Definición S-atribuida (1): expresiones aritméticas

- *Sólo se utilizan atributos **sintetizados**.*

	Regla de Producción	*Acción semántica*
1	**S → E**	Imprimir (E.valor)
2	**E → T**	E.valor = T.valor
3	**E → E_1 + T**	E.valor = E_1.valor + T.valor
4	**T → F**	T.valor = F.valor
5	**T → T_1 * F**	T.valor = T_1.valor * F.valor
6	**F → (E)**	F.valor = E.valor
7	**F →** número	F.valor = número.valor_léxico
8	**F →** identificador	F.valor = identificador. valor

Definición L-atribuida (2): declaración de variables en C

- *Definición Basada en la Sintaxis con atributos **sintetizados** y **heredados***

Regla de Producción	***Acción semántica***
1 D → T L **;**	L.h = T. tipo
2 T → **int**	T.tipo = entero
3 T → **float**	T.tipo = real
4 L → **identificador**	Añadir_tipo(identificador.tabla_símbolos,L.h)
5 L → L_1 **, identificador**	L_1.h = L.h Añadir_tipo(identificador.tabla_símbolos,L.h)

Definición L-atribuida (3): Expresiones aritméticas sin recursividad por la izquierda

- *Definición Basada en la Sintaxis con atributos **sintetizados** y **heredados***

Regla de Producción	*Acción semántica*
1 E → T R	R.h = T.valor E.valor = R.s
2 T → **(** E **)**	T.valor = E.valor
3 T → **número**	T.valor = número.valor_léxico
4 T → **identificador**	T.valor = identificador. valor
5 R → + T R_1	R_1.h = R.h + T.valor R.s = R1.s
6 R → - T R_1	R1.h = R.h - T.valor R.s = R1.s
7 R → ε	R.s = R.h

Esquema de Traducción (1): expresiones aritméticas

- ***Sólo se utilizan atributos sintetizados***

---

1. S → E {Imprimir (E.valor)}

2. E → T {E.valor = T.valor}
3. E → E_1 + T {E.valor = E_1.valor + T.valor}

4. T → F {T.valor = F.valor}
5. T → T_1 * F {T.valor = T_1.valor * F.valor }

6. F → **(** E **)** {F.valor = E.valor }
7. F → **número** {F.valor = número.valor_lógico}
8. F → **identificador** {F.valor = identificador.valor}

Esquema de Traducción (2): declaración de variables en C

- ***Se utilizan atributos heredados y sintetizados***

1. D → T { L.h = T. Tipo } **L** ;
2. T → **int** {T.tipo = entero }
3. T → **float** {T.tipo = real }
4. L → **identificador**
   {Añadir_tipo(identificador.tabla_símbolos,L.h)}
5. L → **{**L_1.h = L.h **}** L_1 **, identificador**
   {Añadir_tipo(identificador.tabla_símbolos,L.h)}

Esquema de Traducción (3): expresiones aritméticas sin recursividad por la izquierda

- ***Se utilizan atributos heredados y sintetizados***

---

1. E → T {R.h = T.valor} R {E.valor = R.s}

2. T → (E) {T.valor = E.valor}
3. T → **número** {T.valor = número.valor_léxico}
4. T → **identificador** {T.valor = identificador.valor}

5. R → + T {R_1.h = R.h + T.valor} R_1 {R.s = R_1.s}
6. R → - T {R_1.h = R.h - T.valor} R_1 {R.s = R_1.s}
7. R → ε {R.s = R.h}

Traductor Descendente Predictivo: Tabla predictiva correspondiente al esquema de traducción (3)

	+	**-**	**(**	**)**	**número**	**identificador**	$
E			1		1	1	
T			2		3	4	
R	5	6		7			7

