Département d'Informatique
MASTER IFA 1ère année
Vendredi 9 janvier 2009

Corrigé du contrôle terminal de Compilation

I Analyse syntaxique

1. table d'analyse LR(1) : (les règles sont numérotées selon l'ordre d'apparition dans l'énoncé de la grammaire)

État	`=`	`*`	`id`	`$`	L	R	S
0		s4	s5		2	3	1
1				acc			
2	s6			r5			
3				r2			
4		s4	s5		8	7	
5	r4			r4			
6		s11	s12		10	9	
7	r3			r3			
8	r5			r5			
9				r1			
10				r5			
11		s11	s12		10	13	
12				r4			
13				r3			

2. table d'analyse LaLR(1) :

État	`=`	`*`	`id`	`$`	L	R	S
0		s4	s5		2	3	1
1				acc			
2	s6			r5			
3				r2			
4		s4	s5		8	7	
5	r4			r4			
6		s4	s5		8	9	
7	r3			r3			
8	r5			r5			
9				r1			

3. La grammaire est LR(1) et LaLR(1) puisque les tables correspondantes contiennent au plus un élément dans chaque case.

II Traduction dirigée par la syntaxe

1 Affectations simultanées

1. Grammaire :

$$
\begin{array}{lcl}
\mathrm{A} & \longrightarrow & \texttt{(}\ \mathrm{L}\ \texttt{)}\ \texttt{:=}\ \texttt{(}\ \mathrm{R}\ \texttt{)} \\
\mathrm{L} & \longrightarrow & \texttt{id} \mid \mathrm{L}\ \texttt{,}\ \texttt{id} \\
\mathrm{R} & \longrightarrow & \texttt{E} \mid \mathrm{R}\ \texttt{,}\ \texttt{E}
\end{array}
$$

2. Schéma de traduction :

A ⟶ `( L ) := ( R )`
{ pour i=1 à longueur(R.temp) faire
gencode(ieme(i,R.temp) := ième(i,R.ptr)) ;
pour i=1 à longueur(R.temp) faire
gencode(ieme(i,L.ptr := ième(i,R.temp)) ; }

L ⟶ `id`
{ L.ptr = creerliste(id.ptr) }

L ⟶ $L^{(1)}$ `, id`
{ L.ptr = ajoute($L^{(1)}$.ptr,id.ptr) }

R ⟶ `E`
{ R.temp = creeliste(newtemp()) ;
R.ptr = creeliste(E.ptr) }

R ⟶ $R^{(1)}$ `, E`
{ R.temp = ajoute($R^{(1)}$.temp,newtemp()) ;
R.ptr = ajoute($R^{(1)}$.ptr,E.ptr) }

Fonctions utilisées :
- longueur([e1,e2,...,en]) = n
- ième([e1,e2,...,en],i) = ei
- ajoute([e1,e2,...,en],e) = [e1,e2,...,en,e]
- creeliste(e) = [e]

2 Commandes gardées de Djikstra

S ⟶ `(` *M* C `)*`
{ complete(C.next, M.quad) ;
S.next = creeliste() }

S ⟶ `begin` L `end`
{ S.next = L.next }

S ⟶ A
{ S.next = creeliste() }

L ⟶ $L^{(1)}$ `;` *M* S
{ complete($L^{(1)}$.next, M.quad) ;
L.next = S.next }

L ⟶ S
{ L.next = S.next }

C ⟶ B `->` *M* S
{ complete(B.true, M.quad) ;
complete(B.false, nextquad+1) ;
complete(S.next, nextquad) ;
C.next = nextquad ;
gencode(goto _) }

C ⟶ $C^{(1)}$ `[]` B `->` *M* S
{ complete(B.true, M.quad) ;
complete(B.false, nextquad+1) ;
complete(S.next, nextquad) ;
C.next = concat($C^{(1)}$.next,nextquad) ;
gencode(goto _) }

M ⟶ ϵ
{ M.quad = nextquad }

III Optimisations

1. code intermédiaire :
   T1 := y
   T2 := x
   x := T1
   y := T2

   code optimisé :
   T2 := x
   x := y
   y := T2

2. code intermédiaire :
   T1 := z
   T2 := x
   T3 := y
   x := T1
   y := T2
   z := T3

   code optimisé :
   T2 := x
   T3 := y
   x := z
   y := T2
   z := T3

3. code intermédiaire :
   T1 := y
   T2 := z
   T3 := x
   x := T1
   y := T2
   z := T3

   code optimisé :
   T3 := x
   x := y
   y := z
   z := T3