Université Bordeaux I **Master Informatique 2015-2016**
Amphis A et D – BATIMENT A29 **Compilation**
Lionel Clément **mercredi 9 décembre 2015 – de 14h00 à 15h30**

Examen de Compilation

Ce sujet comporte 4 pages – documents autorisés

Note d'information: la 3^{e}partie du projet de travaux dirigés est à rendre le 18 décembre 2015

Exercice 1 : 6 points

Une expression de type est définie inductivement de la sorte :

- Un type de base (parmi lesquels on trouve *integer*, *real*, *boolean*, *char*) est une expression de type ;
- Une variable de type $'x$ est une expression de type ;
- Un nom de type n est une expression de type ;
- Si T_1 et T_2 sont deux expressions de type, $T_1 \times T_2$ est une expression de type ;
- Si T_1 et T_2 sont deux expressions de type, $T_1 \to T_2$ est une expression de type ;
- Si T est une expression de type, le type constructeur $constructeur(T)$ est une expression de type. Parmi les constructeurs, nous trouverons *pointer*, *structure*, *list*, et $array[N]$.
  Nous notons `^T` le type $pointer(T)$.

Dans le code suivant, nous avons deux déclarations pour des types `Noeud` et `Liste`, deux déclarations de fonctions `push` et `pop` et le code.

`node^` désigne la valeur pointée par `node`.

Si `var` est de type `structure(T)`, `var.n` désigne le champs `n` de la structure.

`new T` désigne un pointeur vers une nouvelle allocation de type `T`. `null` désigne un pointeur nul.

```
1
2  typedef Noeud = structure {
3          info: 'x;
4          precedent: ^Noeud;
5          suivant: ^Noeud;
6  }
7
8  typedef Liste = structure {
9          tete: ^Noeud;
10         queue: ^Noeud;
11 }
12
13 function push(liste: ^Liste, info: 'y){
14         node: ^Node;
15         node = liste^.tete;
16         liste^.tete = new Node;
17         node^.precedent = liste^.tete;
18         liste^.tete^.suivant = node;
19         liste^.tete^.precedent = null;
20         liste^.tete^.info = info;
21         if (liste^.queue == null)
```

```
22                  liste^.queue = liste^.tete;
23  }
24
25  function pop(liste: ^Liste): 'z{
26          node: ^Node;
27          if (liste^.tete^.suivant != null)
28                  liste^.tete^.suivant^.precedent = null;
29          node = liste^.tete;
30          liste^.tete = liste^.tete^.suivant;
31          return node^.info;
32  }
33
34  begin
35          pile1: ^Liste;
36          pile2: ^Liste;
37          pile1 = new Liste;
38          push(pile1, 1);
39          push(pile1, 2);
40          print(pop(pile1));
41          pile2 = new Liste;
42          push(pile2, 'a');
43          push(pile2, 'b');
44          print(pop(pile2));
45   end
```

Questions

1. Quelle est l'expression de type de `Liste` ?

   Réponse

   $structure((tete \times pointer(Noeud)) \times (queue \times pointer(Noeud)))$

2. Est-ce que la fonction *pop* est polymorphe ? Est-ce qu'il y a une seule ou deux implémentations de la fonction *pop* après compilation du programme ? Motiver la réponse.

   Réponses

   (a) Oui, car la variable de type $'x$ peut prendre plusieurs types différents.

   (b) Il y a deux implémentations différentes de `pop` dans ce programme, selon le type de la valeur de retour : `char` ou `integer`. En effet, l'enregistrement dans la pile, puis l'affectation de cette valeur de retour peut être différente selon son type, qui n'est pas nécessairement simple.

3. En se contentant des quelques lignes affichées dans l'exemple, Est-ce qu'on peut affirmer que le typage est dynamique ? Pourquoi ?

   Réponse

   Non, il est statique, car l'ensemble des calculs de type est réalisé lors de la compilation. Les fonctions `push` et `pop` sont polymorphes, mais vont s'appliquer avec des types connus après compilation.

4. Soit g, une fonction dont l'expression de type est $(pointeur('k) \rightarrow real))$
   Quel est le plus général unificateur de g et de *pop* s'il existe ?

Réponse

$< (`k, structure((tete \times pointer(Noeud)) \times (queue \times pointer(Noeud)))), (`z, real) >$

Exercice 2 : 4 points

Soit le code à 3 adresses résumé ainsi :

- $x = y$ `op` z
  Instruction d'affectation où *op* est un opérateur binaire arithmétique ou logique
- $x =$ `op` y
  Instruction d'affectation où *op* est un opérateur unaire arithmétique, logique, de décalage ou de conversion
- $x = y$
  Instruction de copie où la valeur de y est affectée à x
- `LABEL` L
  Étiquette l'instruction qui suit par le label L
- `JUMP` L
  Branchement inconditionnel qui a pour effet de faire exécuter ensuite l'instruction étiquetée par L
- `IF` x *op* y `JUMP` L
  Branchement conditionnel qui a pour effet de faire exécuter ensuite l'instruction étiquetée par L si la relation x *op* y est satisfaite. Où *op* est un opérateur relationnel ($<$, $>$, $\leq$, ...)
- `PARAM` x
  Passe un paramètre par valeur à une procédure
- `CALL` p, n
  Appel de la procédure p qui prendra en compte n paramètres
- `RETURN` y
  La procédure renvoie la valeur y

Soit le programme suivant :

```
1 a = 0
2 if (i >= 10)
3         while (i > 0)
4                 a = a + i--;
5                 if (i = 0)
6                         break;
7                 foo(a, i);
8 else
9         a = i;
```

Question

Écrire une suite d'instructions de code à 3 adresses qui correspond au programme.

