Partiel

Documents autorisés

Durée: 2h

9 novembre 2010

1 Manipulation d'automates

Q.1.1. Construire un automate reconnaissant le langage des mots sur $\{a, b\}$ comportant un nombre pair de a ou un nombre de b qui ne soit pas un multiple de 3.

Q.1.2. Eliminer les transitions ε de l'automate suivant :

		a	b	c	ε
$\rightarrow$	0			2	1
	1	0			2
$\leftarrow$	2		1		0

quel est le langage reconnu par cet automate ?

Q.1.3. Déterminiser l'automate suivant :

		a	b	c
$\rightarrow$	0	$\{1, 2\}$	1	2
	1	0	2	1
$\leftarrow$	2	1	0	1

Q.1.4. Minimiser l'automate suivant :

		a	b	c
$\rightarrow$	0	5	1	3
	1	6	4	2
$\leftarrow$	2	2	1	3
	3	2	4	0
	4	6	1	5
$\leftarrow$	5	5	4	0
	6	7	5	6
$\leftarrow$	7	3	0	5

2 Grammaire des tableaux HTML

La description d'un tableau en HTML commence par la balise ouvrante `<table>` et se termine par la balise fermante `</table>`. Le tableau lui-même est composé d'une suite de lignes. Chaque ligne débute par la balise ouvrante `<tr>` et se termine par la balise fermante `</tr>`. Une ligne est composée d'une suite de cellules, chaque cellule commence par la balise `<td>` et se termine par la balise fermante `</td>`. Une cellule peut contenir toute sorte de choses, dans notre cas, nous considèrerons qu'elles ne peuvent contenir qu'un chiffre, comme dans l'exemple de la figure 1 [1].

Q.2.1. Ecrire une grammaire G permettant de générer le langage des tableaux HTML. On considèrera que les balises (`<table>`, `<tr>`, `<td>`, `</td>`, `</tr>` et `</table>`) sont des symboles de l'alphabet terminal.

1. Les retours à la ligne sont là uniquement pour faciliter la lecture.

```
<table>
<tr><td>1</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td></td><td>5</td></tr>
<tr><td>6</td></tr>
</table>
```

1	2	3
4		5
6		

FIGURE 1 – Deux représentations d'un tableau

```
1  T  -> <table> LL </table>
2  LL -> L LL
3  LL -> epsilon
4  L  -> <tr> LC </tr>
5  LC -> C LC
6  LC -> epsilon
7  C  -> <td> N </td>
8  N  -> epsilon
9  N  -> 0
10 N  -> 1
...
18 N -> 9
```

Q.2.2. Ecrire une dérivation du tableau `<table><tr><td>1</td></tr></table>`.

```
T => <table>LL</table>
  => <table>L LL</table>
  => <table><tr>LC</tr> LL</table>
  => <table><tr>C LC</tr> LL</table>
  => <table><tr><td>N</td> LC</tr> LL</table>
  => <table><tr><td>1</td> LC</tr> LL</table>
  => <table><tr><td>1</td></tr> LL</table>
  => <table><tr><td>1</td></tr></table>
```

Q.2.3. Calculer les premiers et suivants des symboles de G puis construire la table $LL(1)$ lui correspondant. G est elle $LL(1)$?

```
PREM(T)={<table>}              SUIV(T)={$}
PREM(LL)={<tr>, epsilon}       SUIV(LL)={</table>}
PREM(L)={<tr>}                 SUIV(L)={<tr>, </table>}
PREM(LC)={<td>, epsilon}       SUIV(LC)={</tr>}
PREM(C)={<td>}                 SUIV(C)={<td>, </tr>}
PREM(N)={0,1,2,...,9,epsilon} SUIV(N)={</td>}
```

	<table>	<tr>	<td>	</td>	</tr>	</table>	0	1	$
T	1								
LL		2				3			
L		4							
LC			5		6				
C			7						
N				8			9	10	