Traductor Descendente Predictivo del esquema de traducción (3)

```
Función E ( ): entero
variables
        entero Tvalor  {Atributo sintetizado de T}
               Rh      {Atributo heredado de R}
               Rs      {Atributo sintetizado de R}

inicio
        si  ( (TOKEN = '(') o (TOKEN = número) o (TOKEN = identificador)
                entonces
                        Tvalor = T()
                        Rh  = Tvalor
                        Rs  = R(Rh)   {R tiene un atributo heredado}

                        {Valor del atributo sintetizado de E: E.valor}
                        Devolver Rs

                        {Se puede simplificar escribiendo Devolver R(T()) }
                si no
                        Error(falta paréntesis izquierdo,  número o  identificador)
        fin_si
fin
```

```
Función T ( ): entero
variables
        entero   Evalor                {Atributo sintetizado de E}
                 número_valor          {Atributo sintetizado de número}
                 identificador_valor   {Atributo sintetizado de identificador}
inicio
        si ( TOKEN = '(' )
           entonces
              avanzar (TOKEN)
              Evalor = E()
              si  ( TOKEN = ')' )
                    entonces   avanzar(TOKEN)
                         {Valor del atributo sintetizado de T: T.valor}
                          Devolver Evalor
                    si no
                        Error (falta paréntesis derecho)
                    fin_si
              si no
                    si (TOKEN = número)
                       entonces   número_valor = valor(TOKEN)
                                  avanzar(TOKEN)

                                  {Valor del atributo sintetizado de T: T.valor}
                                  Devolver número_valor
                         si no
                                  si (TOKEN = identificador)
                                    entonces
                                       identificador_valor = valor(TOKEN)
                                      avanzar(TOKEN)

                                  {Valor del atributo sintetizado de T: T.valor}
                                      Devolver identificador_valor
                                    si no
                                            Error (falta identificador, número o
                                                   paréntesis izquierdo)
                                     fin_si
                     fin_si
        fin_si
fin
```

Función R (entero** H**): entero

{H es el argumento correspondiente al atributo heredado de R}

variables
entero *Tvalor {Atributo sintetizado de T}*
Rh1 {Atributo heredado de R1}
Rs1{Atributo sintetizado de R1}

inicio
si *(TOKEN = '+')*
entonces
avanzar (TOKEN)
Tvalor = T()
Rh1 = H + Tvalor
Rs1 = R (Rh1)

Devolver *Rs1 {Valor del atributo sintetizado de R: R.s}*

{Se puede simplificar como ***Devolver*** *R(H + T())}*
si no
si *(TOKEN = '-')*
entonces
avanzar (TOKEN)
Tvalor = T()
Rh1 = H - Tvalor
Rs1 = R (Rh1)
{Valor del atributo sintetizado de R: R.s}
Devolver *Rs1*
{Se puede simplificar como ***Devolver*** *R(H - T())}*

si no
{Valor del atributo sintetizado de R: R.s}
Devolver *H;*

{Se ha utilizado la regla ***R*** $\rightarrow \varepsilon$ *para los demás símbolos terminales, aunque en un principio sólo se podía aplicar a ")" y $}*
fin_si
fin_si
fin

Definición S-atribuida (4)

- *Se ha modificado la definición S-atribuida (1) para utilizar **"la pila valor"** y realizar una evaluación **ascendente***
- *Se pueden conocer "las posiciones" de los valores*

	Regla de Producción	***Acción semántica***
1	S → E	Imprimir (valor[cima])
2	E → T	valor[nueva-cima] = valor [cima] **(superflua)**
3	E → E_1 + T	valor[nueva-cima] = valor [cima – 2] + valor[cima]
4	T → F	valor[nueva-cima] = valor [cima] **(superflua)**
5	T → T_1 * F	valor[nueva-cima] = valor [cima – 2] * valor[cima]
6	F → **(** E **)**	valor[nueva-cima] = valor [cima – 1]
7	F → **número**	valor[nueva-cima] = valor [cima] **(superflua)**
8	F → **identificador**	valor[nueva-cima] = valor [cima] **(superflua)**

Notas:

- Al desplazar, **nueva-cima = cima + 1**
- Al reducir, si A → α es la regla de producción utilizada entonces

nueva-cima = cima - | α | + 1

Tabla de Análisis Sintáctico LALR de la Definición S-atribuida (4)

	ACCIÓN							IR A			
	id	**número**	**+**	*****	**(**	**)**	**$**	**S**	**E**	**T**	**F**
0	d 7	d 6			d 5			1	2	3	4
1							Aceptar				
2	r 1	r 1	d 8	r 1	r 1	r 1	r 1				
3	r 2	r 2	r 2	d 9	r 2	r 2	r 2				
4	r 4	r 4	r 4	r 4	r 4	r 4	r 4				
5	d 7	d 6			d 5				10	3	4
6	r 7	r 7	r 7	r 7	r 7	r 7	r 7				
7	r 8	r 8	r 8	r 8	r 8	r 8	r 8				
8	d 7	d 6			d 5					11	4
9	d 7	d 6			d 5						12
10			d 8			d 13					
11	r 3	r 3	r 3	d 9	r 3	r 3	r 3				
12	r 5	r 5	r 5	r 5	r 5	r 5	r 5				
13	r 6	r 6	r 6	r 6	r 6	r 6	r 6				

