Examen final de compilation

ÉNSIIE, semestre 3

jeudi 10 janvier 2013

Durée : 1h45.
Tout document personnel autorisé (pas de prêt entre voisins).
Ce sujet comporte six exercices indépendants, qui peuvent être traités dans l'ordre voulu.
Il contient 3 pages.
Il va de soi que toute réponse devra être justifiée.
Le barème est donné à titre indicatif, il est susceptible d'être modifié. Le total est sur 20 points. Les questions précédées d'une étoile ($\star$) ont une difficulté plus grande et peuvent n'être traitées qu'à la fin.

Exercice 1 : Syntaxe (2 points)

Donner l'arbre de syntaxe abstraite des instructions Pseudo Pascal suivantes :

1. `e := e + 1`
2. `t[i + 4 * j] := new array of integer [22 + x]`
3. `while x <> y do if x > y then x := x - y else y := y - x`

Exercice 2 : Analyse syntaxique (4 points)

On considère la grammaire suivante :

$$\begin{aligned} L &\to aaLb \\ &\mid aLbb \\ &\mid c \end{aligned}$$

1. Construire l'automate LR(0) pour cette grammaire.
2. Cette grammaire est-elle LR(0) ?
3. ($\star$) Existe-il une grammaire LR(0) reconnaissant le même langage ?

Exercice 3 : Sélection d'instruction (3 points)

1. Donner une traduction naïve de l'expression Pseudo Pascal `3 * (4 * x)` en UPP.

On considère le système de réécriture suivant :

$$\begin{aligned}
\texttt{mul}(\texttt{li}(\underline{1}), e) &\to e \\
\texttt{mul}(\texttt{li}(\underline{0}), e) &\to \texttt{li}(\underline{0}) \\
\texttt{mul}(\texttt{li}(\underline{2^n}), e) &\to \texttt{sll}(e, \underline{n}) && \text{pour } n \geq 1 \\
\texttt{mul}(\texttt{li}(\underline{k}), \texttt{sll}(e, \underline{n})) &\to \texttt{mul}(\texttt{li}(\underline{k * 2^n}), e)
\end{aligned}$$

Bien que ce ne soit pas utile pour les questions suivantes, on rappelle que l'instruction MIPS `sll` permet de décaler les bits d'un entier vers la gauche.

2. Donner la forme normale de la traduction naïve de `3 * (4 * x)`.
3. Le système de réécriture est-il fortement terminant ?
4. Le système de réécriture est-il confluent ? Si non, proposer un système de réécriture équivalent qui soit confluent.

Exercice 4 : Graphe de flot de contrôle (2 points)

On considère l'instruction Pseudo Pascal suivante :

```
while x <> y do if x > y then x := x - y else y := y - x
```

1. Dessiner son graphe de flot de contrôle (en conservant dans les nœuds une syntaxe à la Pseudo Pascal).
2. Traduire cette instruction en RTL (on associera `x` à `%0` et `y` à `%1`).
   Indication : l'instruction MIPS `sub` r_d`,` r_1`,` r_2 place dans le registre r_d le résultat de la soustraction du contenu du registre r_1 par le contenu du registre r_2.

Exercice 5 : Convention d'appel (5 points)

On considère le programme Pseudo Pascal suivant :

```
1 var r : integer
2 function pgcd(x : integer, y : integer)
3   if x <> y
4   then if x > y
5     then pgcd := pgcd(x - y, y)
6     else pgcd := pgcd(x, y - x)
7   else pgcd := x
8 r := pgcd(60,36)
```

Convention par pile

Dans cette partie, on suppose qu'on utilise la convention d'appel par pile : les arguments, l'adresse de retour et la valeur de retour sont passés sur la pile, et les variables locales y sont sauvegardées.

1. Représenter la trame de la fonction `pgcd`.
2. Quelle est l'évolution de la pile lors de l'exécution du programme (sans optimisation des appels terminaux) ?

Convention d'appel MIPS

Dans cette partie, on suppose qu'on utilise la convention d'appel MIPS comme vue en cours. On suppose que la variable globale `r` est stockée dans le registre `$s0`.

3. Quels registres `pgcd` doit-elle sauvegarder ? En déduire la trame de `pgcd`.
4. Décrire l'évolution de la pile et du contenu des registres `$a0 $a1 $v0 $ra $s0` au cours du programme, sans optimisation des appels terminaux.
5. ($\star$) On applique maintenant l'optimisation des appels terminaux dans `pgcd`. Y a-t-il des registres que `pgcd` n'a plus besoin de sauvegarder ? En déduire l'évolution de la pile et des registres. À quelle fonction impérative `pgcd` correspond-elle alors ?

Exercice 6 : Allocation de registres (4 points)

On considère le programme Pseudo Pascal suivant :

```
1 a := 0;
2 if e > 0
3 then d := e
4 else d := -e;
5 b := 2 * d;
6 c := 3 * d;
7 while a < c do begin
8   a := a + b;
9   c := c + 1 end
```

1. Dessiner le graphe de flot de contrôle correspondant (en gardant une syntaxe Pseudo Pascal pour les instructions de base).
2. Donner les variables vivantes en chacun des points du programme.
3. Dessiner le graphe d'interférence du programme de départ (avec les arêtes de préférence).
4. Essayer de 2-colorier ce graphe en appliquant l'algorithme de George et Appel. On détaillera le déroulement de l'algorithme et on justifiera le critère utilisé lors d'une fusion.