Examen

Documents interdits

Durée: 2h

Note Importante : il vous est demandé, dans certaines questions de cet examen, d'écrire du code en langage C. Veillez à écrire votre code de manière la plus lisible possible. Le code illisible ne sera pas lu !

1 Expressions régulières

L'objet général de cet exercice est de déterminer si un mot appartient au langage correspondant à une expression régulière donnée. Pour cela on écrira une grammaire des expressions régulières, à partir de laquelle un analyseur produira un arbre abstrait. Ensuite, on déterminera si un mot appartient au langage correspondant à l'expression régulière en parcourant son arbre abstrait.
Etant donné une expression régulière E, on note $L(E)$ le langage correspondant à E.
Voici la syntaxe des expressions régulières ainsi que le langage correspondant, où X et Y sont des expressions régulières et x est un élément de l'alphabet.

Expression régulière	Langage	Expression régulière	Langage
x	$\{x\}$	$X.Y$	$L(X).L(Y)$
ε	$\{\varepsilon\}$	$X + Y$	$L(X) \cup L(Y)$
$\emptyset$	$\emptyset$	X^*	$\bigcup_{n\geq 0} L(e)^n$

où $L.L' = \{w.w' \mid w \in L, w' \in L'\}$ et L^n est la concaténation de L avec lui même n fois (avec $L^0 = \{\varepsilon\}$).
L'expression régulière $a + b.a^*$, par exemple, correspond au langage $\{a, b, ba, baa \ldots\}$
On considère l'ensemble des terminaux $\{a, b, +, *, (,), .\}$.

1. Donner une grammaire non-ambiguë G des expressions régulières, en supposant que l'étoile est plus prioritaire que le produit, lui-même plus prioritaire que la somme.
2. Donner un arbre de dérivation de l'expression $(a + b)^*a$.

A partir de la grammaire G on construit un analyseur syntaxique en langage C pour expressions régulières. L'analyseur syntaxique produit un arbre abstrait dont les nœuds sont définis par la structure `noeud *` suivante.

```
typedef struct noeud_ {
  enum { produit, somme, etoile, a, b, epsilon, vide } type;
  struct noeud_ *op1;
  struct noeud_ *op2;
} noeud;
```

Si le type d'un nœud est `etoile`, le champ `op2` doit être à `NULL` ; si c'est `a`, `b`, `epsilon` ou `vide`, alors `op1` et `op2` sont à `NULL`.

3. Dessiner l'arbre abstrait correspondant à l'expression régulière de la question 2.

Pour la suite de l'exercice, on a besoin de savoir si le langage d'une expression régulière contient ε. Ceci sera réalisé par un parcours de l'arbre abstrait de l'expression régulière.

4. Écrire la fonction récursive `int contient_epsilon(noeud *e)` qui dit si le langage de l'expression `e` contient ε.

Le *résiduel* d'un langage L par rapport à une lettre c, noté L/c est l'ensemble des mots de L ayant c pour préfixe, auxquels on a éliminé ce préfixe. En d'autres termes :

$$L/c = \{w \mid w \in \Sigma^*, cw \in L\}.$$

Exemple : si $L = \{a, abc, b\}$, $L/a = \{\varepsilon, bc\}$
Si E est une expression régulière correspondant au langage L et c est un symbole, alors le langage L/c peut aussi être décrit par une expression régulière, grâce aux règles suivantes.

$$\begin{array}{rcl} x/y & = & \varepsilon \text{ si } x = y \\ (X.Y)/x & = & (X/x).Y \text{ si } \varepsilon \notin L(X) \\ (X+Y)/x & = & (X/x) + (Y/x) \\ \varepsilon/x & = & \emptyset \end{array} \qquad \begin{array}{rcl} x/y & = & \emptyset \text{ si } x \neq y \\ (X.Y)/x & = & (X/x).Y + Y/x \text{ si } \varepsilon \in L(X) \\ X^*/x & = & (X/x).X^* \end{array}$$

où X, Y sont des expressions régulières sur Σ, et $x, y \in \Sigma$.

4. Donner des expressions régulières qui représentent les résiduels par rapport à la lettre a des langages décrits par les expressions régulières suivantes : $(ab + bc)$, a^*b et $b(a + b)^*$.