Q.2.4. Effectuer l'anlalyse $LL(1)$ du tableau `<table><tr><td>1</td></tr></table>`.

```
(                          T$, <table><tr><td>1</td></tr></table>$, 1)
```

```
(          <table>LL</table>$, <table><tr><td>1</td></tr></table>$, 1)
(                LL</table>$,        <tr><td>1</td></tr></table>$, 1-2)
(              L LL</table>$,        <tr><td>1</td></tr></table>$, 1-2-4)
(      <tr>LC</tr>LL</table>$,        <tr><td>1</td></tr></table>$, 1-2-4)
(          LC</tr>LL</table>$,            <td>1</td></tr></table>$, 1-2-4-5)
(       C  LC</tr>LL</table>$,            <td>1</td></tr></table>$, 1-2-4-5-7)
(<td>N</td>LC</tr>LL</table>$,            <td>1</td></tr></table>$, 1-2-4-5-7)
(    N</td>LC</tr>LL</table>$,                1</td></tr></table>$, 1-2-4-5-7-10)
(    1</td>LC</tr>LL</table>$,                1</td></tr></table>$, 1-2-4-5-7-10)
(     </td>LC</tr>LL</table>$,                 </td></tr></table>$, 1-2-4-5-7-10)
(          LC</tr>LL</table>$,                     </tr></table>$, 1-2-4-5-7-10-6)
(            </tr>LL</table>$,                     </tr></table>$, 1-2-4-5-7-10-6)
(                 LL</table>$,                          </table>$, 1-2-4-5-7-10-6-3)
(                   </table>$,                          </table>$, 1-2-4-5-7-10-6-3)
(                           $,                                  $, 1-2-4-5-7-10-6-3)
```

3 Arbres binaires

A est un arbre binaire si A est l'arbre vide ou bien si A est un triplet $\langle R, G, D\rangle$ où R est un nœud appelé racine, G un arbre binaire appelé fils gauche de A et D un arbre binaire appelé fils droit de A. Un arbre binaire peut être représenté sous la forme d'un mot que l'on codera sous la forme d'une chaîne de carcactère tel que dans l'exemple suivant : `"<3,<2,<4,,>,<5,,>>,<1,,>>"`

Q.3.1. Ecrire une grammaire hors-contexte qui permet de générer tous les arbres binaires dont les nœuds dont étiquetés par un entier i, avec $0 \leq i \leq 9$.

```
A -> <N,A,A>
A -> epsilon
N -> 0
N -> 1
...
N -> 9
```

Q.3.2. Programmer en langage C un analyseur syntaxique pour la grammaire ci-dessus. Vous pourrez faire l'hypothèse de l'existence d'un analyseur lexical implémenté sous la forme d'une fonction `yylex()`.

```
#include<stdio.h>
#include<stdlib.h>

int cc;
FILE *yyin;

int yylex()
{
  cc = getc(yyin);
  while((cc == ' ') || (cc == '\n')) cc = getc(yyin);
  if(feof(yyin)) return;
  if((cc >= '0') && (cc <= '9')){ return cc;}
  else if(cc == '<') { return '<';}
  else if(cc == '>') { return '>';}
  else if(cc == ',') { return ',';}
  else {
    fprintf(stderr, "caractère %c non reconnu\n", cc);
  }
}

void error(void)
{
  fprintf(stderr, "erreur de syntaxe\n");
```

```
  exit(1);
}

int N(void)
{
  if((cc >= '0') && (cc <= '9')){
    cc = yylex();
    return;
  }
  error();
}


int A(void)
{
  if((cc == ',') || (cc == '>')) return;
  if(cc == '<'){
    cc = yylex();
    N();
    if(cc == ','){
      cc = yylex();
      A();
      if(cc == ','){
cc = yylex();
A();
if(cc == '>'){
  cc = yylex();
  return;
}
      }
    }
  }
  error();
}

int main(void)
{
  yyin = stdin;
  cc = yylex();
  A();
  fprintf(stdout, "analyse réussie\n");
}
```

