Examen final de compilation

ÉNSIIE, semestre 3

vendredi 13 janvier 2017

Durée : 1h45.
Tout document personnel autorisé (pas de prêt entre voisins).
Ce sujet comporte 5 exercices indépendants, qui peuvent être traités dans l'ordre voulu.
Il contient 4 pages.
Certaines questions, précédées par le symbole ($\star$) sont plus difficiles et pourront être traitées à la fin.
Le barème est donné à titre indicatif, il est susceptible d'être modifié. Le total est sur 20 points.
Il va de soi que toute réponse devra être justifiée.

Exercice 1 : Syntaxe (2,5 points)

Donner l'arbre de syntaxe abstraite des instructions Pseudo Pascal suivantes, ou expliquer pourquoi elles ne sont pas syntaxiquement correctes :

1. `if c < 0 and y > 3 then x := 1 else y := x`
2. `if c < 0 and y > 3 then begin x := 1 else y := x end`
3. `begin if c < 0 and y > 3 then x := 1 else y := x end`
4. `if c < 0 and y > 3 then begin x := 1 end else y := x`
5. `if begin c < 0 and y > 3 end then x := 1 else y := x`

Exercice 2 : Analyse sémantique (2,5 points)

Soit le programme Pseudo-Pascal suivant :

```
1  program
2
3  var x : integer;
4
5  procedure tab (n : integer);
6  var a, i : integer;
7  begin
8    a := new array of integer [n];
9    i := 0;
10   while i < n do begin
11     a[x] := i;
```

```
12      i := i + 1
13    end;
14    tab := a
15  end;
16
17  function f () : integer;
18  var x : array of integer;
19  begin
20    a := 0;
21    f := x
22  end;
23
24  begin
25     f(1);
26     x := tab(100)
27  end.
```

1. Représenter la table des symboles. Parmi les informations stockées on indiquera la sorte (variable globale, variable locale, fonction, procédure), le type, la portée (globale ou locale à la fonction x) et la ligne de déclaration.
2. Quelle(s) erreur(s) l'analyse sémantique va-t-elle produire ? On fera un parcours systématique de l'AST.

Exercice 3 : Réécriture (5 points)

On considère un opérateur `app` attendant 2 arguments, et trois constantes `k`, `i`, `w`, ainsi que le système de réécriture suivant :

$$\begin{aligned} \texttt{app}(\texttt{app}(\texttt{k}, X), Y) &\to X \\ \texttt{app}(\texttt{i}, X) &\to X \end{aligned}$$

1. Donner la ou les forme(s) normale(s) de $\texttt{app}(\texttt{app}(\texttt{app}(\texttt{k}, \texttt{i}), \texttt{w}), \texttt{app}(\texttt{i}, \texttt{w}))$.
2. Le système de réécriture est-il terminant ?
3. Calculer les paires critiques du système. Lesquelles sont-elles joignables ?
4. Le système de réécriture est-il confluant ?

On ajoute la règle

$$\texttt{app}(\texttt{app}(\texttt{w}, X), Y) \to \texttt{app}(\texttt{app}(X, Y), Y)$$

5. Le système de réécriture est-il terminant ?
   Indication : on pourra s'intéresser au comportement de cette règle quand on remplace X et Y par des constantes bien choisies.
6. Calculer les paires critiques du système. Lesquelles sont-elles joignables ?
7. ($\star$) Le système de réécriture est-il confluant ?

Exercice 4 : Graphe de flot de contrôle et allocation de registres (5 points)

Soit le programme Pseudo-Pascal suivant :

```
1 prod :=  x * y;
2 while x < y
3       or x > y do
4   if x < y
5   then y := y - x
6   else x := x - y;
7 ppcm := prod / x;
8 pgcd := x;
9 writeln(ppcm, pgcd)
```

`writeln` est un procédure d'affichage, elle va donc engendrer les variables `ppcm` et `pgcd`.

1. Donner le graphe de flot de contrôle correspondant en syntaxe Pseudo Pascal. Ne pas oublier le comportement coupe-circuit du `or`.
2. Indiquer dans un tableau quelles variables sont vivantes en chacun des points du programme.
3. Donner le graphe d'interférence du programme (avec les arêtes de préférence). On indiquera sur les arêtes le numéro de ligne d'une instruction qui a causé l'interférence ou la préférence.
4. Essayer de 2-colorier ce graphe en appliquant l'algorithme de George et Appel. On détaillera bien chaque étape. On utilisera le critère de George pour justifier les fusions éventuelles.

Exercice 5 : Convention d'appel (5 points)

On considère le programme Pseudo Pascal suivant :

```
1  program
2
3  var t : array of integer;
4
5  function odd (n : integer) : integer;
6  begin
7     if n = 0 then
8        odd := 0
9     else
10       odd := even (n-1)
11 end;
12
```

```
13 function even (n : integer) : integer;
14 begin
15    if n = 0 then
16       even := 1
17    else
18       even := odd (n-1)
19 end;
20
21 procedure check ();
22 var i : integer;
23 begin
24    i := 100;
25    t := new array of integer [i+1];
26    while (i >= 0) do begin
27       t[i] := even(i);
28       i := i - 1
29    end
30 end;
31
32 begin
33    check()
34 end.
```

On suppose qu'on utilise la convention d'appel MIPS comme vue en cours. On supposera que la variable locale `i` de la procédure `check` est stockée dans le registre sauvegardé par l'appelant `$t0`.

1. Quels registres `odd` doit-elle sauvegarder ? En déduire la trame de `odd`.
2. Même question pour `check`.
3. Pour expliciter la convention d'appel, quelles instructions faut-il ajouter :
   a) Au début de `odd` ?
   b) Avant l'appel à `even` dans `odd` ?
   c) Après l'appel à `even` dans `odd` ?
   d) À la fin de `odd` ?
   e) Au début de `check` ?
   f) Avant l'appel à `even` dans `check` ?
   g) Après l'appel à `even` dans `check` ?
   h) À la fin de `check` ?
4. On optimise les appels terminaux dans `odd` et dans `even`. Quel impact cela a-t-il sur les trames de `odd` et `even` ?