On se donne à présent les primitives qui permettent de créer l'arbre abstrait.. **On ne demande pas de les implémenter.**

```
noeud* creer_a();          noeud* creer_somme(noeud *e1, noeud *e2);
noeud* creer_b();          noeud* creer_etoile(noeud *e);
noeud* creer_epsilon();    noeud* copie_noeud(noeud *e);
noeud* creer_vide();       noeud* creer_produit(noeud *e1, noeud *e2);
```

5. Ecrire la fonction `noeud* residuel(char c, noeud *e)` qui construit l'arbre abstrait correspondant au résiduel de l'expression régulière `e` par le symbole `c`.
6. En déduire la fonction `int appartient (char *mot, noeud *e)` qui dit si `mot` appartient au langage de l'expression `e`.

2 Analyse d'un programme

Soit le programme Pascal suivant :

```
1  program mystere;
2  var a, b : integer;
3
4  function rec_mys(a, b: integer) : integer;
5  var c : integer;
6  begin
7     c := a mod b;
8     if c = 0 then rec_mys := b
9     else rec_mys := rec_mys(b, c);
10 end;
11
12 begin
13    a := read();
14    b := read();
15    if a > b then write (rec_mys(a,b))
16    else write (rec_mys(b,a));
17 end.
```

1. Donner le résultat de `rec_mys(12,8)`. Que fait ce programme ?
2. Dessiner un arbre abstrait correspondant aux lignes 6 à 10.
3. Traduire en code trois adresses les lignes 6 à 10.
4. Traduire en assembleur les lignes 6 à 10.

Annexe : liste partielle d'opérations MIPS

`lw`	reg, adresse	charge la valeur pointée par l'adresse (4 octets)
`sw`	reg, adresse	sauvegarde le registre à l'adresse
`li`	reg, valeur	met la valeur dans le registre
`la`	reg, adresse	copie l'adresse elle-même, comme un pointeur
`move`	reg, reg	copie le registre de droite vers celui de gauche
op	résultat, arg1, arg2	où *op* $\in$ {`add, sub, and, or, xor`}
opi	résultat, arg1, valeur	où *opi* $\in$ {`addi, andi, ori, xori`}
`mult`	arg1, arg2	met le produit dans `Hi` et `Lo`
`mfhi`	reg	récupère la valeur de `Hi`
`mflo`	reg	récupère la valeur de `Lo`
`div`	arg1, arg2	met le quotient dans `Lo`
		et le reste dans `Hi`
`j`	adresse	saute à l'adresse
`jal`	adresse	idem, enregistre l'adresse de retour dans `$ra`
`jr`	reg	saute à l'adresse contenue dans le registre
`beq`	arg1, arg2, adresse	saute si `arg1` = `arg2` ;
		même format pour {`bne, bgt, bge, blt, ble`},
		respectivement $\{\neq, >, \geq, <, \leq\}$.
`syscall`		effectue un appel système selon la valeur de `$v0` :
	si `$v0` = 1,	affiche l'entier `$a0`,
	si `$v0` = 4,	affiche une chaîne pointée par `$a0`,
	si `$v0` = 5,	lit un entier et le met dans `$v0`,
	si `$v0` = 10,	termine le programme.

Annexe B : liste des instructions du code 3 adresses

description	`op`	`arg1`	`arg2`	`var`
opération binaire	`add, ...`	ligne	ligne	
opération unaire	`not, negatif`	ligne		
	`lire`			
	`ecrire`	ligne		
charge une variable	`load`			variable
sauvegarde une variable	`store`	ligne		variable
charge depuis un tableau	`ltab`		indice	variable
sauvegarde dans un tableau	`stab`	ligne	indice	variable
charge une valeur directe	`loadimm`	valeur		
ajoute une valeur directe (utile pour les indices des tableaux)	`addimm`	ligne	valeur	
saut	`jump`		adresse	
saut conditionnel	`jsifaux`	ligne	adresse	
marque une expression comme argument du prochain `call`	`param`	ligne		
appelle une fonction	`call`			nom
entre dans une fonction	`entree`			nom
sort d'une fonction	`sortie`			nom