Documents : interdits **Durée :** 1h30 / t.t. 2h

Exercice 1 Analyse de vie (/7pts) Soit l'extrait de code trois adresses suivant, où l'on considère que seules les variables r1 et r3 sont utilisées dans la suite du programme.

```
        c=r3
        a=r1
        b=r2
        d=0
        e=a
F :
        d=d+b
        e=e-1
        if(e<0) then goto F
        r1=d
        r3=c
```

1. Représentez le flot de contrôle de ce programme, et étiquetez chaque instruction par l'ensemble des variables qui sont vivantes à la fin de cette instruction.
2. Représentez le graphe d'interférences
3. Indiquez les préférences, en justifiant, et en expliquant leur utilité.
4. Combien de registres au minimum sont nécessaires à l'allocation des variables (si on s'interdit d'utiliser la pile) ? Justifiez.

Exercice 2 Coloration (/6pts) Considérez l'algorithme suivant (étudié en cours) :

```
1  Fonction ColorGraph(G, k) :
2      if G est vide then
3          Ne rien faire
4      else
5          if il existe v tel que deg(v) < k then
6              ColorGraph(G\{v}, k)
7              Attribuer à v une couleur disponible
8          else
9              Choisir v dans G
10             ColorGraph(G\{v}, k)
11             Attribuer π à v
```

1. À quoi correspond π ?
2. Pourquoi restera-t-il forcément une couleur pour v si la condition est respectée ?
3. Comment choisir v ? Justifiez votre réponse.
4. Donnez un exemple où l'algorithme donne une solution mauvaise. Expliquez comment il peut simplement être amélioré.
5. Expliquez en quelques phrases en quoi consistent les algorithmes avec fusion.

Exercice 3 Tableaux (/8pts)

On considère un langage impératif simple, sans fonctions, tels que les langages `plk` et `gazo` vus en TP. On souhaite y rajouter la gestion des tableaux. Comme dans les langages standards, les tableaux ont une taille fixée, et leurs éléments sont du même type. On souhaite autoriser des tableaux de tableaux.

— On instancie un nouveau tableau en précisant sa taille et son type (exemple : `new int[10]`)

— On accède aux éléments via des indices entiers (exemples : `t[0], t[0][0]`)

— On affecte les éléments comme d'habitude (exemple : `t[0]=true`)

1. Écrivez les ajouts et modifications que vous proposez à la grammaire existante (précisez bien quelles sont les nouvelles expressions et les nouvelles instructions).
2. Présentez les règles de typage pour les nouvelles constructions syntaxiques.
3. On considère une sémantique opérationnelle à petit pas telle que celles vues en cours et TD (sous forme de réductions $(\sigma, p) \to (\sigma', p')$). On rappelle que l'environnement σ = {x↦ v1, y↦ v2... } associe chaque variable a une valeur. Les règles sont définies pour les expressions et pour les instructions, comme rappelé en bas de page. Rajoutez les règles nécessaires pour la gestion des tableaux. (Les valeurs de type tableau peuvent être représentées schématiquement (cases du tableau)).
4. Pour motrer que vos règles sont cohérentes, montrez que le programme suivant :
   `t = new int[1]; i = 0; while(i<1){t[i]=5+i; i=i+1;} ; res=(t[0]);`
   se réduit en `(`σ`, skip)` où σ(`res`) contient la valeur attendue. On suppose que `t, i` et `res` sont déjà déclarés. Quand vous utilisez le fait qu'une expression se réduit en une certaine valeur, indiquez-le toujours avec l'évaluation nécessaire.
5. Expliquez en quoi l'inférence ou la vérification du typage est indispensable à une correcte gestion des tableaux en mémoire.

$$\frac{}{(\sigma, \mathtt{n}) \to n}\ n \in \mathbb{N} \qquad \frac{}{(\sigma, \mathtt{x}) \to \sigma(\mathtt{x})} \qquad \frac{(\sigma, \mathtt{e}) \to v_1 \qquad (\sigma, \mathtt{e'}) \to v_2}{(\sigma, \mathtt{e + e'}) \to v_1 + v_2} \qquad \frac{(\sigma, \mathtt{e}) \to v_1 \qquad (\sigma, \mathtt{e'}) \to v_2}{(\sigma, \mathtt{e < e'}) \to \text{true si } v_1 < v_2, \text{ sinon false}}$$

— $(\sigma,$ `if e then c else c'`$) \Rightarrow (\sigma,$ `c`$)$ si $e \to$ true (ift)

— $(\sigma,$ `if e then c else c'`$) \Rightarrow (\sigma,$ `c'`$)$ si $e \to$ false (iff)

— $(\sigma, \text{while } e \text{ do } c) \Rightarrow (\sigma, \text{if } e \text{ then } \{c; \text{while } e \text{ do } c\} \text{ else skip})$ (while)

— $(\sigma, \text{skip} ; \text{c}) \Rightarrow (\sigma, c)$ (skip)

— $(\sigma, x = e; c) \Rightarrow (\sigma[x \mapsto v], c)$ si $e \to v$ (aff)

— $(\sigma, c_1; c_2) \Rightarrow (\sigma', c_1'; c_2)$ si $(\sigma, c_1) \Rightarrow (\sigma', c_1')$ (seq)