NOMBRE:

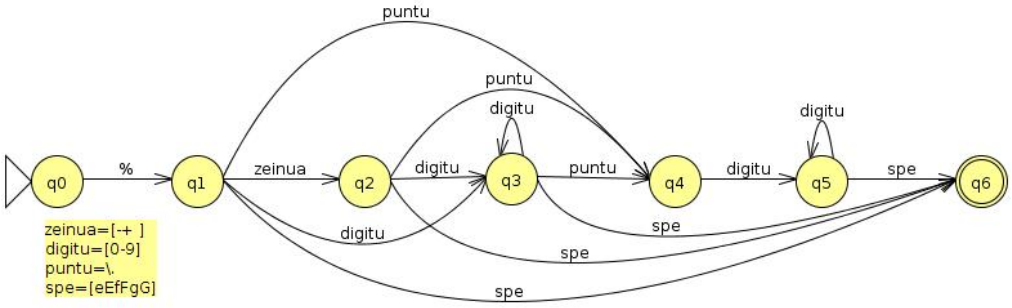
1.

```
\<!([^/-]|\-?\-?[^/-])*\-?\-?\/\>   Menos óptima

\<!(\-?\-?[^/-])*\-?\-?\/\>

\<!\-?\-?([^/-]\-?\-?)*\/\>
```

2.



3. **E → T ^ E | T** **T → T . F | F** **F → id | F ( ) | * F | (E)**

Primero(E) = Primero(T) = Primero(F) = **{ id * (}**
Siguiente(E) = **{ #) }**
Siguiente(T) = **{ ^ . #) }**
Siguiente(F) = **{ (^ . #) }**

Condicioneas LL(1) : Para cada no terminal con reglas del tipo A → α | β

1. condición: Primero(α) ∩ Primero(β) = ∅
   E no cumple:
   Primero(T ^ E) ∩ Primero(T) = **{ id * (}** ≠ ∅
   T no cumple:
   Primero(T . F) ∩ Primero(F) = **{ id * (}** ≠ ∅
   F no cumple:
   Primero(**id**) ∩ Primero(F ())= **{id}** ≠ ∅
   Primero(**id**) ∩ Primero(***** F) = ∅
   Primero(**id**) ∩ Primero((E)) = ∅
   Primero(***** F) ∩ Primero(F ()) = **{*}** ≠ ∅
   Primero(***** F) ∩ Primero((E)) = ∅
   Primero((E)) ∩ Primero(F()) = **{(}** ≠ ∅
2. condición: si α $\overset{*}{\Rightarrow}$ ξ (o β $\overset{*}{\Rightarrow}$ ξ), entonces Primero(A) ∩ Siguiente(A) = ∅
   No se da esta condición.

Por lo tanto, no se cumplen las condiciones LL(1) .
Para cambiar la gramática:

- **No es ambigua**
- **Quitar la recursividad a la izquierda (T, F):**
  T → F T'
  T' → . F T' | ξ
  F → id F' | * F F' | (E) F'
  F' → ( ) F' | ξ
- **Quitar factores comunes (E) :**
  E → T E'
  E' → ^ E | ξ
  Gramática Final:
  E → T E'
  E' → ^ E | ξ
  T → F T'
  T' → . F T' | ξ
  F → id F' | * F F' | (E) F'
  F' → ( ) F' | ξ

4. **E → T E'** **T → id**
E' → + T E' **| – (E)**
| – T E'
| ξ

COMPLETA LA TABLA:

	#	(	)	id	+	–
E				**E → T E'**		**E → T E'**
E'	**E' →ξ**		**E' →ξ**		**E' → + T E'**	**E' → – T E'**
T				**T → id**		**T → – (E)**

Analiza esta entrada: **id + id – – id**

Pila	Entrada	Acción
E #	**id + id – – id #**	**E → T E'**
T E' #	**id + id – – id #**	**T → id**
id E' #	**id + id – – id #**	**match id**
E' #	**+ id – – id #**	**E' → + T E'**
+ T E' #	**+ id – – id #**	**match +**
T E' #	**id – – id #**	**T → id**
id E' #	**id – – id #**	**match id**
E' #	**– – id #**	**E' → – T E'**
– T E' #	**– – id #**	**match –**
T E' #	**– id #**	**T → – (E)**
– (E) E' #	**– id #**	**match –**
(E) E' #	**id #**	**ERROR**

Onartzen al du gramatikak sarrera? NO

5. (1) D → T **id** L
(2) T → int
(3) T → float
(4) L → **,** **id** L
(5) L → **;**

ESTADO	acción						goto		
	int	float	id	,	;	$	D	T	L
0	s3	s4					1	2	
1						aceptar			
2			s5						
3			r2						
4			r3						
5				s7	s8				6
6						r1			
7			s9						
8						r5			
9				s7	s8				10
10						r4			