*Traduccción Ascendente de la expresión: 3 * (4 + 5)*

PILA de análisis sintáctico	*ENTRADA*	*ACCIÓN*	*VALOR*
0	**número** * (**número** + **número**) $	d 6	
0 **número** 6	* (**número** + **número**) $	r 7 F → **número**	3
0 **F** 4	* (**número** + **número**) $	r 5 T → F	3
0 T 3	* (**número** + **número**) $	d 9	3
0 T 3 * 9	(**número** + **número**) $	d 5	3 -
0 T 3 * 9 (5	**número** + **número**) $	d 6	3 - -
0 T 3 * 9 (5 **número** 6	+ **número**) $	r 7 F → **número**	3 - - 4
0 T 3 * 9 (5 **F** 4	+ **número**) $	r 5 T → F	3 - - 4
0 T 3 * 9 (5 **T** 3	+ **número**) $	r 2 E → T	3 - - 4
0 T 3 * 9 (5 **E** 10	+ **número**) $	d 8	3 - - 4
0 T 3 * 9 (5 E 10 + 8	**número**) $	d 6	3 - - 4 -
0 T 3 * 9 (5 E 10 + 8 **número** 6	) $	r 7 F → **número**	3 - - 4 - 5
0 T 3 * 9 (5 E 10 + 8 **F** 4	) $	r 5 T → F	3 - - 4 - 5
0 T 3 * 9 (5 **E** 10 **+** 8 **T** 11	) $	r 3 E → E + T	3 - - 4 - 5
0 T 3 * 9 (5 E 10	) $	d 13	3 - - 9
0 T 3 * 9 **(** 5 **E** 10 **)** 13	$	r 6 F → (E)	3 - - 9 -
0 T 3 * 9 **F** 12	$	r 5 T → T * F	3 - 9
0 **T** 3	$	r 3 E → T	27
0 **E** 2	$	r 1 S → E	27
0 S 1	$	**Aceptar**	**27**

Esquema de traducción (4): uso de acciones intercaladas

- Este esquema de traducción permite **transformar**
  - expresiones aritméticas escritas con notación **"infija"**: 10 - (2 + 3)
  - a expresiones escritas con notación **"postfija":** 10 (2 3 +) -

---

1. E → T R
2. R → **+** T { *imprimir("+")* } R
3. R → **-** T { *imprimir("-")* } R
4. R → ε
5. T → **(** { *imprimir("(")* } E **)** { *imprimir(")")* }
6. T → **número** {*imprimir(*número.valor_léxico) }
7. T → **identificador** {*imprimir(*identificador.valor) }

Esquema de Traducción (5)

- *Se han usado de **"símbolos marcadores"** para eliminar las "acciones intercaladas" del esquema de traducción (4)*

1. E → T R
2. R → + T M_1 R
3. R → - T M_2 R
4. R → ε
5. T → **(** M_3 E **)** { *imprimir(")")* }
6. T → **número** {*imprimir(*número.valor_léxico) }
7. T → **identificador** {*imprimir(*identificador.valor) }

8. M_1 → ε { *imprimir("+")* }
9. M_2 → ε { *imprimir("-")* }
10. M_3 → ε { *imprimir("(")* }

Definición L-atribuida (2): declaración de variables en C

- *La gramática subyacente es* ***recursiva por la izquierda***
- *Se indican* ***"las reglas de copia"***

Regla de Producción	***Acción semántica***
1 D → T L ;	L.h = T. tipo ***(regla de copia)***
2 T → **int**	T.tipo = entero
3 T →**float**	T.tipo = real
4 L →**identificador**	Añadir_tipo(identificador.tabla_símbolos,L.h)
5 L → L_1 , **identificador**	L_1.h = L.h ***(regla de copia)*** Añadir_tipo(identificador. tabla_símbolos,L.h)

Definición L-atribuida (5): declaración de variables en C

- *Se ha modificado la definición L-atribuida (2) para utilizar **"la pila valor"** y realizar una evaluación **ascendente**.*
- *Se pueden conocer **"las posiciones"** de los valores.*
- *La gramática subyacente es **recursiva por la izquierda.***

Regla de Producción	*Acción semántica*
1 D → T L **;**	
2 T → **int**	valor[nueva_cima] = entero (superflua)
3 T → **float**	valor[nueva_cima] = real (superflua)
4 L → **identificador**	Añadir_tipo(valor[cima],valor[cima-1])
5 L → L_1 **, identificador**	valor[cima-2] = valor[cima-3] (superflua) Añadir_tipo(valor[cima],valor[cima-3])

Tabla de Análisis Sintáctico LALR de la Definición L-atribuida (5)

