Examen de Compilation

Licence Informatique 3

3 Mai 2013

Le sujet est composé de 4 exercices indépendants. Aucun document n'est autorisé. Durée de l'épreuve : 0x78 minutes. La note finale tiendra trés largement compte de la présentation.

1 Automate

Dans l'ensemble $\{0,1\}^*$ un mot binaire $x_n x_{n-1} \dots x_1$ représente la valeur entière $\sum_{i=1}^{n} x_i 2^{i-1}$. Conformément à l'usage un nombre binaire est un mot commençant par la lettre 1 ou bien égal à 0. Contruire les automates pour reconnaitre :

1. le langage des mots de valeur multiple de trois.
2. le langage des mots $0 + 1\{0,1\}^*$.
3. le langage des nombres multiple de 3.

2 Théorie des langages

Dans l'ensemble $\{a,b\}^*$, on condidère le langage X des mots contenant un nombre de a supérieur ou égal à celui de b.

1. Donner la définition de l'équivalence de Nérode entre deux états d'un automate.
2. Montrer que X n'est pas régulier. Indication : étudier l'équivalence de Nérode entre les états q_i (i entier) obtenus par la lecture du mot a^i à partir de l'état initial.

3 Ambiguité

On considère la grammaire

$$\begin{array}{lcl} \text{X} & \longrightarrow & \text{a X} \\ \text{X} & \longrightarrow & \text{a X b} \\ \text{X} & \longrightarrow & \epsilon \end{array}$$

1. Décrire le langage des mots reconnus.
2. Montrer que la grammaire est ambiguë.
3. Proposer une sémantique pour fixer l'ambiguité.
4. En déduire une grammaire non ambiguë.

4 flex-bison

Le jeu d'instruction d'une machine similaire à MILXI est partiellement décrit ci-dessous. Un compilateur est implanté au travers des codes `scan.l` et `parse.y`, il produit du code d'assemblage à partir d'un langage évolué.

READ x	mem[x] := lecture	WRITE x	écrire mem[x]
LOAD x	ACC := mem[x]	STO x	mem[x] := ACC
ADD x	ACC += mem[x]	MUL x	ACC *=mem[x]
CONST x	ACC := x	STOP	arrêt

1. Préciser les phases de compilation effectivement présentes dans ces codes.
2. Donner un `makefile` pour compiler une commande `minias` à partir de ces sources.
3. Quel sera le résultat de `./minias < prog.as` où `prog.as` est un fichier texte dont le contenu, écrit ici sur trois colonnes, est :

```
var x                   input x                 output z
var y                   input y
var z                   z := (x+y) * (x+y)
```

4. Quelles modifications faut-il effectuer pour gérer une d'arrêt STOP.
5. Quelles modifications faut-il effectuer pour gérer les constantes.

scan.l

```
%{
#include "parse.h"

int value = 0;
typedef struct _symb_ {
  int    val;
  char* key;
  struct _symb_ * next;
} enrs, *symb;
symb ts   = NULL;
int insert( char * k )
  {
    symb x;
    x = ts;
    while ( x ) {
if ( strcmp( x->key, k ) == 0 )
     return x->val;
     x = x -> next;
     }
    x = malloc( sizeof( enrs) );
    x->key = strdup( k );
    x->next = ts;
    x->val = value;
    value++;
    ts = x;
    return x->val;
  }

%}

%%
":="          return AFF;
"+"           return ADD;
";"           return NL;
"*"           return MUL;
"("           return PG;
")"           return PD;
"var"         return VAR;
"input"       return IN;
"output"      return OUT;
[a-z]+     {  yylval = insert( yytext );
              return ID;
           }
[0-9]+        return NB;
\n            return NL;
.             ;
%%
```

parse.y

```
%{
#include <stdio.h>
int used[64] = {0};
int getreg( void ) {
  int r = 0;
  while ( used[r] ) r++;
  used[ r ] = 1;
  return r;
}
void ungetreg( int n ) {
 if ( used[n] == 1 )
        used[n] = 0;
}
extern int yylex( void );
int yyerror( char *msg );

%}

%token ID NB VAR NL AFF
%token IN OUT PG PD
%left  MUL
%left  ADD

%%

PROG  : DECL LIST
DECL  : VAR ID NL
   { used[$2] = 2;} DECL
      | /* empty */
      ;
LIST  : INSTR NL LIST
      | /* empty */
      ;
INSTR : ID AFF EXP  {
   printf("\nLOAD  %d", $3);
   printf("\nSTO   %d", $1);
   ungetreg( $3 );
    }
   | IN  ID   {
     printf("\nREAD  %d", $2);
    }
   | OUT ID   {
     printf("\nWRITE %d", $2);
    }
         ;
EXP : EXP ADD EXP {
      printf("\nLOAD  %d", $1);
      printf("\nADD   %d", $3);
      ungetreg( $1 );
      ungetreg( $3 );
      $$ = getreg(  );
      printf("\nSTO   %d", $$ );
     }
    | EXP MUL EXP {
      printf("\nLOAD  %d", $1);
      printf("\nMUL   %d", $3);
      ungetreg( $1 );
      ungetreg( $3 );
      $$ = getreg(  );
      printf("\nSTO   %d", $$ );
     }
    | PG  EXP  PD  { $$ = $2;}
    | ID  { $$ = $1;}
    ;

%%

int main( void )
{
yyparse();
return 0;
}

int yyerror(char *msg) {
printf("%s", msg);
return 0;
}
```