LOG3210 - Élément de langages et compilateur
TP5 : Aide

Ettore Merlo – Professeur
Doriane Olewicki – Chargée de laboratoire

Hiver 2020

1 Etapes

1. Traduire le code intermédiaire en code machine (CODE, liste de MachLine).

   Exemple:

   - a = b * c : [MUL, @a, @b, @c]. Attention que si b et c n'existent pas avant la ligne, il faut les charger de la mémoire ([LD, @b, b] et [LD, @c, c]).
   - a = - b : [SUB, @a, #0, @b]. Attention que si b n'existe pas avant la ligne, il faut le charger de la mémoire ([LD, @b, b]).
   - a = b : [ADD, @a, #0, @b]. Attention que si b n'existe pas avant la ligne, il faut le charger de la mémoire ([LD, @b, b]).

2. Définissez les ensembles DEF, REF, PRED et SUCC pour toutes les lignes.

3. Calculez les ensembles Life_IN et Life_OUT. La dernière ligne devra contenir les variables dans l'expression return, moins les variables remises en mémoires (via l'expression ST)

   Rappel de l'algorithme :

```
forall (node in nodeSet) {
   IN[node] = {}
   OUT[node] = {}
}

workList = {}
workList.push(stop_node);

while(!workList.empty()) {
   node = workList.pop();

   for (succNode in successors(node)) {
      OUT[node] = OUT[node] union IN[succNode];
   }

   OLD_IN = IN[node];
   IN[node] = (OUT[node] - DEF[node]) union REF[node];

   if(IN[node] != OLD_IN) {
```

```
        for(predNode in predecessors(node)) {
            workList.push(predNode);
        }
    }
}
```

Variation possible : plutôt que d'utiliser les PRED et SUCC, baser vous sur les numéros de lignes. Si vous êtes à la ligne "n", sont successeurs est "n+1" et sont prédécesseurs et "n-1".

4. Calculez les ensembles Next_IN et Next_OUT.

   Algorithme:

```
forall (node in nodeSet) {
   IN[node] = {}
   OUT[node] = {}
}

workList = {}
workList.push(stop_node);

while(!workList.empty()) {
   node = workList.pop();

   for (succNode in successors(node)) {
      OUT[node] = OUT[node] union IN[succNode];
   }

   OLD_IN = IN[node];
   for ((v,n) in OUT[node]) {
      if (v not in DEF[node]) {
         IN[node] = IN[node] union {(v,n)}
      }
   }

   for (v in REF[node]) {
      IN[node] = IN[node] union {(v, current_line_number)}
   }

   if(IN[node] != OLD_IN) {
      for(predNode in predecessors(node)) {
         workList.push(predNode);
      }
   }
}
```

   N.B.: dans l'expression "(v,n)", "v" est une variable et "n" un numéro de ligne d'utilisation. "current_line_number" est le numéro de la ligne de code qu'on est entrain de parcourir ("node").

5. Générez le graphe interférence du bloc de code. Le graphe d'interférence est un graphe non-dirigé dont les nœuds et arrêtes sont définis comme suit:

   - Noeuds: pour chaque pointeur d'identifiants ("@*something*"), on crée un noeud.

- Arrêtes: Next_OUT, ajouter des arrêtes entre chaque pointeur présent dans le set. Exemple:

```
MUL @a @a @c (Next_OUT : [a:1, b:2, c:1, d:3])
=> 4 noeuds  (@a, @b, @c, @d)
=> 6 arretes (@a–@b, @a–@c, @a–@d, @b–@c, @b–@d, @c–@d)
```

6. Coloration de graphe. Assignation de registre en utilisation la coloration de graphe. Une fois votre graphe interférence généré, vous voulez "colorer" les noeuds (c'est-à-dire donner une numéro de registre à chaque pointeur, ex: "@a" devient "R0"). Dans un premier temps implémentez une coloration sans limite de registre (la limite est tout de même à 256 donc garder vos morceaux de code petit).

   (Sans contrainte, vous n'aurez pas besoin de do_spill() en commençant.)

```
G: Graph d interference
REG: nombre de registres disponibles
Stack: stack vide
ColorMap: hashmap  (noeud-couleur)

while(!G.empty()) {
    // Choisir le noeud avec le nombre de voisin le plus proche et
    // inferieur a REG. Enlever le noeud et ses arretes dans G.
    node = G.getNode()

    if (no node find) {
        do_spill(node)
        stop // arreter la coloration
    }

    Stack.push(node)
}

while(!Stack.empty())  {
    node = Stack.pop()

    // remet le node dans le Graph G avec ses arretes par rapports aux
    // noeuds deja replaces
    G.putback(node)

    color = 0
    while(color in [ColorMap.get(neigh) for neigh in node.neighbourgs]) {
        color++
    }
    ColorMap.put(node,  color)
}
```

7. Maintenant, gérons en plus la limitation de registre. Le nombre de registres limite est donné au début des fichiers tests et est déjà stocké dans la variable "REG" du visiteurs.

   ATTENTION: un seul spill par noeud est possible !

   Un problème arrive à l'étape (**??**) s'il n'y a plus de nœuds ayant moins de k voisins lors de l'empilage de la Stack. Alors, il faut arrêter l'empilage et "do_spill()" le nœud, ensuite, recommencer le coloriage.

```
CODE: Array de MachLine
```

```
node: registre a spill

first: numero de ligne de la premiere utilisation de node dans une
expression "OP" (pas "ST" ni "LD")


if (node is modified) {
   // Ajouter une ligne de code machine pour sauver en memoire le node
   // apres l'utilisation a la ligne first.
   CODE.add(first+1, MachLine("ST", node))
}

if (!CODE.get(first).NEXT_USE.get(node).empty()) {
   // Si la variable liee au node est utilisee plus tard, charger de la
   // memoire dans un nouveau noeud+"!" (renomme)
   CODE.add(node.NEXT_USE.get(0), MachLine("LD", node+"!"))

   for (int i in [node.NEXT_USE.get(0), CODE.size-1]) {
      if (CODE.get(i) is ST_statement) {
         CODE.remove(i)
      }
      else if (CODE.contains(node)) {
         CODE.replace(node, node+"!")
      }
   }
}
```

8. Réduction de code. Si des expressions dans votre code machine final sont triviales, vous pouvez les enlever. Exemple: la ligne "ADD R0, #0, R0" peut-être supprimée. Si vous faites des réductions complémentaires, veuillez le préciser dans le rapport.

9. Affichage de l'output. Vous devez écrire dans "m_writer" vos résultats sous le format suivant:

```
   *code machine Line0*
   // ===== LINE 0 =====
   // Life_IN = [...]
   // Life_OUT = [...]
   // Next_IN = [...]
   // Next_OUT = [...]
   *code machine Line1*
   // ===== LINE 1 =====
   (etc.)
```

   Un exemple vous est donné dans le dossier test-suite pour la version "full" mais à vous de trouver le résultat pour 3 registres et 5 registres. Une fonction d'impression vous est aussi fournie.

10. Testez le code de Fibonacci (voir énoncé pour lancer les scripts).

11. Générez des tests supplémentaires et justifiez les (voir énoncé).