	;	int	float	identificador	,	$		D	T	L
0		d 3	d 4					1	2	
1						ACEPTAR				
2				d 6						5
3	r 2	r 2	r 2	r 2	r 2	r 2				
4	r 3	r 3	r 3	r 3	r 3	r 3				
5	d 7				d 8					
6	r 4	r 4	r 4	r 4	r 4	r 4				
7	r 1	r 1	r 1	r 1	r 1	r 1				
8				d 9						
9	r 5	r 5	r 5	r 5	r 5	r 5				

Definición L-atribuida (6): declaración de variables en C

- *Se ha modificado la Definición L-atribuida (5) para que la gramática subyacente sea* ***recursiva por la derecha***
- *Las posiciones de los valores de los atributos* ***no son predecibles***

	Regla de Producción	***Acción semántica***
1	D → T L ;	L.h = T. tipo ***(regla de copia)***
2	T → **int**	T.tipo = entero
3	T → **float**	T.tipo = real
4	L → **identificador**	Añadir_tipo(identificador.tabla_símbolos,L.h)
5*	L → **identificador ,** L_1	Añadir_tipo(identificador.tabla_símbolos,L.h) L_1.h = L.h

Tabla de Análisis Sintáctico LALR de la Definición L-atribuida (6)

	;	int	float	identificador	,	$		D	T	L
0		d 3	d 4					1	2	
1						ACEPTAR				
2				d 6						5
3	r 2	r 2	r 2	r 2	r 2	r 2				
4	r 3	r 3	r 3	r 3	r 3	r 3				
5	d 7									
6	r 4	r 4	r 4	r 4	d 8	r 4				
7	r 1	r 1	r 1	r 1	r 1	r 1				
8				d 6						9
9	r 5	r 5	r 5	r 5	r 5	r 5				

Definición L-atribuida (7): declaración de variables en C

- *Se han introducido **"símbolos marcadores"** en la Definición L-atribuida (6)*
- *Las posiciones de los valores de los atributos **sí son predecibles***

	Regla de Producción	***Acción semántica***
1*	D → T M_1 L ;	M_1.h = T. tipo L.h = M_1.s
2	T → **int**	T.tipo = entero
3	T → **float**	T.tipo = real
4	L → **identificador**	Añadir_tipo(identificador.tabla_símbolos,L.h)
5*	L → **identificador ,** M_2 L_1	M_2.h = L.h L_1.h = M_2.s Añadir_tipo(identificador.tabla_símbolos,L.h)
6*	M_1 → ε	M_1.s = M_1.h
7*	M_2 → ε	M_2.s = M_2.h

- **Nota importante**: no se debe confundir el significado de los subíndices
  - M_1 y M_2 son símbolos diferentes
  - L_1 representa una nueva aparición del símbolo L

Definición L-atribuida (8): declaración de variables en C

- *Se ha modificado la definición L-atribuida (7) para utilizar "la pila valor" y realizar una evaluación* ***ascendente****.*
- *Se pueden conocer* ***"las posiciones"*** *de los valores.*

	Regla de Producción	***Acción semántica***
1*	D → T M_1 L **;**	
2	T → **int**	valor[nueva_cima] = entero (superflua)
3	T → **float**	valor[nueva_cima] = real (superflua)
4	L → **identificador**	Añadir_tipo(valor[cima],valor[cima-1])
5*	L → **identificador ,** M_2 L_1	Añadir_tipo(valor[cima-3],valor[cima-1])
6*	M_1 → ε	valor[nueva_cima] = valor[cima]
7*	M_2 → ε	valor[nueva_cima] = valor[cima-2]

Tabla de Análisis Sintáctico LALR de la Definición L-atribuida (8)

	;	**int**	**float**	**identificador**	**,**	**$**	D	T	L	M_1	M_2
0		d 3	d 4				1	2			
1						ACEPTAR					
2	r 6	r 6	r 6	r 6	r 6	r 6				5	
3	r 2	r 2	r 2	r 2	r 2	r 2					
4	r 3	r 3	r 3	r 3	r 3	r 3					
5				d 7					6		
6	d 8										
7	r 4	r 4	r 4	r 4	d 9	r 4					
8	r 1	r 1	r 1	r 1	r 1	r 1					9
9	r 7	r 7	r 7	r 7	r 7	r 7					10
10				d 7					11		
11	r 5	r 5	r 5	r 5	r 5	r 5					

Definición S-atribuida (9): declaración de variables en C

- *Sólo se utilizan atributos **sintetizados***
- ***Inconveniente**: la gramática **"oculta"** la sintaxis del lenguaje*

