbf{ else } c_2 \Downarrow \sigma'}$$

$$\frac{\sigma, b \rightsquigarrow false}{\sigma, \textbf{while b do } c \Downarrow \sigma} \qquad \frac{\sigma, b \rightsquigarrow true \quad \sigma, c \Downarrow \sigma' \quad \sigma', \textbf{while b do } c \Downarrow \sigma''}{\sigma, \textbf{while b do } c \Downarrow \sigma''}$$

Puis les règles d'inférence pour la sémantique à petits pas de IMP.

$$\frac{\sigma, a \rightsquigarrow k}{\sigma, x \,\textbf{:=} a \to \sigma', \textbf{skip}}\ \sigma' = \sigma[k/x] \qquad \frac{\sigma, c_1 \to \sigma', c_1'}{\sigma, c_1 \,\textbf{;} c_2 \to \sigma', c_1' \,\textbf{;} c_2} \qquad \frac{}{\sigma, \textbf{skip} \,\textbf{;} c_2 \to \sigma, c_2}$$

$$\frac{\sigma, b \rightsquigarrow true}{\sigma, \textbf{if b then } c_1 \textbf{ else } c_2 \to \sigma, c_1} \qquad \frac{\sigma, b \rightsquigarrow false}{\sigma, \textbf{if b then } c_1 \textbf{ else } c_2 \to \sigma, c_2}$$

$$\frac{\sigma, b \rightsquigarrow false}{\sigma, \textbf{while b do } c \to \sigma, \textbf{skip}} \qquad \frac{\sigma, b \rightsquigarrow true}{\sigma, \textbf{while b do } c \to \sigma, c \,\textbf{;}\, \textbf{while b do } c}$$

On rappelle la syntaxe des expressions algébriques :

- AB : A puis B
- A|B : A ou B
- A* : A plusieurs fois (potentiellement zéro fois)
- A+ : A une fois ou plus, soit AA*
- $\epsilon$ : l'expression vide (si besoin)
- A ? : A zéro ou une fois, soit (A|$\epsilon$)
- (A) : utilisez des parenthèses, les opérateurs ne se rapportent qu'au dernier élément, exemple AB* veut dire A suivi de plusieurs B, mais (AB)* veut dire AB répété plusieurs fois.

---

# **Partie 1:** Sémantique opérationnelle

**Question 1.1 :** Restitution de connaissances *(15 minutes)*
Répondez à chacune des questions en détaillant votre réponse.

a - Expliquez ce qu'est un arbre de dérivation.

b - Expliquez la différence grands pas, petits pas.

c - Peut-on ajouter des règles d'inférences sans changer la sémantique ?

**Question 1.2 :** Règles d'inférences *(24 minutes)*
Pour chacune des commandes présentées si dessous, donnez les règles d'inférence correspondantes pour la sémantique à grands pas **ET** à petits pas

`c` est une commande IMP, `x` est une variable, `b` est une condition (expression booléenne), `i` et `j` sont des expressions arithmétiques.

a - `do c while b`. Cette commande exécute l'instruction `c` puis recommence tant que `b` est vrai.

b - `x:=b?i:j`. Cette commande évalue la condition `b` puis attribue à `x` la valeur `i` si `b` est vraie, `j` sinon.

c - `for x=i to j do c`. Cette commande initialise `x` à `i` et tant que `x` est différent de `j`, elle exécute `c` puis incrémente `i` de 1.

**Attention piège :** pensez que `i` et `j` sont des expressions arithmétiques et non des entiers.

**Question 1.3 :** Arbres de dérivation *(20 minutes)*

Soit l'état mémoire $\sigma_1\,(i \mapsto 3)$ et la commande
$k := 4$; $r := 1$; **while** i<k **do** $(r := r * i;\ i := i + 1)$.
Calculez l'état final $\sigma'$ et donnez l'arbre de dérivation de $\sigma, C \Downarrow \sigma'$.