Analiza esta entrada: int id , id , id ;

Pila	**Entrada**	**Acción**
0	**int id , id , id ; $**	**desplazar 3**
0 int 3	**id , id , id ; $**	T → **int + goto(0,T)**
0 T 2	**id , id , id ; $**	**desplazar 5**
0 T 2 id 5	**, id , id ; $**	**desplazar 7**
0 T 2 id 5 , 7	**id , id ; $**	**desplazar 9**
0 T 2 id 5 , 7 id 9	**, id ; $**	**desplazar 7**
0 T 2 id 5 , 7 id 9 , 7	**id ; $**	**desplazar 9**
0 T 2 id 5 , 7 id 9 , 7 id 9	**; $**	**desplazar 8**
0 T 2 id 5 , 7 id 9 , 7 id 9 ; 8	**$**	L → **;** + **goto(9,L)**
0 T 2 id 5 , 7 id 9 , 7 id 9 L 10	**$**	L → **,** **id** L + **goto(9,L)**
0 T 2 id 5 , 7 id 9 L 10	**$**	L → **,** **id** L + **goto(5,L)**
0 T 2 id 5 L 6	**$**	D → T **id** L + **goto(0,D)**
0 D 1	**$**	**ACEPTAR**

Esta entrada es generada por la gramática? SI

6.

1. a := a * a **(a, v,3)**
2. b := b * b **(b, v,3)**
3. t1 := a + b **(t1, v,4) (a, v,4) (b, v,8)**
4. t2 := t1 – a **(t2, v,5) (t1, v,5) (a, v,6)**
5. t3 := t1 + t2 **(t3, v,6) (t1, v,7) (t2, m)**
6. t2 := a * t3 **(t2, m) (a, v,8) (t3, m)**
7. c := t1 **(c, v,8) (t1, m)**

	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
a	v8		V6		V4	V3		V1
b	v8					V3	V2	
c	v8	M						
t1	M	v7		V5	V4	m		
t2	M		M	V5	M			
t3	m		V6	M				

Hay código muerto? Porqué?

Sí, la intrucción 6 es código muerto porque se define un valor que no se usa posteriormente. (Por consiguiente, la 5 también y la 4 también).

7.

Instrucción	**Código**	**Descriptor de Registros**	**Descriptor de direcciones**
		R0:vacío R1:vacío	a,b,c en memoria
1)t1:= a - b (t1,v,2) (a,m) (b,v,5)	MOV a, R0 SUB b, R0	R0:t1 R1:vacío	a,b,c en memoria t1 en R0
2)a:= t1 - c (a,v,3) (t1,m) (c,v,3)	SUB c, R0	R0:a R1:vacío	b,c en memoria a en R0
3)t2:= a - c (t2,v,4) (a,v,5) (c,m)	MOV R0, R1 SUB c, R1	R0:a R1:t2	b,c en memoria a en R0 t2 en R1
4)c:= t2 (c,v,5) (t2,m)		R0:a R1:c	b en memoria a en R0 c en R1
	MOV R0, a MOV R1, c		a,b,c en mem.

8.

```
struct expresionstruct {        %union {                    %type <expr> expr
string nombre ;                 expresionstruct *expr ;     %type <number> M
vector<int> trues ;             int number ;                %type <lisEnt> N
vector<int> falses ;            vector<int> *lisEnt ;
};                              }
```

```
stmt : RIF expr
        RTHEN M stmts N
        RELSE M stmts RENDIF M
        {codigo.completarInstrucciones($2->trues,$4) ;
         codigo.completarInstrucciones($2->falses,$8) ;
         codigo.completarInstrucciones(*$6, $11) ;
        delete $2 ;
        delete $6;}
        ;

M:      { $$ = codigo.obtenRef() ; }
        ;

N:      { $$ = new vector<int>;
        $$->push_back(codigo.obtenRef()) ;
        codigo.anadirInstruccion("goto"); }
        ;
```

9.

S → while id in range E .. E

{ añadir-Inst (if || id.nom || < || E1.nom || goto || error-semántico);
añadir-Inst (if || id.nom || > || E2.nom || goto || error-semántico);}

M
{ añadir-Inst (if || id.nom || < || E1.nom || goto);
añadir-Inst (if || id.nom || > || E2.nom || goto);}
begin lista_sentencias end;
M
{añadir-Inst(goto || M1.ref);
Completar(ini.lista(M1.ref), M2.ref+1);
Completar(ini-lista(M1.ref+1), M2.ref+1);}

La comprobación (en amarillo) se realizará en tiempo de ejecución, cuando se conozcan los valores de E1 y E2.