	Regla de Producción	*Acción semántica*
1	D → L **identificador** ;	Añadir_tipo(identificador.tabla_símbolos,L.s)
2	L → L_1 **identificador ,**	Añadir_tipo(identificador.tabla_símbolos, L_1.s) L.s = L_1.s
3	L → T	L.s = T.tipo
4	T → **int**	T.tipo = entero
5	T → **float**	T.tipo = real

Definición basada en la sintaxis (10): declaración de variables en Pascal

- *Se utilizan atributos heredados y sintetizados.*
- ***Esta definición no es L-atribuida***
- *La gramática "refleja con claridad" la sintaxis del lenguaje Pascal*

	Regla de Producción	*Acción semántica*
1	D → L : T ;	L.h = T.tipo (No L-atribuida)
2	L → **identificador**	Añadir_tipo(identificador.tabla_símbolos, L.h)
3	L → L_1 **, identificador**	Añadir_tipo(identificador.tabla_símbolos, L.h) L_1.h = L.h
4	T → **integer**	T.tipo = entero
5	T → **real**	T.tipo = real

Definición S-atribuida (11): declaración de variables en Pascal

- *Sólo se utlizan atributos sintetizados.*
- *La gramática **"oculta"** la sintaxis del lenguaje Pascal*

	Regla de Producción	*Acción semántica*
1	D → **identificador** L	Añadir_tipo(identificador.tabla_símbolos, L.s)
2	L → , **identificador** L_1	Añadir_tipo(identificador. tabla_símbolos,L_1.s) L.s = L_1.s
3	L → : T **;**	L.s = T.tipo
4	T → **integer**	T.tipo = entero
5	T → **real**	T.tipo = real

Definición S-atribuida (12): declaración de variables en Pascal

- *Se ha modificado la definición S-atribuida (11) para utilizar **"la pila valor"** y realizar una evaluación **ascendente**.*
- *Se pueden conocer **"las posiciones"** de los valores.*

	Regla de Producción	*Acción semántica*
1	D → **identificador** L	Añadir_tipo(valor[cima-1],valor[cima])
2	L → , **identificador** L_1	Añadir_tipo(valor[cima-1],valor[cima]) valor[nueva_cima]=valor[cima]
3	L → : T **;**	valor[nueva_cima]=valor[cima]
4	T → **integer**	valor[nueva_cima]=entero (superflua)
5	T → **real**	valor[nueva_cima]=real (superflua)

Tabla de Análisis Sintáctico LALR de la Definición S-atribuida (12)

	identificador	**,**	**:**	**integer**	**real**	**$**		D	L	T
0	d 2							1		
1						ACEPTAR				
2		d 4	d 5						3	
3	r 1	r 1	r 1	r 1	r 1	r 1				
4	d 6									
5				d 8	d 9					7
6		d 4	d 5						10	
7	r 3	r 3	r 3	r 3	r 3	r 3				
8	r 4	r 4	r 4	r 4	r 4	r 4				
9	r 5	r 5	r 5	r 5	r 5	r 5				
10	r 2	r 2	r 2	r 2	r 2	r 2				

Definición S-atribuida (13): Generación de código intermedio para expresiones aritméticas

	Regla de Producción	*Acción semántica*
1	S → **id** = E	S.código = E. código || generar (id.nombre '=' E.variable)
2	E → T	E.variable = T.variable E.código = T.código
3	E → E_1 + T	E.variable = nueva_temporal() E. código = E_1. código || T. código || generar(E.variable '=' E_1.variable '+' T.variable)
4	T → F	T. variable = F. variable T. código = F. código
5	T → T_1 * F	T.variable = nueva_temporal() T. código = T_1. código || F. código || generar(T.variable '=' T_1.variable '*' F.variable)
6	F → **(** E **)**	F. variable = E. variable F. código = E. código
7	F → **número**	F.variable = nueva_temporal() F. código = generar(F.variable '=' número.valor_léxico)
8	F → **id**	F.variable = id.nombre F. código = ''

Definición S-atribuida (13): Generación de código intermedio para expresiones aritméticas

Ejemplo: **a = 3 * b + 2**

- Se genera el código

```
t1 =  3
t2 = t1 * b
t3 = 2
t4 = t2 + t3
a = t4
```

- Este código se puede optimizar:
  - Paso 1: eliminación de asignación de constantes a variables

```
t2 = 3 * b
t4 = t2 + 2
a = t4
```

  - Paso 2: eliminación de propagación de copias

```
t2 = 3 * b
a = t2 + 2
```