Compilation TD n° 8

Devan SOHIER

1 Quadruplets

Le code intermédiaire, que l'on peut considérer comme un assembleur générique, est constitué d'une séquence d'opérations élémentaire (au sens où elles ne peuvent être directement chaînées : $y+2*x$ doit être décomposé en $t1 \leftarrow 2$, $t2 \leftarrow t1*x$ et $y+t2$). Les opérations que nous avons le droit d'utiliser peuvent être décrites par des quadruplets, dont le premier élément décrit la nature, et les trois autres les adresses des opérandes et du résultat (qu'ils s'agissent de variables ou d'instructions).

- (op, x, y, z) avec *op* un opérateur arithmétique ($+$, $-$, $*$, $/$, $\%$, ...) et x, y, et z les adresses de variables représente $z \leftarrow x \ op \ y$;
- $(:=, x, , y)$
- $(cst, v, , x)$ avec v une valeur numérique (et non une adresse) et x l'adresse d'une variable représente $x \leftarrow v$;
- (op, x, y, z) avec *op* un opérateur de comparaison ($<$, $>$, $<=$, $>=$, $==$, ...), x et y, les adresses de deux variables et z l'adresse d'une instruction représente `Si` x *op* y `alors aller à` z ;
- $(goto, , , z)$ avec z l'adresse d'une instruction représente `aller à` z.

Ces quelques opérations peuvent être enrichies d'opérations différentes implémentées par les processeurs. Néanmoins, elles suffisent à produire un langage Turing-complet. Ainsi le programme

```
Tantque x<3+4*2 faire
   x:= x-1
FinTantque
```

peut devenir, par exemple

```
1 cst, 4, , t1
2 cst, 2, , t2
3 *, t1, t2, t3
/* t3 vaut 4*2*/
4 cst, 3, , t4
5 +, t4, t3, t5
/* t5 vaut 3+4*2 */
6 >=, x, t5, 10
/* si x>=t5 aller à l'instruction 10 (sinon, aller à l'instruction suivante, 7)*/
7 cst, 1, , t6
8 -, x, t6, x
9 goto, , , 6
```

2 Génération de code pour les conditions

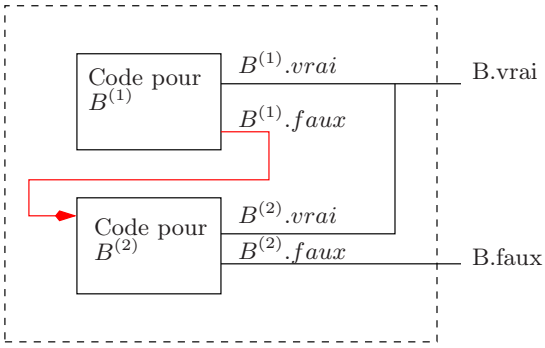
Les conditions jouent un rôle particulier car ils servent, dans les conditionnelles et les boucles `Tantque`, à définir les branchements du code.

Il est nécessaire d'associer au non-terminal `B` deux attributs, `B.vrai`, `B.faux` : deux listes qui contiennent des numéros de quadruplets concernant des sauts à compléter.

La fonction `backtrack(l: liste_quad, n: num_quad)` : complète les quadruplets de la liste `l` avec l'adresse de saut `n`.

La fonction `nextquad()` renvoie le numéro du prochain quadruplet qui sera généré.

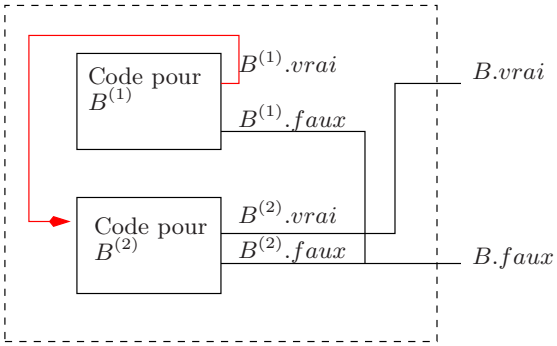
2.1 $B \to B \vee B$



On a besoin de conserver l'adresse du quadruplet inconnu dans un attribut : on introduit le non-terminal M pour ce faire.

