CMP2 – Construction des compilateurs

EPITA – Promo 2011
Tous documents (notes de cours, polycopiés, livres) autorisés
Calculatrices et ordinateurs interdits.

Mars 2009 (1h30)

Une copie synthétique, bien orthographiée, avec un affichage clair des résultats, sera toujours mieux notée qu'une autre demandant une quelconque forme d'effort de la part du correcteur. Une argumentation informelle mais convaincante, sera souvent suffisante.

Cette épreuve est longue ; le but n'est pas de répondre vite et mal à toutes les questions, mais plutôt d'en faire bien une partie significative. Une lecture préalable du sujet est recommandée.

Écrivez votre nom en haut de la première page du sujet, et rendez-le avec votre copie.

1 Bootstrap

1. Combien de nombres entiers différents peut-on représenter avec 64 bits ?
2. Combien de nombres *flottants* différents peut-on représenter avec 64 bits ?
3. Pourquoi est-ce que l'écriture

   ```
   print_int (−2147483648 / 42)
   ```

   est interdite par notre compilateur Tiger[1]. ?

1. Pour information, $-2147483648 = -2^{31}$

2 Partie centrale

1. Définissez la partie centrale d'un compilateur.
2. Ci-après figure la représentation intermédiaire haut-niveau (HIR), en langage Tree, d'un programme Tiger [2].

```
move
 temp rv
 eseq
  move temp t2 const 1
  eseq
  seq
   label l2
   cjump gt temp t1 const 0 name l3 name l1
   label l3
   seq
    move
     temp t2
     binop mul temp t2 temp t0
    move
     temp t1
     binop sub temp t1 const 1
   seq end
   jump name l2
   label l1
  seq end
  temp t2
jump ra
```

   « Décompilez-le » en un programme Tiger ou C équivalent.
3. Que fait ce code ?
4. Nous allons *canoniser* ce programme Tree pour le transformer en représentation intermédiaire bas-niveau (LIR). La première étape consiste en la *linéarisation* du programme. Pour chacune des règles ci-dessous, écrivez la partie droite de la règle de réécriture (sur votre copie).
   (a) **move** (**temp** t, **eseq** (s, e)) =>
   (b) **move** (**mem** (**eseq** (s, e1)), e2) =>
   (c) **move** (**mem** (e1), **eseq** (s, e2)) =>
5. Aidez-vous des règles que vous avez écrites à la question précédente ainsi que de celles de l'annexe A pour réécrire le programme en une version linéarisée (sans `eseq` ni `seq`).
6. Découpez le code de la question précédente en blocs basiques (commençant par une étiquette et terminant par un saut conditionnel ou inconditionnel).
7. Réassemblez les blocs basiques de la question précédente en une trace valide (de tel sorte que les `cjumps` soient normalisés en saut à une branche).

2. Les plus attentifs remarqueront que ce code
   – ne comporte que le corps d'une routine (par de prologue ni d'épilogue) ;
   – n'est pas encapsulé dans une `seq` ;
   – comporte une instruction finale `jump ra` (normalement absente d'une sortie HIR de `tc`).
Il s'agit de modifications visant à simplifier l'exercice.

3 Partie terminale

Dans cet exercice, on considère une machine hypothétique, dont le jeu d'instructions est suffisamment clair pour ne pas avoir à être défini, et qui possède trois registres, `r1`, `r2`, `r3`. Le premier est sous la responsabilité de l'appelant, et les deux autres sous celle de l'appelé.

1. Définissez la partie terminale d'un compilateur.
2. Écrivez un programme Tiger ou C correspondant au programme suivant, avant allocation des registres.

```
enter:  u <− r3
        t <− r2
        n <− r1
        a <− 0
        b <− 1
loop:   if n <= 0 jump exit
        c <− a + b
        a <− b
        b <− c
        n <− n − 1
        jump loop
exit:   r1 <− b
        r2 <− t
        r3 <− u
        return
```

3. Que fait ce programme ?
4. D'après ce programme, détailler les conventions d'appel de cette machine, et en particulier, pour chacun des registres dire lesquels servent au passage d'argument, et au retour de résultat.
5. Étant données les conventions d'appel, le sens du programme, expliquer quelles sont les variables vives lors du `return`.
6. Expliquez pourquoi les lignes

