Documents : interdits **Durée :** 1h / t.t. 1h20

Questions de cours

1. Le passage de paramètres aux fonctions (tel que vu en cours et implémenté en TP spécifiquement) est-il par nom, par valeur ou par référence ? En quoi cela peut-il produire du code plus efficace ?
2. Donnez un exemple de construction qui apparaît dans l'ast mais pas dans la représentation intermédiaire.
3. Quelle est la différence entre le code trois adresses et la représentation intermédiaire vue en cours ?

Fonctions

Considérez un langage impératif simple tel que ceux vus en cours et en TP, ainsi que les méthodes de compilation étudiées pour ce langage (pas d'allocation de registres pour les variables : utilisation de la pile uniquement). Votre client Jean-Jacques vous demande d'intervenir sur ce code :

1. Jean-Jacques n'est pas très à l'aise avec la récursion et souhaite simplifier le langage pour que les fonctions récursives n'y apparaissent pas. Pouvez-vous l'aider ? Expliquez précisément ce que vous changeriez dans le mécanisme du compilateur vu en cours/TP pour implémenter simplement cette interdiction.
2. Le client n'y avait pas pensé, mais il vous faut également empêcher la récursion indirecte (f appelle g, qui appelle $f \ldots$). À votre avis, comment peut-on l'empêcher ? Expliquez à quel stade du compilateur elle peut être détectée pour produire une erreur.

3 Jean-Jacques se souvient de ses cours de master, et pense qu'avec le langage simplifié on peut prédire à peu près l'espace mémoire utilisé par le code produit. Il a raison. Il vous demande donc de fournir un code (pseudocode, visiteur,...) qui analyse le programme source et estime la taille de la pile nécessaire à un code donné.

4 Le client est satisfait, mais il souhaite pouvoir déployer son langage pour de nouvelles applications. Vous lui expliquez que vous ne pouvez pas le développer sans remettre des fonctions récursives. Sa réponse : "J'ai des contraintes matérielles : faites ce que vous voulez, tant que l'on peut continuer à prévoir la taille de la pile pour chaque programme". Comment allez-vous vous en sortir ? Expliquez votre stratégie.

5 Pourrez-vous autoriser des fonctions avec un nombre d'arguments quelconques, ou serez-vous obligé de les limiter ? Expliquez précisément pourquoi.

6 Votre réponse aux deux questions précédente implique une certaine mise en place pour la gestion des fonctions. Expliquez comment ajuster votre compilateur (en se basant sur la structure étudiée en cours/tp) pour implémenter cette gestion en respectant les contraintes de Jean-Jacques.

Il est attendu une explication super précise et détaillée, que vous pouvez illustrer avec des exemples. Vous devez situer tout ce qui doit être modifié dans les mécanismes de production de code.

Traduction vers le haut et vers le bas

On considère le langage assembleur MIPS32, dont certaines instructions sont rappelées dans le tableau ci-dessous. On rappelle également que $sp est le pointeur de sommet de pile, que les adresses de la pile vont décroissant, que les mots sont de taille 4 (4 octets=32bits).

mv r1 r2	r1 ← r2 (registre à registre)
li r1 42	r1 ← 42 (entier à registre)
sw r1 adr	adr ← r1 (registre à adresse)
lw r1 adr	r1 ← adr (adresse à registre)
j L	saute au label d'instruction L
jal L	saute au label L, et enregistre l'adresse de la prochaine instruction dans $ra.
op r1 r2 r3	r1 ← r2 op r3 (op= add, sub, mul,)
beq r1 r2 L	← saute à L si r1=r2 (sinon continue à l'instruction suivante)
syscall	← appel système lisant $v0. (10=>exit, 1=>impression de l'entier stocké en $a0.

Considérez la représentation intermédiare suivante :

```
L0
 CJump ((reg0 == reg1), L6, L7)
 L6
 reg2 := 0
 goto L1
 goto L8
 L7
 reg5 := call L0[(reg0 + 1), reg1]
 reg2 := (reg0 + reg5)
 goto L1
 L8
L1
L2
 reg3 := 42
 goto L3
L3
L4
 reg6 := 5
 reg8 := call L2[]
 reg7 := reg8
 reg11 := call L0[reg6, reg7]
 reg12 := call entryPrintInt[reg11]
 reg4 := 0
 goto L5
L5
```

```
---------
FRAMES :

Frame{
  entryPoint=L0,
  exitPoint=L1,
  parameters=[reg0, reg1],
  result=reg2,
  locals=[reg5],
  size=0
}
Frame{
  entryPoint=L2,
  exitPoint=L3,
  parameters=[],
  result=reg3,
  locals=[],
  size=0
}
Frame{
  entryPoint=L4,
  exitPoint=L5,
  parameters=[],
  result=reg4,
  locals=[reg6, reg7, reg8, reg11, reg12],
  size=0
}
```

Indications : `entryPrintInt` est une fonction prédéfinie dont le frame n'apparaît pas ici. La fonction dont le label d'entrée est `L4` est la fonction d'entrée du programme (la fonction `main`). `L5` correspond donc à la fin du programme.

1. Donnez un code source (écrit de la façon la plus naturelle possible) depuis lequel aurait pu être compilée cette représentation.
2. Donnez le code assembleur x86 qui serait généré à partir de cette représentation intermédiaire uniquement pour les lignes entre L2 et L3. Attention, en L2 on rentre dans une fonction, il faut résoudre cet appel.[1]

[1] Le compilateur du cours génère 12 lignes pour ce bout de code.