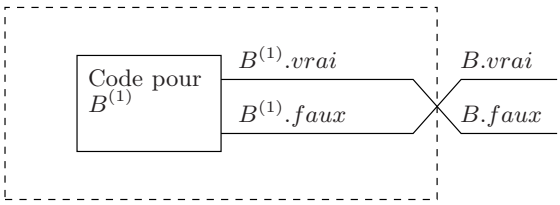
$B \to B \vee MB$ $\{backtrack(B^{(1)}.faux, M.quad);$
$B.vrai \leftarrow fusion(B^{(1)}.vrai, B^{(2)}.vrai); B.faux \leftarrow B^{(2)}.faux\}$
$M \to \varepsilon$ $\{M.quad \leftarrow nextquad()\}$

2.2 $B \to B \wedge B$



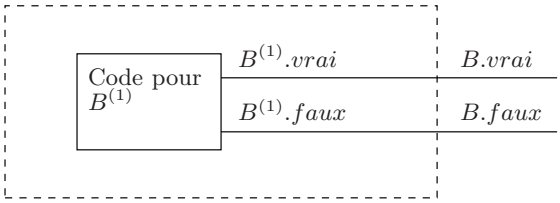
$B \to B \wedge MB$ $\{backtrack(B^{(1)}.vrai, M.quad);$
$B_1.faux \leftarrow fusion(B^{(1)}.faux, B^{(2)}.faux); B.vrai \leftarrow B^{(2)}.vrai\}$
$M \to \varepsilon$ $\{M.quad \leftarrow nextquad()\}$

2.3 $B \to \neg B$



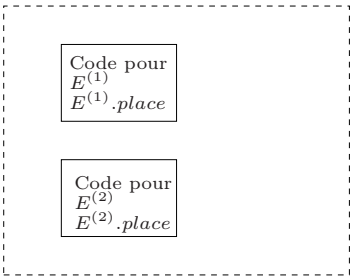
$B \to \neg B$ $\{B.vrai \leftarrow B^{(1)}.faux\,;\, B.faux \leftarrow B^{(1)}.vrai\}$

2.4 $B \to (B)$



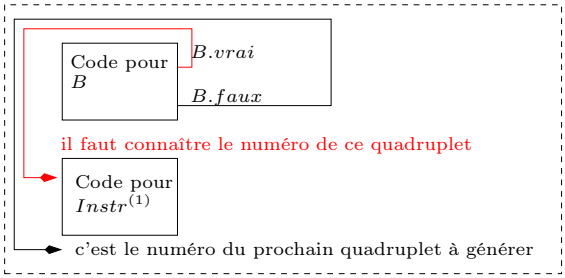
$B \to (B)$ $\{B.vrai \leftarrow B^{(1)}.vrai\,;\, B.faux \leftarrow B^{(1)}.faux\}$

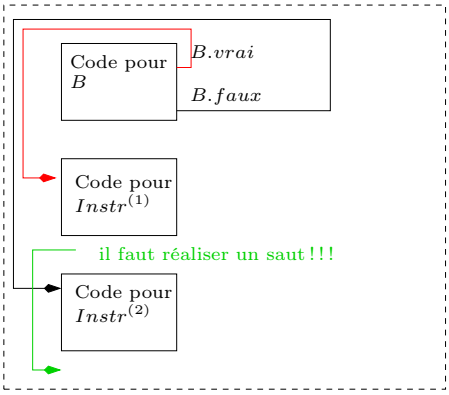
2.5 $B \to E\ relop\ E$



$B \to E\ relop\ E$ { $B.vrai \leftarrow list(nextquad())$
$gen(relop.lex, E^{(1)}.place, E^{(2)}.place, ?)$
$B.faux \leftarrow list(nextquad())$
$gen(goto, \ , \ , ?)$}