# **Partie 2:** Logique de Hoare

**Question 2.1 :** Annotez un programme *(20 minutes)*
On veut annoter un programme avec la logique de Hoare pour prouver son fonctionnement, ci-dessous la structure d'un programme et de sa preuve a été écrit, complétez-la.

$$
\begin{array}{l}
\{N \geqslant 0\} \implies \{\dots\} \\
R := 0 \\
\{\dots\} \\
X := 0 \\
\{\dots\} \\
\textbf{while}(X \geqslant N) \\
\quad \{\dots\} \implies \{\dots\} \\
\quad R := R + X \\
\quad \{\dots\} \\
\quad X := X + 1 \\
\quad \{\dots\} \\
\{\dots\} \implies \left\{ R := \dfrac{N(N+1)}{2} \right\}
\end{array}
$$

## **Partie 3:** Boite blanche

La fonction suivante est la méthode d'une classe écrite en C++. Vous n'avez pas besoin de la comprendre pour répondre aux questions. Il s'agit ici, de chercher un ensemble de tests à effectuer.

```
1  Motif Factory::computeMotif(size_t id)
2  {
3    if(renders.find(id)!=renders.end())
4      return renders[id];
5    int depth=0;
6    std::stack<Action*> stack;
7    stack.push(mng->get(id));
8    Motif mwork;
9    while(true)
10   {
11     depth++;
12     Action* da = stack.top();
13     if(da==nullptr)break;
14     if(renders.find(id)!=renders.end())
15     {
16       mwork=renders[id];
17       break;
18     }
19
20     stack.push(da->Prev());
21   }
22   if(stack.top()==nullptr)
23     stack.pop();
24   while(!stack.empty())
25   {
26     stack.top()->apply(mwork);
27     stack.pop();
28   }
29   if(depth>=cacheDepth3)
30     renders[id]=mwork;
31   return mwork;
32 }
```

**Question 3.1 :** Graphe de flot de contrôle *(20 minutes)*
Veuillez tracer le graphe de flot de contrôle correspondant à la fonction.

*Indication 1 :* On considère les appels de fonctions comme des simples instructions.

*Indication 2 :* Pour les nœuds de commandes **uniquement**, vous pouvez écrire seulement les numéros de ligne. Pour tout nœud de condition, **recopiez** clairement la condition.

**Question 3.2 :** Chemins *(5 minutes)*
Commencez par donner des lettres aux nœuds de votre graphe pour les identifier, puis indiquez

l'expression algébrique de tout les chemins du graphe. *Vous pouvez vous référer au rappel en début de sujet pour la syntaxe des expressions régulières.*

**Question 3.3 :** Tests *(15 minutes)*

En le **justifiant**, pour chaque critère, donnez un ensemble de chemins couvrant celui-ci :

a - tout les nœuds.

b - tout les arcs.

c - base de chemins indépendants.

# **Partie 4:** Boite noire

---

On rappelle que les tests en boite noire sont des tests qui ne considèrent pas le code, d'ailleurs celui n'est peut-être pas accessible au testeur.

On suppose avoir une structure de données multi-ensemble d'entiers. Elle représente un ensemble qui contient des entiers, mais un entier peut être présent plusieurs fois dans cet ensemble. Par exemple l'ensemble peut contenir une fois 1, deux fois 2 et trois fois 3, on l'écrit par $\{1, 2, 2, 3, 3, 3\}$.

Cette structure contient une fonction d'affichage que l'on va supposer correcte, le testeur l'appellera pour voir le contenu de la structure de données. Aussi, on supposera avoir à disposition une fonction qui crée un multi-ensemble vide.

**Question 4.1 :** Ajout d'élément *(10 minutes)*

On voudrait vérifier le fonctionnement de la fonction `ajouterElement` qui ajoute un élément (donc un entier) à la structure, proposez un ensemble de tests en justifiant (expliquant) votre choix.

*Indice :* S'il y a besoin d'un ensemble précis, il suffit de partir de l'ensemble vide, et d'ajouter un à un les éléments.

**Question 4.2 :** Retrait d'élément *(5 minutes)*

On voudrait tester la fonction `retraitElement` qui retire un élément du multi-ensemble, proposez un ensemble de tests, justifiez votre proposition.