Examen - Compilation (L3 Info)
Durée : 2h - documents interdits

19 mai 2015

1 Grammaires attribuées

On s'intéresse dans cet exercice à des arbres dont les nœuds sont étiquetés par des entiers de 0 à 9. De tels arbres peuvent être représentés par une expression parenthésée dans laquelle un nœud est représenté par son étiquette suivie de la liste de ses fils entre parenthèses. L'expression `1(2(3())4())`, par exemple, représente l'arbre ayant `1` pour racine. Cette dernière possède elle même deux fils (`2` et `4`) et `2` a pour fils `3`.

On appelle profondeur d'un noeud sa distance par rapport à la racine, dont la profondeur est 0. La hauteur d'un arbre est la profondeur maximale de ses nœuds. On appelle degré d'un nœud, le nombre de fils qu'il possède. Les nœuds de degré 0 sont appelés feuilles, les autres nœuds sont appelés nœuds internes.

Le langage des arbres est généré par la grammaire G ci-dessous [1] :

$$\begin{array}{lcllcl} R & \to & N & E & \to & 0 \\ N & \to & E(L) & E & \to & 1 \\ L & \to & NL & \ldots & & \\ L & \to & \varepsilon & E & \to & 9 \end{array}$$

Q.1. Dessinez l'arbre de dérivation correspondant à l'arbre `1(2(3())4())`

Dans chacune des questions suivantes on cherche à répondre à une question concernant un arbre généré par la grammaire. Pour chacune, il vous est demandé de définir des attributs ainsi que des actions sémantiques sur la grammaire G permettant de calculer la valeur des attributs. Tous les attributs sont à valeurs entières. Pour chacune des questions, vous pourrez utiliser les attributs que vous avez défini pour répondre aux questions précédentes.

Pour chaque attribut défini, vous indiquerez :

1. ce qu'il représente
2. s'il est hérité ou synthétisé
3. sa valeur pour chaque nœud de l'arbre de dérivation de la question précédente.

Pour calculer la valeur des attributs, vous pourrez utiliser les opérations arithmétiques, les opérations booléennes, l'opérateur binaire $max(x, y)$ ainsi que le prédicat binaire $egal(x, y)$ qui vaut 1 si $x = y$, et vaut 0 sinon.

Q.2. Combien de nœuds possède l'arbre ?

$$\begin{array}{lcll} R & \to & N & R.n = N.n \\ N & \to & E(L) & N.n = L.n + 1 \\ L & \to & NL_1 & L.n = N.n + L_1.n \\ L & \to & \varepsilon & L.n = 0 \end{array}$$

Q.3. Quel est le degré de chaque nœud N ?

$$\begin{array}{lcll} R & \to & N & R.d = N.d \\ N & \to & E(L) & N.d = L.d \\ L & \to & NL_1 & L.d = L_1.d + 1 \\ L & \to & \varepsilon & L.d = 0 \end{array}$$

Q.4. Quelle est la profondeur de chaque nœud N ?

$$\begin{array}{lcll} R & \to & N & N.p = 0 \\ N & \to & E(L) & L.p = N.p + 1 \\ L & \to & NL_1 & N.p = L.p \\ & & & L_1.p = L.p \end{array}$$

Q.5. Quelle est la hauteur de l'arbre ?

$$\begin{array}{lcll} R & \to & N & R.h = N.h \\ N & \to & E(L) & N.h = L.h + 1 \\ L & \to & NL_1 & L.h = \max(N.h, L_1.h) \\ L & \to & \varepsilon & L.h = 0 \end{array}$$

1. La première règle n'est pas nécessaire, mais elle sera utile pour la suite.

OU

R	$\rightarrow$	N	$R.h = N.h$
N	$\rightarrow$	$E(L)$	$N.h = L.h$
L	$\rightarrow$	NL_1	$L.h = \max(N.p, L_1.p)$
L	$\rightarrow$	ε	$L.h = 0$

Q.6. Combien de feuilles possède l'arbre ?

N	$\rightarrow$	$E(L)$	$N.f = L.f + egal(L.d, 0)$
L	$\rightarrow$	NL_1	$L.f = N.f + L_1.f$
L	$\rightarrow$	ε	$L.f = 0$

Q.7. Chaque nœud N a-t-il le même degré que la racine ?

R	$\rightarrow$	N	$N.r = N.d$
N	$\rightarrow$	$E(L)$	$N.m = egal(N.r, N.d)$
			$L.r = N.r$
L	$\rightarrow$	NL_1	$N.r = L.r$
			$L_1.r = L.r$

Q.8. L'arbre est il équilibré (tous ses nœuds internes sont de même degré) ?