3 Les conditionnelles





$Instr \to$ **si** B **alors** $M\ LInstr$ **finsi** {$backtrack(B.vrai, M.quad)$;
$backtrack(B.faux, nextquad())$}}
| **si** B **alors** $M\ LInstr$ **sinon** $N\ LInstr$ **finsi**
{$backtrack(B.vrai, M.quad)$
$backtrack(B.faux, N.quad)$
$backtrack(list(N.quad - 1), nextquad())$}
| `id` :=E {gen(:=, $E^{(1)}.place$, , @`id`.lex)}

$LInstr \to Instr\ LInstr$
| ε
$M \to \varepsilon$ {$M.quad \leftarrow nextquad()$}
$N \to \varepsilon$ {$gen(goto, \ , \ , \ ?)$; $N.quad \leftarrow nextquad()$}

E et B sont traités comme des terminaux dans cet exercice, à l'exception de la question 5 : dans cette question, lorsque vous aurez à décaler sur E ou B, vous devrez exécuter l'analyseur correspondant, dont la table vous est donnée en annexe.

	`si`	B	`alors`	`sinon`	`finsi`	`id`	`:=`	E	$\dashv$	$Instr$	$LInstr$	M	N
0	d2					d3				1			
1									accept				
2		d4											
3							d5						
4			d6										
5								d7					
6	r6			r6	r6	r6						8	
7	r3			r3	r3	r3			r3				
8	d2			r5	r5	d3				10	9		
9				d12	d11								
10	d2			r5	r5	d3				10	13		
11	r1			r1	r1	r1			r1				
12	r7				r7	r7							14
13	r4			r4	r4	r4							
14	d2			r5	r5	d3				10	15		
15					d16								
16				r2	r2				r2				

1. Exécutez l'analyseur SLR sur le programme ci-dessous (vous donnerez la pile et le flot d'entrée à chaque étape, et préciserez les quadruplets générés et/ou complétés au fur et à mesure ; lorsque vous devrez faire un décalage sur E ou sur B, vous exécuterez l'analyseur correspondant, en remplaçant $\dashv$ par $Suivant(E)$, resp. $Suivant(B)$, qu'il vous faudra donc calculer).

   ```
   si(n>3 ∧ n*n<51) alors
      n := n*n
   finsi
   ```

2. Dessinez chacun des blocs dans la liste de quadruplets, ainsi que les adresses laissées vides $B.vrai$ et $B.faux$, les N et M réduits à ε, etc.
3. Calculez les neuf premiers items LR(0) de cette grammaire.
4. Justifiez la résolution les conflits présents dans la table d'analyse des expressions booléennes (fournie en annexe ; on donne la priorité au $\wedge$ sur le $\vee$; le $\neg$ est le plus prioritaire des opérateurs).
5. Ecrivez une grammaire pour le `Tantque`
6. Représentez les blocs nécessaires à décorer cette grammaire.
7. Décorez la.
8. Calculez sa table SLR(0).
9. Exécutez l'analyseur sur

   ```
   tantque(n>3 ∧ n*n<51) faire
      n := n*n
   fintantque
   ```

10. Dessinez chacun des blocs dans la liste de quadruplets, ainsi que les adresses laissées vides $B.vrai$ et $B.faux$, les M réduits à ε, etc.