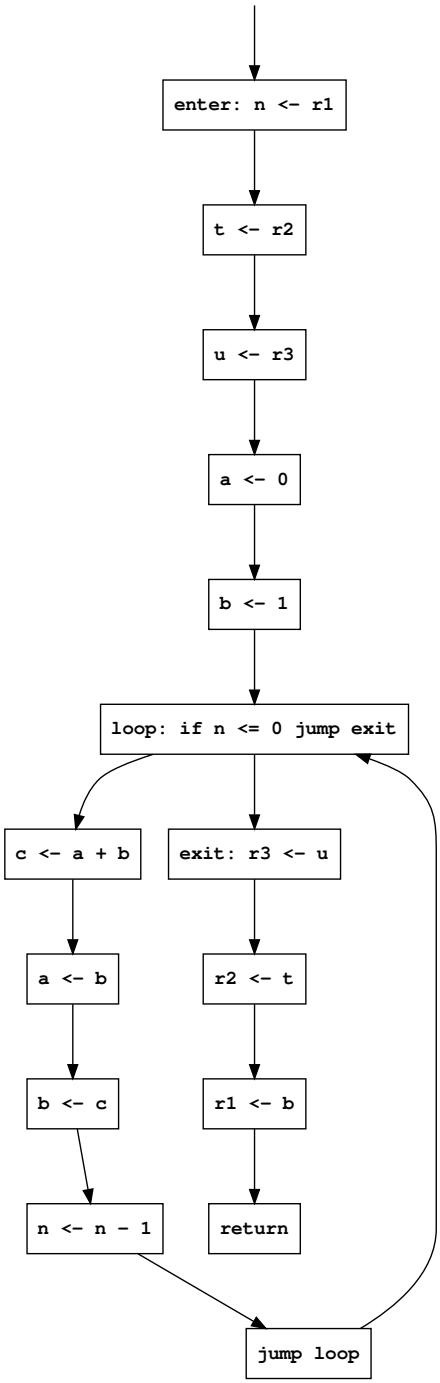
```
u <− r3
t <− r2
```

et

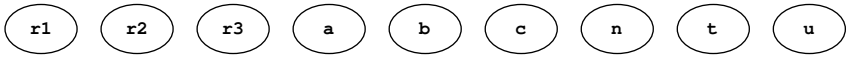
```
r2 <− t
r3 <− u
```

ont été générées.

7. Étiquetez les ~~arêtes~~ arcs du graphe de contrôle de flux de ce programme ci-après avec les temporaires vives (**répondez directement sur le sujet**).



8. Complétez le graphe d'interférence en en traçant les arêtes, y compris les fusions (*coalesce*) possibles avec des arêtes en pointillés (**répondez directement sur le sujet**).



9. Expliquez pourquoi il est impossible de colorier ce graphe tel quel.
10. Nous allons donc devoir utiliser la mémoire pour diminuer la pression sur les registres. Combien de temporaires va-t-il au moins falloir verser (*spill*) sur la pile pour que le problème de l'allocation soit soluble ? Justifiez votre réponse.
11. Réécrivez le programme en versant une ou plusieurs variables sur la pile de sorte à ce que le nouveau graphe associé puisse être colorié.

4 À propos de ce cours

Pour terminer cette épreuve, nous vous invitons à répondre à un petit questionnaire. Les renseignements ci-dessous ne seront bien entendu pas utilisés pour noter votre copie. Ils ne sont pas anonymes, car nous souhaitons pouvoir confronter réponses et notes. En échange, quelques points seront attribués pour avoir répondu. Merci d'avance.

Sauf indication contraire, vous pouvez cocher plusieurs réponses par question. Répondez sur la feuille de QCM qui vous est remise. N'y passez pas plus de dix minutes.

Le cours

1. Quelle a été votre implication dans les cours CMP1, CMP2 et TYLA ?
   A Rien.
   B Bachotage récent.
   C Relu les notes entre chaque cours.
   D Fait les annales.
   E Lu d'autres sources.
2. Ce cours
   A Est incompréhensible et j'ai rapidement abandonné.
   B Est difficile à suivre mais j'essaie.
   C Est facile à suivre une fois qu'on a compris le truc.
   D Est trop élémentaire.
3. Ce cours
   A Ne m'a donné aucune satisfaction.
   B N'a aucun intérêt dans ma formation.
   C Est une agréable curiosité.
   D Est nécessaire mais pas intéressant.
   E Je le recommande.
4. La charge générale du cours en sus de la présence en amphi (relecture de notes, compréhension, recherches supplémentaires, etc.) est
   A Telle que je n'ai pas pu suivre du tout.
   B Lourde (plusieurs heures par semaine).
   C Supportable (environ une heure de travail par semaine).
   D Légère (quelques minutes par semaine).