R	$\rightarrow$	N	$R.e = N.e$
N	$\rightarrow$	$E(L)$	$N.e = egal(N.d, 0) \vee (N.m \wedge L.e)$
L	$\rightarrow$	NL_1	$L.e = N.e \wedge L_1.e$
L	$\rightarrow$	ε	$L.e = 1$

2 Compilateur d'arbres LaTeX

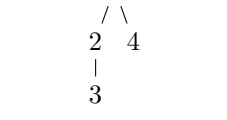
Il existe de nombreuses bibliothèques LaTeX pour dessiner des graphes et des arbres dans des documents, par exemple les bibliothèques `qtree` et `tikz`. L'objectif de cet exercice est d'écrire un compilateur en C qui transforme un arbre au format `tikz` en un arbre au format `qtree`. La figure 1 présente l'arbre de l'exercice précédent (a) dans les formats `qtree` (b) et `tikz` (c, d).

— **Arbre `qtree`** : pour obtenir le résultat (a) avec `qtree`, nous utilisons une représentation parenthésée similaire à l'exercice précédent, montrée en (b). Chaque sous-arbre est entouré de crochets `[` et `]`. Le crochet ouvrant est immédiatement suivi d'un point `.`, lui-même suivi de l'étiquette du nœud. Pour simplifier, nous considérons comme étiquettes possibles les entiers de 0 à 9. L'étiquette est suivie de la liste des sous-arbres séparés par des espaces. Cette liste est vide dans le cas d'une feuille.

— **Arbre `tikz`** : pour obtenir le résultat (a) avec `tikz`, nous utilisons une liste de nœuds, montrée en (c) et (d). Chaque nœud commence par la commande `\node` et se termine par un point-virgule `;`. Il contient obligatoirement une étiquette (un nombre entier entre 0 et 9), entre accolades. Entre la commande `\node` et l'étiquette, il est possible de spécifier entre parenthèses un identificateur pour le nœud. Pour simplifier, l'identificateur dans les exemples est identique à l'étiquette. Après l'étiquette, il est possible de spécifier une liste d'arcs vers des nœuds déjà déclarés. Il est impossible de rajouter un arc vers un nœud qui est déclaré plus loin. Chaque arc est représenté par le mot-clef `edge` suivi, entre parenthèses, de l'identificateur du nœud vers lequel il pointe. Les arcs de la liste sont séparés par des espaces.
Pour représenter des arbres, deux ordres sont possibles :

1. *rootFirst* : la racine est le *premier* nœud à être déclaré. Ensuite, chaque nœud possède exactement un arc vers son père. Les feuilles n'ont pas besoin d'identificateur. La racine est le seul nœud qui ne possède pas d'arc.
2. *rootLast* : la racine est le *dernier* nœud à être déclaré. Chaque nouveau nœud déclaré possède une liste d'arcs vers ses fils. La racine n'a pas besoin d'identificateur. Les feuilles ne possèdent pas d'arc.

1. Écrivez une grammaire LL(1) qui reconnaît un arbre `tikz`. Cette grammaire doit en particulier reconnaître les entrées 1(c) et 1(d). Afin de faciliter la manipulation et l'implémentation de la grammaire, essayez de minimiser le nombre de symboles non terminaux. Utilisez les symboles terminaux ci-dessous, renvoyés par l'analyseur lexical :
   — `NODE` : commande `\node`
   — `EDGE` : mot-clef `edge`
   — `CHIFFRE` : chiffre entre 0 et 9
   — Caractères `{ } ( ) ;`



(a) Exemple d'arbre, résultat de (b) (c) (d)

```
\Tree[.1 [.2 [.3 ] ] [.4 ] ]
```

(b) Source LaTeX de l'exemple, bibliothèque `qtree`

```
\node (1) {1};
\node (2) {2} edge (1);
\node {4} edge (1);
\node {3} edge (2);
```

(c) Source LaTeX de l'exemple, bibliothèque `tikz`, ordre *rootFirst*

```
\node (3) {3};
\node (2) {2} edge (3);
\node (4) {4};
\node {1}  edge (2) edge (4);
```

(d) Source LaTeX de l'exemple, bibliothèque `tikz`, ordre *rootLast*

FIGURE 1 – Exemple d'un arbre en LaTeX (a), avec trois formats (b) (c) (d) qui, compilés, génèrent le même résultat.

```
1. liste_noeuds -> noeud liste_noeuds
2. liste_noeuds ->
3. noeud -> NODE opt_id '{' CHIFFRE '}' opt_liste_edges ';'
4. opt_id -> '(' CHIFFRE ')'
5. opt_id ->
6. opt_liste_edges -> EDGE '(' CHIFFRE ')' opt_liste_edges
7. opt_liste_edges ->
```

2. Calculez les PREMIER et SUIVANT pour chaque non terminal de la grammaire. Ensuite, construisez la table d'analyse $LL(1)$ de la grammaire.

	PREMIER	SUIVANT	NODE	EDGE	CHIFFRE	{	}	;	(	)	$\perp$
liste_noeuds	NODE ε	$\perp$	1								2
noeud	NODE	NODE $\perp$	3								
opt_id	(ε	{				5			4		
opt_liste_edges	EDGE ε	;		6				7			

Remarque : Pour les deux questions suivantes, vous pouvez écrire un seul programme.