Annexe — grammaire et table SLR pour les expressions arithmétiques

$E \to E + E$ {$E.place$=`newtmp()` ; `gen(+,` $E^{(1)}.place$`,` $E^{(2)}.place$`,` $E.place$`)`}
| $E * E$ {$E.place$=`newtmp()` ; `gen(*,` $E^{(1)}.place$`,` $E^{(2)}.place$`,` $E.place$`)`}
| (E) {$E.place = E^{(1)}.place$}
| `id` {$E.place$=@`id`.lex}
| `cst` {$E.place$=`newtmp()` ; `gen(cst,` `cst.`val`,` `,` $E.place$`)`}

	+	*	(	)	id	cst	⊣	E
0			d2		d3	d4		1
1	d5	d6					acc	
2			d2		d3	d4		7
3	r4	r4		r4			r4	
4	r5	r5		r5			r5	
5			d2		d3	d4		8
6			d2		d3	d4		9
7	d5	d6		d10				
8	(d5)/r1	d6/(r1)		r1			r1	
9	(d5)/r2	(d6)/r2		r2			r2	
10	r3	r3		r3			r3	

$I_0 = \{E' \to .E \dashv, E \to .E + E, E \to .E * E, E \to .(E), E \to .\texttt{id}, E \to .\texttt{cst}\}$
$I_1 = \{E' \to E. \dashv, E \to E. + E, E \to E. * E\}$
$I_2 = \{E \to (.E), E \to .E + E, E \to .E * E, E \to .(E), E \to .\texttt{id}, E \to .\texttt{cst}\}$
$I_3 = \{E \to \texttt{id}.\}$
$I_4 = \{E \to \texttt{cst}.\}$
$I_5 = \{E \to E + .E, E \to .E + E, E \to .E * E, E \to .(E), E \to .\texttt{id}, E \to .\texttt{cst}\}$
$I_6 = \{E \to E * .E, E \to .E + E, E \to .E * E, E \to .(E), E \to .\texttt{id}, E \to .\texttt{cst}\}$
$I_7 = \{E \to (E.), E \to E. + E, E \to E. * E\}$
$I_8 = \{E \to E + E., E \to E. + E, E \to E. * E\}$
$I_9 = \{E \to E * E., E \to E. + E, E \to E. * E\}$
$I_{10} = \{E \to (E).\}$

Annexe — grammaire et table SLR pour les expressions booléennes

$B \to$ $B \vee N\ B$ $\{B.vrai{=}B^{(1)}.vrai \cup B^{(2)}.vrai\,;\ B.faux{=}B^{(2)}.faux\,;$ `backtrack`$(B^{(1)}.faux,\ N.quad)\}$

| $B \wedge N\ B$ $\{B.faux{=}B^{(1)}.faux \cup B^{(2)}.faux\,;\ B.vrai = B^{(2)}.vrai\,;$ `backtrack`$(B^{(1)}.vrai,\ N.quad)\}$

| $\neg B$ $\{B.vrai{=}B^{(1)}.faux\,;\ B.faux = B^{(1)}.vrai\}$

| (B) $\{B.vrai{=}B^{(1)}.vrai\,;\ B.faux = B^{(1)}.faux\}$

| E `relop` E $\{B.vrai{=}list($`nextquad()`$)\,;$ `gen(relop.`lex`,`$E^{(1)}.place$`,`$E^{(2)}.place$`,?)`$\,;$ $B.faux{=}list($`nextquad()`$)\,;$`gen(goto,,,?)`$\}$

$N \to$ ε $\{N.quad :=$`nextquad()`$\}$

`relop` désigne un opérateur de comparaison ($=$, $<$, $>$, $\leq$, $\geq$), et `relop`.lex est la chaîne correspondant à cet opérateur.

	$\vee$	$\wedge$	$\neg$	(	)	`relop`	E	$\dashv$	B	N
0			d2	d3			d4		1	
1	d5	d6						acc		
2			d2	d3			d4		7	
3			d2	d3			d4		8	
4						d9				
5			r6	r6			r6			10
6			r6	r6			r6			11
7	(d5)/r3	(d6)/r3			r3			r3		
8	d5	d6			d12					
9							d13			
10			d2	d3			d4		14	
11			d2	d3			d4		15	
12	r4	r4			r4			r4		
13	r5	r5			r5			r5		
14	(d5)/r1	d6/(r1)			r1			r1		
15	(d5)/r2	(d6)/r2			r2			r2		