Les formateurs

5. L'enseignant
   A N'est pas pédagogue.
   B Parle à des étudiants qui sont au dessus de mon niveau.
   C Me parle.
   D Se répète vraiment trop.
   E Se contente de trop simple et devrait pousser le niveau vers le haut.
6. Les assistants
   A Ne sont pas pédagogues.

B Parlent à des étudiants qui sont au dessus de mon niveau.

C M'ont aidé à avancer dans le projet.

D Ont résolu certains de mes gros problèmes, mais ne m'ont pas expliqué comment ils avaient fait.

E Pourraient viser plus haut et enseigner des notions supplémentaires.

Le projet Tiger

7. Vous avez contribué au développement du compilateur de votre groupe (une seule réponse attendue) :
   A Presque jamais.
   B Moins que les autres.
   C Équitablement avec vos pairs.
   D Plus que les autres.
   E Pratiquement seul.
8. La charge générale du projet Tiger est
   A Telle que je n'ai pas pu suivre du tout.
   B Lourde (plusieurs jours de travail par semaine).
   C Supportable (plusieurs heures de travail par semaine).
   D Légère (une ou deux heures par semaine).
   E J'ai été dispensé du projet.
9. Y a-t-il de la triche dans le projet Tiger ? (Une seule réponse attendue.)
   A Pas à votre connaissance.
   B Vous connaissez un ou deux groupes concernés.
   C Quelques groupes.
   D Dans la plupart des groupes.
   E Dans tous les groupes.

Questions 10-16 Le projet Tiger vous a-t-il bien formé aux sujets suivants ? Répondre selon la grille qui suit. (Une seule réponse attendue par question.)

A Pas du tout
B Trop peu
C Correctement
D Bien
E Très bien

10. Formation au C++.
11. Formation à la modélisation orientée objet et aux *design patterns*.
12. Formation à l'anglais technique.
13. Formation à la compréhension du fonctionnement des ordinateurs.
14. Formation à la compréhension du fonctionnement des langages de programmation.
15. Formation au travail collaboratif.
16. Formation aux outils de développement (contrôle de version, systèmes de construction, débogueurs, générateurs de code, etc.)

Questions 17-29 Comment furent les étapes du projet ? Répondre selon la grille suivante. (Une seule réponse attendue par question ; ne pas répondre pour les étapes que vous n'avez pas faites.)

A Trop facile.
B Facile.
C Nickel.
D Difficile.
E Trop difficile.

17. Rush `.tig` : mini-projet en Tiger (Bistromatig).
18. `TC-0`, Scanner & Parser.
19. `TC-1`, Scanner & Parser, Tâches, Autotools.
20. `TC-2`, Construction de l'AST et pretty-printer.
21. `TC-3`, Liaison des noms et renommage.
22. `TC-E`, Calcul des échappements.
23. `TC-4`, Typage.
24. Désucrage des constructions objets (transformation Tiger → Panther).
25. `TC-5`, Traduction vers représentation intermédiaire.
26. Option `TC-A`, Surcharge des fonctions.
27. Option `TC-D`, Suppression du sucre syntaxique (boucles `for`, comparaisons de chaînes de caractères).
28. Option `TC-B`, Vérification dynamique des bornes de tableaux.
29. Option `TC-I`, Mise en ligne du corps des fonctions.

A Quelques règles de réécriture pour la canonisation

```
eseq (s1, eseq (s2, e))      =>  eseq (seq (s1, s2), e)

binop (op, eseq (s, e1), e2)  =>  eseq (s, binop (op, e1, e2))

jump (eseq (s, e1))          =>  seq (s, jump (e1))

cjump (op, eseq (s, e1), e2, l1, l2)
 =>  seq (s, cjump (op, e1, e2, l1, l2))

seq (ss1, seq (ss2), ss3)    => seq (ss1, ss2, ss3)
```

Si `e1` et `s` commutent :

```
binop (op, e1, eseq (s, e2))
 =>  eseq (s, binop (op, e1, e2))

cjump (op, e1, eseq (s, e2), l1, l2)
 =>  seq (s, cjump (op, e1, e2, l1, l2))
```

Si `e1` et `s` ne commutent pas :

```
binop (op, e1, eseq (s, e2))
 =>  eseq (seq (move (temp t, e1), s),
        binop (op, temp t, e2))

cjump (op, e1, eseq (s, e2), l1, l2)
 =>  seq (seq (move (temp t, e1), s),
       cjump (op, temp t, e2, l1, l2))
```

(où `t` est une temporaire fraîche).