3. Écrivez un reconnaisseur $LL(1)$ en C qui implémente la table d'analyse et reconnaît des entrées au format `tikz`. Vous pouvez utiliser directement les fonctions fournies dans l'annexe A.

4. Enrichissez l'analyseur de la question précédente de manière à créer un arbre abstrait lors de la reconnaissance. Nous nous intéressons désormais uniquement aux arbres décrits avec la stratégie *rootFirst*. L'annexe A présente les fonctions `creer_noeud` et `ajoute_fils` (lignes 20 à 41) que vous devez utiliser pour créer et relier les nœuds entre eux.

```
% void opt_liste_edges( t_noeud *fils ) {
%     if( uc == EDGE ) {
%         consommer( EDGE );
%         consommer( '(' );
%         ajoute_fils( yyval, fils );
%         consommer( CHIFFRE );
%         consommer( ')' );
%         opt_liste_edges( fils );
%     }
%     else if( uc != ';' )
%         erreur( "';' attendu, %c trouvé\n", uc );
% }
% int opt_id() {
%     int id = -1;
%     if( uc == '(' ) {
%         consommer( '(' );
%         id = yyval;
%         consommer( CHIFFRE );
%         consommer( ')' );
%     }
%     else if( uc != '{' )
```

```
%         erreur( "'{' attendu, %c trouvé\n", uc );
%     return id;
% }
% t_noeud *noeud() {
%     int id = -1, etiquette;
%     t_noeud *$$;
%     consommer( NODE );
%     id = opt_id();
%     consommer( '{' );
%     etiquette = yyval;
%     consommer( CHIFFRE );
%     consommer( '}' );
%     $$ = creer_noeud( id, etiquette );
%     opt_liste_edges( $$ );
%     consommer( ';' );
%     return $$;
% }
% t_noeud *liste_noeuds() {
%     t_noeud *$$;
%     if(uc == NODE) {
%         $$ = noeud();
%         liste_noeuds();
%         return $$; /* Seulement le premier noeud compte */
%     }
%     else if(uc != EOF)
%         erreur("noeud ligne %d non reconnu", nbligne);
% }
```

5. Écrivez une fonction en C ayant pour en-tête `void affiche_qtree( t_noeud *racine )`. Cette fonction prend en paramètre la racine de l'arbre abstrait, et affiche à l'écran le même arbre au format `qtree`. Le résultat de l'appel à cette fonction sur l'exemple de la figure 1(c) doit être 1(b). Remarquez dans la structure de données (lignes 8 à 13) que chaque nœud possède deux pointeurs : vers son premier fils et vers son frère droit.

```
% void affiche_qtree_rec( t_noeud *root ) {
%     if( root != NULL ) {
%         printf( "[.%d ", root->etiquette );
%         affiche_qtree_rec(root->fils);
%         printf("] ");
%         affiche_qtree_rec(root->frere);
%     }
% }
% void affiche_qtree( t_noeud *root ) {
%     printf( "\\Tree" );
%     affiche_qtree_rec( root );
%     printf( "\n" );
% }
```

A Fonctions disponibles

```c
#define NODE 'n'
#define EDGE 'e'
#define CHIFFRE '0'
#define ISNUMBER(a) ( a >= '0' && a <= '9' )
#define erreur(m, ...) { fprintf(stderr, "ERREUR : "); fprintf(stderr, m, __VA_ARGS__); exit(-1); }
/*****************************************************************************/
// noeud de l'arbre abstrait
typedef struct t_noeud {
    int id;
    int etiquette;
    struct t_noeud *fils;
    struct t_noeud *frere;
} t_noeud;

t_noeud *noeuds[ 10 ] = { NULL };    // ``table des symboles'' simplifiee
int uc;                              //unite courante
FILE *yyin;                          //fichier source tikz
int yyval = -1;                      //valeur d'un chiffre
/*****************************************************************************/
t_noeud *creer_noeud(int id, int etiquette) {
    t_noeud *n = malloc(sizeof(t_noeud));
    n->id = id;         n->etiquette = etiquette;
    n->fils = NULL;     n->frere = NULL;
    if( id >= 0 && noeuds[id] == NULL )
        noeuds[id] = n;
    else if( id >= 0 )
        erreur( "Deux noeuds avec le meme ID %d\n", id );
    return n;
}
/*****************************************************************************/
void ajoute_fils(int id_pere, t_noeud *fils) {
    t_noeud *na = noeuds[id_pere];
    if( na->fils == NULL )
        na->fils = fils;
    else {
        na = na->fils;
        while(na->frere != NULL)
            na = na->frere;
        na->frere = fils;
    }
}

/*****************************************************************************/
void consommer(char c){
    if(uc == c)
        uc = yylex();
    else
        erreur("%c attendu, %c trouve\n",c,uc);
}
/*****************************************************************************/
```