Q.3.3. La hauteur $h(x)$ d'un nœud x est définie récursivement de la manière suivante :
- $h(x) = 0$ si x est la racine de l'arbre
- $h(x) = 1 + h(pere(x))$ si x n'est pas la racine de l'arbre

La hauteur $H(A)$ d'un arbre A est égale à la hauteur du nœud le plus « profond » de A.
C'est-à-dire : $H(A) = \max h(x)$ pour tout nœud x de A
modifier l'analyseur syntaxique de la question précédente de manière à calculer la hauteur de l'arbre analysé.

```
int A(int p)
{
  int pg, pd;
  if((cc == ',') || (cc == '>')) return p;
  if(cc == '<'){
    cc = yylex();
    N();
    if(cc == ','){
```

```
      cc = yylex();
      pg = A(p+1);
      if(cc == ','){
cc = yylex();
pd = A(p+1);
if(cc == '>'){
  cc = yylex();
  return max(pg, pd);
}
      }
    }
  }
  error();
}

int main(void)
{
  int l;
  yyin = stdin;
  cc = yylex();
  l = A(0);
  fprintf(stdout, "analyse réussie prof = %d\n", l);
}
```

Q.3.4. Un arbre binaire de recherche est un arbre binaire tel que, pour tout nœud v de l'arbre :
- les éléments de tous les nœuds du sous-arbre gauche de v sont inférieurs ou égaux à l'élément contenu dans v ;
- les éléments de tous les nœuds du sous-arbre droit de v sont supérieurs à l'élément contenu dans v

modifier l'analyseur syntaxique de manière à vérifier si l'arbre analysé est un arbre binaire de recherche.

```
#include<stdio.h>
#include<stdlib.h>
#define ARBRE_VIDE -1
int cc;
FILE *yyin;

int yylex()
{
  cc = getc(yyin);
  while((cc == ' ') || (cc == '\n')) cc = getc(yyin);
  if(feof(yyin)) return;
  if((cc >= '0') && (cc <= '9')){ return cc;}
  else if(cc == '<') { return '<';}
  else if(cc == '>') { return '>';}
  else if(cc == ',') { return ',';}
  else {
    fprintf(stderr, "caractère %c non reconnu\n", cc);
  }
}

void error(void)
{
  fprintf(stderr, "erreur de syntaxe\n");
  exit(1);
}

int N(void)
```

```
{
  int val;
  if((cc >= '0') && (cc <= '9')){
    val = cc;
    cc = yylex();
    return val;
  }
  error();
}

int A(int *min, int *max)
{
  int rac, min_g, min_d, max_g, max_d;
  if((cc == ',') || (cc == '>')){
    return ARBRE_VIDE;
  }
  if(cc == '<'){
    cc = yylex();
    rac = N();
    if(cc == ','){
      cc = yylex();
      if(A(&min_g, &max_g)== ARBRE_VIDE)
*min = rac;
      else
if(max_g <= rac)
  *min = min_g;
else{
  fprintf(stderr, "ce n'est pas un arbre binaire de recherche\n");
  exit(1);
}
      if(cc == ','){
cc = yylex();
if(A(&min_d, &max_d) == ARBRE_VIDE)
  *max = rac;
else
  if(min_d > rac)
    *max = max_d;
  else{
    fprintf(stderr, "ce n'est pas un arbre binaire de recherche\n");
    exit(1);
  }
if(cc == '>'){
  cc = yylex();
  return;
}
      }
    }
  }
  error();
}



int main(void)
{
  int min, max;
  yyin = stdin;
  cc = yylex();
```

```
  A(&min, &max);
  fprintf(stdout, "analyse réussie\n");
  fprintf(stdout, "c'est un arbre binaire de recherche\n");
}
```