Partiel de Synthèse de Compilation

EPITA – Promo 2005 – **Tous documents autorisés** *

Septembre 2003

Une copie concise, bien orthographiée, dont les résultats sont clairement exposés, sera toujours mieux notée qu'une copie qui aura demandé une quelconque forme d'effort de la part du correcteur.

Certaines questions suggèrent volontairement une mauvaise réponse. Dans ce cas il vous faut expliquer brièvement la bonne façon de résoudre le problème évoqué par la question.

1 Partie frontale

1. Définir la partie frontale d'un compilateur.
2. On cherche à ignorer les commentaires Tiger dans notre reconnaisseur lexical engendré par Flex. Montrer l'expression régulière qui permet de le faire.
3. Montrer le code Bison, **avec les actions écrites en C++ avec STL**, permettant d'analyser une liste de zéro ou plusieurs `exp` séparées par des points-virgules. Bien entendu, on ne détaillera pas la définition de `exp` lui-même.
4. Qu'est-ce qu'un AST ?
5. On désire optimiser statiquement, au niveau Tiger, des programmes tels que

   ```
   if 0 = 1 - 2 / 2 then print ("foo\n") else print ("bar\n")
   ```

   Expliquer comment vous allez faire dans l'architecture `tc` Illustrer le fonctionnement de votre méthode en prenant appui sur l'exemple.
6. Quel problème significatif la question précédente rencontre avec l'architecture courante de la partie frontale de `tc`.
7. Proposer des solutions.

2 Partie centrale

1. Définir la partie centrale d'un compilateur.
2. Pour les expressions Tree suivantes, montrer ce que les règles de canonisation doivent produire. Détailler les cas où des résultats différents pourraient être obtenus selon des conditions de commutabilité.

   ```
   move (temp t, eseq (s, e))
   ```
3. `sxp (eseq (s, e))`
4. `sxp (call (f, e1, e2, eseq (s, e3)))`

*"Tout document autorisé" signifie que notes de cours, livres, annales, etc. sont explicitement consultables pendant l'épreuve. Le zèle de la part des surveillants n'a pas lieu d'être, mais dans un tel cas me contacter au 01 53 14 59 42.

3 Partie terminale

Dans cette section, on considère une machine hypothétique, dont le jeu d'instruction est suffisamment clair pour ne pas avoir à être défini, et qui possède trois registres, `r1`, `r2`, `r3`. Les deux premiers sont sous la responsabilité de l'appelant, et le dernier sous celle de l'appelé.

1. Définir la partie terminale d'un compilateur.
2. "Désassembler" le programme suivant, avant allocation des registres, en un programme C ou Tiger :

```
enter: n  <- r1
       c  <- r3
       r  <- 1
loop:  if n <= 1 jump exit
       r  <- r * n
       n  <- n - 1
       goto loop
exit:  r1 <- r
       r3 <- c
       return
```

3. 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 ?
4. Étant données les conventions d'appel, le sens du programme, expliquer quelles sont les variables vives lors du `return`.
5. Expliquer pourquoi les lignes `c <- r3` et `r3 <- c` ont été générées.
6. Calculer son graphe de contrôle de flux étiqueté sur les arêtes par les temporaires vives.
7. Calculer le graphe d'interférence, y compris les fusions (*coalesce*) possibles.
8. Est-il possible de le colorer tel quel ?
9. Détaillez ce que il faut faire lorsque la coloration de ce graphe échoue. Dans la suite, on considère que si la coloration a effectivement échoué dans le cas présent, alors vous avez effectué les opérations nécessaires pour poursuivre l'allocation des registres.
10. Proposer une coloration du graphe obtenu.
11. Donner le code assembleur final.