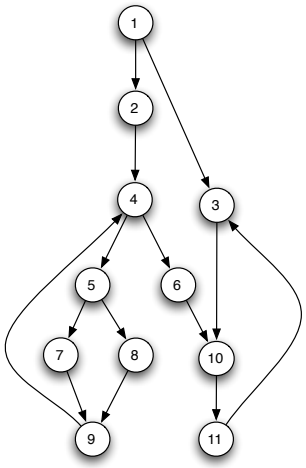
Réponse

```
1 a = 0
2 IF i >= 10 JUMP L1
3 JUMP L2
4 LABEL L1
5  LABEL L3
6  IF i > 0 JUMP L4
7  JUMP L5
8  LABEL L4
9    tmp = i
```

```
10    i = i - 1;
11    a = a + tmp
12    IF i = 0 JUMP L6
13    JUMP L7
14    LABEL L6
15     JUMP L5
16    LABEL L7
17    PARAM a
18    PARAM i
19    CALL foo
20   JUMP L3
21   LABEL L5
22 LABEL L2
23 a = i
```

Exercice 3 : 5 points

Soit le graphe de contrôle suivant, où 1 représente le bloc initial :



1. Est-ce que ce graphe de flot de contrôle contient une ou plusieurs boucles telles que définies dans le cours ? Si oui, lesquelles ?

   Réponse

   $\{4, 5, 7, 8, 9\}$

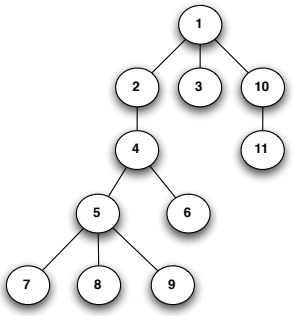
2. Est-ce qu'un programme écrit en langage Java permet de produire un tel flot de contrôle ? Pour quelle raison ?

   Réponse

   En Java, il est impossible d'écrire une instruction `goto`. Or l'arc $6 \rightarrow 10$ correspond à un saut vers l'intérieur d'une boucle qu'on ne peut pas programmer avec les structures de contrôle de Java.

3. Dessiner le graphe des dominants après avoir montré comment l'obtenir.

Réponse



Exercice 4 : 5 points

Soit G une grammaire algébrique $(\Sigma, \Phi, \mathtt{S}, \mathtt{R})$, avec $\Sigma = \{\mathtt{aff}, \mathtt{id}\}$, $\Phi = \{\mathtt{S}, \mathtt{E}\}$, dont les règles de production `R` sont les suivantes :

S → E
E → E aff E
E → id

Le tableau suivant résume l'analyse Earley de la suite `id aff id aff id`

état	à lire	lu	items
0	id aff id aff id	ϵ	0, S $\rightarrow$ • E 0, E $\rightarrow$ • E aff E 0, E $\rightarrow$ • id
1	aff id aff id	id	0, E $\rightarrow$ id • 0, S $\rightarrow$ E • 0, E $\rightarrow$ E • aff E
2	id aff id	id aff	0, E $\rightarrow$ E aff • E 2, E $\rightarrow$ • E aff E 2, E $\rightarrow$ • id
3	aff id	id aff id	2, E $\rightarrow$ id • 0, E $\rightarrow$ E aff E • 2, E $\rightarrow$ E • aff E 0, S $\rightarrow$ E • 0, E $\rightarrow$ E • aff E
4	id	id aff id aff	2, E $\rightarrow$ E aff • E 0, E $\rightarrow$ E aff • E 4, E $\rightarrow$ • E aff E 4, E $\rightarrow$ • id
5	ϵ	id aff id aff id	4, E $\rightarrow$ id • 2, E $\rightarrow$ E aff E • 0, E $\rightarrow$ E aff E • 4, E $\rightarrow$ E • aff E 0, E $\rightarrow$ E aff E • 2, E $\rightarrow$ E • aff E 0, S $\rightarrow$ E • 0, E $\rightarrow$ E • aff E

Questions

1. Expliquer comment peut-on conclure de ce tableau que la séquence est reconnue par l'analyseur.

   Réponse

   L'item (0, S $\rightarrow$ E •) appartient à l'état 5, et la séquence à analyser comporte 5 terminaux.
2. Expliquer comment peut-on conclure de ce tableau que la séquence est ambigüe.

   Réponse

   L'état 3 contient l'item (2, E $\rightarrow$ E • aff E) et l'item (0, E $\rightarrow$ E • aff E) issu de la réduction de l'item (0, E $\rightarrow$ E aff E •). L'action *shift* est donc ambigüe entre `(id aff id) aff id` et `id aff (id aff id)`.
3. En précisant que l'opérateur `aff` est associatif à droite, peut-on systématiquement obtenir l'analyse attendue ?

   Réponse

   Oui. Quand un état contient l'item (i, E $\rightarrow$ E • aff E), et que le prochain non terminal à lire est `aff`, il suffit d'en exclure systématiquement l'item (j, E $\rightarrow$ E aff E •). Ce qui revient à privilégier *shift* sur *reduce* pour cet opérateur.